Department of Computer Science
Series of Publications A
Report A-2014-1

Graph and Hypergraph Decompositions for Exact Algorithms

Janne H. Korhonen

To be presented, with the permission of the Faculty of Science of the University of Helsinki, for public criticism in the auditorium of the Arppeanum building, Snellmaninkatu 3, on January 16th, 2014, at 12 o'clock noon.

University of Helsinki
Finland

Supervisors
Mikko Koivisto, University of Helsinki, Finland
Petteri Kaski, Aalto University, Finland

Pre-examiners
Markus Bläser, Saarland University, Germany
Ryan Williams, Stanford University, United States of America

Opponent
Dieter Kratsch, University of Lorraine, France

Custos
Esko Ukkonen, University of Helsinki, Finland

Contact information

Department of Computer Science
P.O. Box 68 (Gustaf Hällströmin katu 2b)
FI-00014 University of Helsinki
Finland

Email address: info@cs.helsinki.fi
URL: http://www.cs.helsinki.fi/
Telephone: +358 9 1911, telefax: +358 9 191 51120

ISSN 1238-8645
ISBN 978-952-10-9638-9 (paperback)
ISBN 978-952-10-9639-6 (PDF)
Computing Reviews (1998) Classification: F.1.3, F.2.1, F.2.2, G.2.1, G.2.2
Helsinki 2013
Unigrafia

Graph and Hypergraph Decompositions for Exact Algorithms

Janne H. Korhonen

Department of Computer Science
P.O. Box 68, FI-00014 University of Helsinki, Finland
janne.h.korhonen@helsinki.fi
http://www.cs.helsinki.fi/janne.h.korhonen/

PhD Thesis, Series of Publications A, Report A-2014-1
Helsinki, December 2013, 62 + 66 pages
ISSN 1238-8645
ISBN 978-952-10-9638-9 (paperback)
ISBN 978-952-10-9639-6 (PDF)

Abstract

This thesis studies exact exponential and fixed-parameter algorithms for hard graph and hypergraph problems. Specifically, we study two techniques that can be used in the development of such algorithms: (i) *combinatorial decompositions* of both the input instance and the solution, and (ii) evaluation of *multilinear forms over semirings*.

In the first part of the thesis we develop new algorithms for graph and hypergraph problems based on techniques (i) and (ii). While these techniques are independently both useful, the work presented in this part is largely characterised by their joint application. That is, combining results from different pieces of the decompositions often takes the from of multilinear form evaluation task, and on the other hand, decompositions offer the basic structure for dynamic-programming-style algorithms for the evaluation of multilinear forms.

As main positive results of the first part, we give algorithms for three different problem families. First, we give a fast evaluation algorithm for linear forms defined by a *disjointness matrix* of small sets. This can be applied to obtain faster algorithms for counting maximum-weight objects of small size, such as k-paths in graphs. Second, we give a general framework for exponential-time algorithms for finding *maximum-weight subgraphs of bounded tree-width*, based on the theory of tree decompositions. Besides

basic combinatorial problems, this framework has applications in learning Bayesian network structures. Third, we give a fixed-parameter algorithm for finding *unbalanced vertex cuts*, that is, vertex cuts that separate a small number of vertices from the rest of the graph.

In the second part of the thesis we consider aspects of the complexity theory of linear forms over semirings, in order to better understand technique (ii). Specifically, we study how the presence of different *algebraic catalysts* in the ground semiring affects the complexity. As the main result, we show that there are linear forms that are easy to compute over semirings with idempotent addition, but difficult to compute over rings, unless the strong exponential time hypothesis fails.

Computing Reviews (1998) Categories and Subject Descriptors:
F.1.3 [Computation by Abstract Devices] Complexity Measures and Classes – relations among complexity measures
F.2.1 [Analysis of Algorithms and Problem Complexity] Numerical Algorithms and Problems – computation of transforms
F.2.2 [Analysis of Algorithms and Problem Complexity] Nonnumerical Algorithms and Problems – computations on discrete structures
G.2.1 [Discrete Mathematics] Combinatorics – combinatorial algorithms, counting problems
G.2.2 [Discrete Mathematics] Graph Theory

General Terms:
algorithms, design, theory

Additional Key Words and Phrases:
algebraic circuits, exact algorithms, fixed-parameter algorithms, semigroups and semirings, separators in graphs, strong exponential time hypothesis, tree-width

Acknowledgements

A PhD thesis is not, despite the stereotypical image, a product of a solitary student cloistered in his or her research chamber, but rather a dynamical and social endeavour – well, except for the part where one actually *writes* the thing – and indeed, this thesis would not have become reality if not for the support of many brilliant and wonderful people I have had the pleasure to work with. Thus, some thanks are in order.

First and foremost, I want to thank my advisors Petteri Kaski and Mikko Koivisto. Working with them has given me an opportunity to study extremely interesting topics, and I feel I could not have made much better a choice when deciding where to do my PhD studies. I also owe a lot to Esko Ukkonen, who was my supervisor during my years as an undergraduate research assistant at the department. His support has been invaluable during my career, and maybe even more importantly, he taught me to appreciate also the practical side of algorithmics.

My home for the past couple of years has been the New Paradigms in Computing group at the Helsinki Institute of Information Technology HIIT. The other senior researchers at the group, besides my advisors – Matti Järvisalo, Jukka Suomela and Valentin Polishchuk – have also been immensely supportive; in particular, Matti's relaxed attitude to the hardships of academic life has helped me to cope with that side of this business. My fellow PhD student Joel Rybicki has been one of best friends during these years; he, Mika Göös, Juho Hirvonen and Juhana Laurinharju have formed my default social context here at the Kumpula campus, and we've had lots of scientific and non-scientific fun together. I also want to thank the other 'juniors' at the New Paradigms group for the great atmosphere and fruitful discussions; thanks go to Esther Galbrun, Juho-Kustaa Kangas, Tuomo Lempiäinen, Teppo Niinimäki and Pekka Parviainen.

Before I started my PhD studies, I was part of the bioinformatics research community at the Department of Computer Science of the University of Helsinki; these years were instrumental to my growth as a young researcher. In particular, I thank Pasi Rastas for his excellent mentoring. I also want

to thank Markus Heinonen, Teemu Kivioja, Jussi Kollin, Veli Mäkinen, François Nicolas, Kimmo Palin, Esa Pitkänen, Leena Salmela and Jarkko Toivonen for my time in bioinformatics research.

More generally, working at the Department of Computer Science has been immensely enjoyable. In addition to all the people already mentioned, I want to thank Emilia Hjelm, Jyrki Kivinen, Jaakko Kurhila, Matti Luukkainen, Jaakko Nurro, Sini Ruohomaa and Jouni Sirén; Arto Vihavainen for his crazy shenanigans in particular; Esa Junttila for his role in 'Friday events'; and the IT staff, in particular Jani Jaakkola, Pekka Niklander and Pekka Tonteri, for the excellent IT services. The financial support from Hecse, Algodan and HIIT is also much appreciated.

During the late autumn of 2012, I visited the Algorithms Group at the University of Bergen for two months; I thank my co-authors Fedor Fomin and Petr Golovach for their hospitality during the visit. Igor Sergeev's many comments on circuit complexity topics have also been invaluable. Furthermore, I thank the pre-examiners of this thesis and the anonymous referees of the papers for valuable comments and feedback.

I am thankful for my many awesome friends outside the computer science community, as they have provided a much-needed counterpoint to the days I have spent buried under piles of research papers. Thus, many thanks to all my friends from Alter Ego, Lambda, Matrix, Ropecon and the University of Helsinki in general; I unfortunately do not have room to list you all here. Still, special thanks go to Henriikka Hallamaa, Aleksi Hankalahti, Miska Husgafel, Tommi Hyvönen, Joel Kaasinen, Jenni Kauppinen, Olli-Kalle Kauppinen, Sami Ketola, J Matias Kivikangas, Joonas Kirsi, Maija Korhonen, Emi Maeda, Kristiina Mannermaa, Katri Saarelma, Santeri Virtanen and Olli Wilkman.

Finally, I want to thank my parents Helinä Ikonen and Matti Korhonen for their support during all my long years at the University of Helsinki.

Helsinki, 28th of November, 2013
Janne H. Korhonen

Contents

Original publications

This thesis is based on the following original publications, which are referred to in the text as Papers I–V. The papers are reprinted at the end of this thesis.

I. Petteri Kaski, Mikko Koivisto, Janne H. Korhonen, and Igor S. Sergeev. Fast monotone summation over disjoint sets. Submitted.

II. Petteri Kaski, Mikko Koivisto, and Janne H. Korhonen. Fast monotone summation over disjoint sets. In *7th International Symposium on Parameterized and Exact Computation (IPEC 2012)*, pages 159–170. Springer, 2012. doi:10.1007/978-3-642-33293-7_16.

III. Janne H. Korhonen and Pekka Parviainen. Exact learning of bounded tree-width Bayesian networks. In *16th International Conference on Artificial Intelligence and Statistics (AISTATS 2013)*, pages 370–378. JMLR, 2013.

IV. Fedor V. Fomin, Petr A. Golovach, and Janne H. Korhonen. On the parameterized complexity of cutting a few vertices from a graph. In *38th International Symposium on Mathematical Foundations of Computer Science (MFCS 2013)*, pages 421–432. Springer, 2013. doi:10.1007/978-3-642-40313-2_38.

V. Matti Järvisalo, Petteri Kaski, Mikko Koivisto, and Janne H. Korhonen. Finding efficient circuits for ensemble computation. In *15th International Conference on Theory and Applications of Satisfiability Testing (SAT 2012)*, pages 369–382. Springer, 2012. doi:10.1007/978-3-642-31612-8_28.

Chapter 1

Introduction

In this chapter, we introduce the central themes and techniques of this thesis and summarise the contributions, as well as provide background on the exact algorithmics of hard problems.

1.1 Background

One of the fundamental goals of computer science is the development of efficient *algorithms*. From a theoretical perspective, algorithms are considered to be efficient if they run in *polynomial time*. Alas, there are also problems that do not have polynomial-time algorithms; for instance, NP-hard problems fall into this category, unless the common assumption of $\mathrm{P} \neq \mathrm{NP}$ fails [66, 67]. Regardless, we still often want to solve such problems, so it makes sense to ask what we can do despite intractability.

Exponential algorithms. *Exact exponential algorithmics* [59, 60] considers the development of techniques that aim to avoid brute force search over the whole solution space while still accepting that the running time of the algorithm will have exponential dependence on the input size n – essentially, this means developing less exponential algorithms. Over the years, a wide variety of techniques have been successfully applied for this purpose, such as branching [126], dynamic programming [7, 8, 73] and meet-in-the-middle [74, 118].

More formally, when considering a problem with nominally super-exponential solution space, the first question is whether we can still get the dependence down to single-exponential level, that is, construct algorithms with running time $c^n n^{O(1)}$ for some constant $c > 1$. Furthermore, we can then ask how small we can make the constant c in the running time. For

some problems, such as the maximum independent set problem, the constant c can be made strictly smaller than 2, while at the other end of the spectrum lie problems such as CNF-SAT and related problems, for which improving upon $c = 2$ is a longstanding open problem [32, 48, 76, 114].

One trend in exact exponential algorithmics that we make a special note of is the use of *algebrisation*, that is, casting a combinatorial problem as a task of evaluating an expression or function over a suitable algebraic structure. Efficient algorithms for the algebraic task can then be obtained by exploiting the properties of the algebraic structure, such as the existence of additive inverses or roots of unity. For example, Hamiltonian path and cycle problems can be solved in polynomial space using inclusion–exclusion techniques [83, 90]; more recent examples of algebrisation include a matrix multiplication based algorithm for MAX-2-SAT [133], subset convolution for k-colouring and related problems [13, 17], polynomial interpolation/evaluation for Tutte polynomial [15] and determinant sums for Hamiltonian cycles [11].

Parameterised algorithms. The theory of *parameterised algorithms and complexity* [53, 57, 109] provides a way to study the exponential dependence in the running time from a more structural perspective. The basic idea is to isolate the exponential complexity into some parameter k that is independent of n. Problems for which this is possible are called *fixed-parameter tractable*; more precisely, a problem with parameter k is fixed-parameter tractable if there is an algorithm with a running time of the form $f(k)n^{O(1)}$ for some function f. For example, when the parameter k is the size of the solution, techniques such as bounded-depth branching [52], colour-coding [1] and iterative compression [116] have been applied to obtain fixed-parameter algorithms.

The parameter can also describe the structure of the input; a prominent example also relevant to the work of this thesis is the *tree-width* of a graph [117], which measures how tree-like a graph is. Many combinatorial problems are fixed-parameter tractable when parameterised by the tree-width w of the input graph [3, 21, 24]; essentially, one can take (a) a polynomial-time algorithm on trees and (b) an exact exponential algorithm on general graphs, and run these on a *tree decomposition* of the input graph to obtain an algorithm with running time $f(w)n^{O(1)}$. Typically the exact exponential algorithms used within this scheme use dynamic programming, but recently algebrisation has been employed to obtain better dependence on tree-width w [25, 128].

Of course, not all parameterised problems are fixed-parameter tractable; for example, W[1]-hard problems are assumed to not have fixed-parameter

algorithms much in the same way as NP-hard problems are assumed not to have polynomial-time algorithms [52, 57]. Still, many W[1]-hard parameterised problems have natural algorithms with running time $n^{k+O(1)}$ that simply do brute-force enumeration of all subsets of size at most k. For such problems, we can ask whether we can obtain algorithms with a running time of the form $n^{ck+O(1)}$ for $c < 1$, for instance by applying fast matrix multiplication [108, 130] or the counting-in-halves [16] type of techniques.

Other approaches. While we are interested in exact algorithms and worst-case complexity, there are also other approaches to dealing with intractability that do not quite fall within these two criteria. *Approximation algorithms* [131] consider finding solutions that are not necessarily optimal, but are still guaranteed to be within a known factor of the optimum; in particular, this can often be achieved in polynomial time. On the other hand, algorithms with exponential worst-case complexity may still be feasible in practice, especially when solving real-world instances; for example, carefully implemented Boolean satisfiability solvers [10] have proven to be a useful tool for solving instances of hard problems in practice [9, 78, 86].

1.2 Outline

This thesis contributes to a thread of research aimed at developing and understanding new techniques for exact algorithms for hard graph and hypergraph problems. This work is centred on two main techniques, namely *combinatorial decompositions* and evaluation of *multilinear forms over semirings,* which we will discuss in detail in Section 1.2.1.

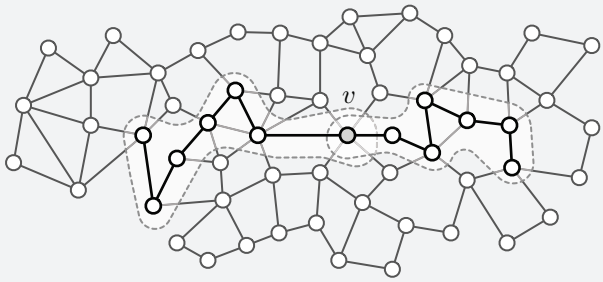
In the first part of this thesis, we develop new algorithms for combinatorial problems based on combinatorial decompositions and evaluation of multilinear forms over semirings; an overview of these results is given in Section 1.2.2. The second part of this thesis considers the complexity theory of multilinear form evaluation in the presence of different types of *algebraic catalysts*, as we will discuss in Section 1.2.3.

1.2.1 Techniques

Combinatorial decompositions. The first technique we repeatedly employ is the development of suitable *combinatorial decompositions*. We consider (i) *decompositions of instances* and (ii) *decompositions of solutions*:

(i) Given an instance X, we decompose the instance into sub-instances $X_1, X_2, \ldots, X_m$, solve the original problem or some variant of it on

Example 1.1: Decomposing k-paths into halves



Let $G = (V, E)$ be a graph, $v \in V$ and let k be an even integer. To find a path with k edges and a middle point v in the graph G, it suffices to find two paths with $k/2$ edges and an end point v such that the vertex sets of these paths only intersect at v; the illustration above shows a 12-path decomposed into two 6-paths. This idea will be expanded upon in Example 1.3.

each of the instances X_i, and then assemble the solution for X from the solutions to instances X_i.

(ii) When the goal is to find a solution Y, we first find some simpler structures $Y_1, Y_2, \ldots, Y_m$, and then we assemble the desired solution from the decomposition pieces Y_i.

Depending on the problem at hand, we either use (i) or (ii) or both to construct efficient algorithms. The decompositions can also be nested or recursive, that is, each piece of the decomposition is again decomposed into smaller pieces.

▶ *See Example 1.1.*

Multilinear forms over semirings. Our second technique is the evaluation of *multilinear forms over semirings*. The simplest example of this type of tasks is the evaluation of *linear forms* defined by a Boolean matrix. Let $(S, \oplus)$ be a commutative semigroup, that is, a set S equipped with binary operation $\oplus$ that follows the usual rules of addition but is not necessarily invertible. Given a Boolean matrix $A \in \{0,1\}^{m \times n}$ and an input vector $x \in S^n$, the task is to compute an output vector $y \in S^m$ defined by

$$y_i = \bigoplus_{j\colon A_{ij}=1} x_j \,.$$

In the general case we operate over a semiring $(S, \oplus, \odot)$, that is, we also have a not necessarily invertible multiplication operation $\odot$ that distributes over addition. Given a $(d+1)$-dimensional tensor $T \in S^{m \times n_1 \times n_2 \times \cdots \times n_d}$ and d input vectors

$$x_i = (x_{i,1}, x_{i,2}, \ldots, x_{i,n_d}) \in S^{n_i} \qquad \text{for } i = 1, 2, \ldots, d,$$

the task is to compute an output vector $y \in S^m$ defined by

$$y_i = \bigoplus_{j_1=1}^{n_1} \bigoplus_{j_2=1}^{n_2} \cdots \bigoplus_{j_d=1}^{n_d} T_{i,j_1,j_2,\ldots,j_d} \bigodot_{k=1}^{d} x_{k,j_k} .$$

The use of multilinear forms usually becomes relevant when we consider counting and optimisation problems, where we have to consider – at least in some sense – all possible solutions instead of finding just one. The basic pattern of usage is that we pick (a) a suitable set of multilinear forms and (b) a suitable semiring, so that after evaluating the multilinear forms we can recover the solution from the output vector. This often requires some pre- and post-processing in addition to the evaluation itself.

In this thesis, we generally consider cases where the semiring of choice does not have full ring structure; this is usually necessitated by applications where rings do not have sufficient expressive power. For example, the tropical semiring $(\mathbb{R} \cup \{-\infty\}, \max, +)$ used in optimisation algorithms has non-invertible addition. Working over semirings in particular means that algebraic techniques such as inclusion–exclusion and reduction to matrix multiplication cannot be used, as these rely on the existence of the additive inverses in the ground ring.

► *See Example 1.2.*

1.2.2 Algorithm design

Overview. Our main goal is the development of efficient algorithms, to which end we employ combinatorial decompositions and evaluation of multilinear forms. In particular, the joint application of these techniques will appear frequently;

1. combining results from different pieces of the decompositions often takes the form of a multilinear form evaluation task, and
2. decompositions offer the basic structure for dynamic-programming-style algorithms for the evaluation of multilinear forms.

► *See Example 1.3.*

Example 1.2: Complement of the identity matrix

Consider the Boolean complement of the $n \times n$ identity matrix, that is, the $n \times n$ matrix $\bar{I}$ with entries

$$\bar{I}_{ij} = \begin{cases} 1, & \text{if } i \neq j, \text{ and} \\ 0, & \text{if } i = j. \end{cases}$$

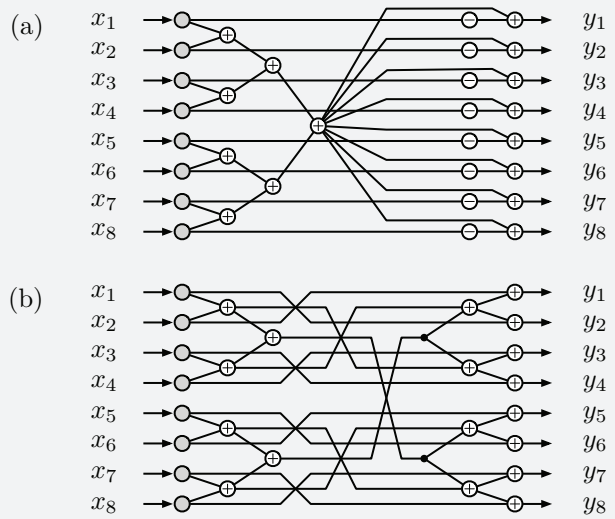
The matrix $\bar{I}$ defines the linear forms that transform input $x \in S^n$ to output $y \in S^n$ by

$$y_i = \bigoplus_{j \neq i} x_j \,. \tag{1.1}$$

Over a commutative group $(G, +)$, the straightforward way of computing the linear forms (1.1) uses $2n-1$ binary additions and n negations; we first compute the sum $\sum_{j=1}^{n} x_j$, and then obtain the outputs as

$$y_i = \Big(\sum_{j=1}^{n} x_j\Big) - x_i \,.$$

This is presented in circuit form in illustration (a), below. However, the construction uses negations and thus cannot be employed over semigroups. Still, it is possible to compute (1.1) over any commutative semigroup using $3n-6$ additions in total; the construction is illustrated in (b).



Papers I and II. Papers I and II are centred around the set of linear forms defined by the *disjointness matrix* of sets of small size. That is, we consider matrix D indexed by subsets of a ground set U. The rows are indexed by subsets of size at most q and the columns by subsets of size at most p, with entries

$$D(Y,X) = \begin{cases} 1, & \text{if } Y \cap X = \emptyset, \text{ and} \\ 0, & \text{if } Y \cap X \neq \emptyset. \end{cases}$$

The corresponding linear forms are called the (n,p,q)-*disjoint transform*. That is, given an input value $f(X)$ in a commutative semigroup $(S,\oplus)$ for each subset X of size at most p, the task is to output the values

$$e(Y) = \bigoplus_{X\colon X\cap Y=\emptyset} f(X) \tag{1.2}$$

for each set Y of size at most q. A recursive tiling presented in Paper I gives an algorithm for (n,p,q)-disjoint summation that uses

$$\left[\binom{n}{p} + \binom{n}{q}\right] \cdot \min\{2^p, 2^q\} \cdot n^{O(1)}$$

binary operations in S; the construction of Paper II is slightly less efficient, but extends to a case where the input and output indices are allowed to overlap to a degree.

For most applications it is convenient to extend the (n,p,q)-disjoint summation into the *disjoint product*

$$\Delta(g,f) = \bigoplus_{Y,X\colon X\cap Y=\emptyset} g(Y) \odot f(X)$$

over semirings $(S,\oplus,\odot)$, where the parameters of input functions g and f are sets of size at most q and at most p, respectively. We have

$$\Delta(g,f) = \bigoplus_{Y} g(Y) \odot e(Y)\,,$$

where e is as in (1.2); that is, a fast algorithm for (n,p,q)-disjoint summation also gives a fast algorithm for the disjoint product. In particular, this can be used to reduce running times of form $\binom{n}{k}n^{O(1)}$ to $\binom{n}{k/2}n^{O(1)}$ for example for counting decomposable maximum-weight objects of size k and rectangular matrix permanents over non-commutative semirings.

► *See Section 3.1 for details.*

Example 1.3: Counting in halves

Let us consider the setting of Example 1.1, but instead of just finding a k-path – that is, a path with k edges – we want to count how many k-paths there are in a directed input graph. This problem can be reduced to the evaluation of *disjoint product*

$$\Delta(g, f) = \bigoplus_{X,Y\colon X\cap Y=\emptyset} g(Y) \odot f(X)\,,$$

where X and Y range over vertex sets of size $k/2$ and f and g are input functions. Indeed, for any subset $X \subseteq V \setminus \{v\}$ of size $k/2$, let $f(X)$ and $g(X)$ be the number of $k/2$-paths vertex set $X \cup \{v\}$ starting from v and ending at v, respectively. We now have that $\Delta(g, f)$ is the number of k-paths with middle vertex v.

This idea can be extended to counting maximum-weight k-paths in a graph with edge weights $w\colon E \to \mathbb{Q}$. We will use the semiring

$$\big(\mathbb{N} \times (\mathbb{Q} \cup \{-\infty\}), \vee, \otimes\big)\,,$$

where

$$(c, w) \vee (c', w') = \begin{cases} (c, w) & \text{if } w > w', \\ (c', w') & \text{if } w < w', \\ (c + c', w) & \text{if } w = w', \end{cases}$$

and $(c, w) \otimes (c', w') = (c \cdot c', w + w')$. The value (c, w) can be thought of as a representation of a collection of c objects of weight w; the operation $\vee$ selects the objects with higher weight, and $\otimes$ is analogous to taking a Cartesian product of the two collections. Now let $f_2(X)$ be the maximum weight of a $k/2$-path with vertex set $X \cup \{v\}$ starting from v, and let $f_1(X)$ be the number of such paths. Let $f(X) = (f_1(X), f_2(X))$ and define $g(X)$ analogously for paths ending at v; the value $(c, w) = \Delta(g, f)$ gives the weight w and the number c of maximum-weight k-paths with middle vertex v.

The efficient evaluation of disjoint products is discussed in detail in the context of Papers I and II; to summarise, the inclusion–exclusion algorithm of Björklund et al. [16] can be used to evaluate the disjoint product over rings in time $\binom{n}{k/2} n^{O(1)}$, while the results of Paper I give a $2^{k/2}\binom{n}{k/2} n^{O(1)}$ time algorithm over semirings.

Paper III. Paper III considers the problem of finding *maximum bounded tree-width subgraphs*. The work is motivated by the study of graphical probabilistic models, specifically *Bayesian networks*, which can be viewed as directed acyclic graphs. The framework introduced in Paper III can be used to find subgraphs of tree-width at most w and

1. maximum weight in time $3^n n^{w+O(1)}$, and
2. maximum number of edges in time $2^n n^{w+O(1)}$.

Furthermore, a variant of this algorithm can be applied to the *Bayesian network structure learning* problem. The algorithm exploits the existence of nice tree decompositions for bounded tree-width graphs, using the combinatorial structure of the tree decompositions as a basis for Bellman–Held–Karp-style dynamic programming with additional branching step.

▶ *See Section 3.2 for details.*

Paper IV. Paper IV studies finding *small unbalanced vertex cuts* in a graph. That is, the task is to find a partition (X, S, Y) of the vertex set such that X is a non-empty set of size at most k, the set S has size at most t and S separates X and Y.

We show that this problem is fixed-parameter tractable with parameter t by giving an algorithm with running time $4^t n^{O(1)}$. The algorithm is based on a combinatorial decomposition of the solution in terms of *important separators*. Roughly speaking, to find a solution (X, S, Y), we guess a vertex $x \in X$ and use important separators to find sets $Y_1, Y_2, \ldots, Y_\ell$ that are in a certain sense furthest away from x. Some subfamily of sets Y_i is then contained in the larger side Y of the cut, and an important separator between this subfamily and x gives us the solution.

▶ *See Section 3.3 for details.*

1.2.3 Algebraic catalysts

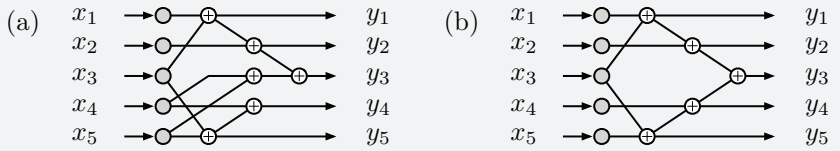
Overview. Working over semirings prevents us from using many algebraic techniques, which usually rely on some *algebraic catalyst*, most often the existence of additive inverses in the ground ring. Indeed, evaluating multilinear forms over semirings may be viewed as evaluating the forms in a monotone model of computation; it is known that monotone models are generally weaker than non-monotone ones. We will see this with the multilinear form evaluation algorithms of Papers I–III, as our combinatorial decomposition

Example 1.4: Idempotent versus non-idempotent addition

Let us consider the matrix

$$A = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \end{bmatrix}.$$

The illustration below shows the optimal circuit for evaluating the linear forms defined by A (a) over commutative groups and (b) over commutative idempotent semigroups, with 6 and 5 addition gates, respectively. Note in particular that negation does not help over groups in this case.



algorithms do not match the algebraic algorithms for analogous multilinear forms over rings.

However, in some cases the ground semiring may have properties that can be used as catalyst for efficient algorithms even if it lacks additive inverses. We are particularly interested in the case of Boolean matrices and linear forms when the additive semigroup $(S, \oplus)$ is *idempotent*, that is, $x \oplus x = x$ for all $x \in S$; an example is the maximum operation, as we have $\max(x, x) = x$. Specifically, then the main question is whether some linear forms are easier to evaluate over idempotent semigroups than groups.

- *See Chapter 4 for more background.*
- *See also Example 1.4.*

Paper V. The main result of Paper V considers *intersection matrices of arbitrary set families*; given two set families $\mathcal{F}, \mathcal{G} \subseteq 2^X$, their intersection matrix N is a matrix whose rows are indexed by sets $Y \in \mathcal{G}$ and the columns by sets $X \in \mathcal{F}$, with entries

$$N(Y, X) = \begin{cases} 1, & \text{if } Y \cap X \neq \emptyset, \text{ and} \\ 0, & \text{if } Y \cap X = \emptyset. \end{cases}$$

The set of linear forms defined by any such intersection matrix, called the *intersection transform*, has an almost linear algorithm over commutative idempotent semigroups. However, any strictly sub-quadratic algorithm for commutative groups would give us a $2^{(1-\varepsilon)n}(n+m)^{O(1)}$ time algorithm for CNF-SAT for a constant $\varepsilon > 0$, violating the *strong exponential-time hypothesis*. This can be interpreted as conditional evidence suggesting that intersection transforms are difficult to evaluate over commutative groups. On the other hand, it should be noted that the strong exponential time hypothesis is a very strong complexity assumption, and as such may very well be false; from this perspective, the result can also be seen as a possible step towards faster CNF-SAT algorithms.

► *See Chapter 4 for details.*

1.3 Author contributions

Papers I & II. The algorithm of Paper II for disjoint summation is due to the present author, while the applications are joint work with Petteri Kaski and Mikko Koivisto. Igor Sergeev [119] subsequently gave a faster algorithm for disjoint summation; this algorithm was further improved in Paper I jointly by the present author and Igor Sergeev.

Paper III. The results are joint work by the present author and Pekka Parviainen. The algorithm implementation is the work of the present author.

Paper IV. The results are joint work by Fedor Fomin, Petr Golovach and the present author.

Paper V. The results concerning lower bounds for intersection transforms are joint work by Petteri Kaski, Mikko Koivisto and the present author. The SAT encoding was developed together with Matti Järvisalo and the experiments were implemented and run by the present author.

Chapter 2

Preliminaries

This chapter briefly introduces notions and notations used throughout the rest of this thesis. We will assume in later chapters that the reader is familiar with the notations introduced in this chapter. Furthermore, the reader is assumed to be familiar with basic notions of graph theory [51], algorithmics and complexity theory [5] and parameterised complexity theory [52, 57].

2.1 Notations and definitions

Sets and set families. Let $0 \leq k \leq n$ be integers, and let X be a set. We write

1. $[n] = \{1, 2, \ldots, n\}$ for the set of integers at most n,
2. $2^X = \{Y \subseteq X\}$ for the power set of X,
3. $\binom{X}{k} = \{Y \subseteq X \colon |Y| = k\}$ for the family of subsets of X of size k, and
4. $\binom{X}{\downarrow k} = \bigcup_{i=0}^{k} \binom{X}{i}$ for the family of subsets of X of size at most k.

Graphs and hypergraphs. A *hypergraph* (*set family*, *set system*) is a pair $H = (V, \mathcal{E})$, where V is the set of *vertices* and $\mathcal{E} \subseteq 2^V$ is the set of *edges*. When the vertex set is clear from the context, we identify a hypergraph with its edge set.

A *graph* is a hypergraph where each edge contains exactly two elements. We follow the usual convention of using regular upper-case letters as the names of graph edge sets and lower-case letters as the names of single edges.

Binary operations. Let S be a set and let $\circ\colon S \times S \to S$ be a binary operation on S. An element $e \in S$ is an *identity element for* $\circ$ if $e \circ x = x \circ e = x$ for all $x \in S$. We say that $\circ$ is

1. *associative* if $(x \circ y) \circ z = x \circ (y \circ z)$ for all $x, y, z \in S$,
2. *commutative* if $x \circ y = y \circ x$ for all $x, y \in S$,
3. *idempotent* if $x \circ x = x$ for all $x \in S$, and
4. *invertible* if there is a unique identity element e for $\circ$ and for each $x \in S$, there is $y \in S$ such that $x \circ y = y \circ x = e$.

Semigroups and groups. Let S be a set and $\oplus\colon S \times S \to S$ a binary operation on S. We say that the pair $(S, \oplus)$ is

1. a *semigroup* if $\oplus$ is associative,
2. a *monoid* if $\oplus$ has an identity element and is associative, and
3. a *group* if $\oplus$ has an identity element and is associative and invertible.

Furthermore, we call the structure $(S, \oplus)$ *commutative* if $\oplus$ is commutative.

Semirings and rings. More generally, let S be a set and $\oplus, \odot\colon S \times S \to S$ be two binary operations on S. We say that $(S, \oplus, \odot)$ is a *semiring* if

(a) $(S, \oplus)$ is a commutative monoid with identity element 0,

(b) $(S, \odot)$ is a monoid with identity element 1,

(c) multiplication $\odot$ distributes over addition $\oplus$, that is, we have

$$x \odot (y \oplus z) = (x \odot y) \oplus (x \odot z)\,,$$
$$(y \oplus z) \odot x = (y \odot x) \oplus (z \odot x)$$

for all $x, y, z \in S$, and

(d) multiplication by 0 annihilates, that is, $0 \odot x = x \odot 0 = 0$ for all $x \in S$.

We say that semiring $(S, \oplus, \odot)$ is *commutative* if $\odot$ is commutative, and $(S, \oplus, \odot)$ is a *ring* if $\oplus$ is also invertible.

Iverson bracket notation. Let P be a logical predicate and let $(S, \oplus, \odot)$ be semiring with additive identity element 0 and multiplicative identity element 1. The *Iverson bracket* is the expression

$$[P] = \begin{cases} 1, & \text{if } P \text{ is true, and} \\ 0, & \text{otherwise.} \end{cases}$$

2.2 Matrices and multilinear forms

Vectors and matrices. For our purposes it is usually convenient to consider vectors and matrices that are indexed by objects other than natural numbers. Thus, we will generally view vectors as functions, that is, given a finite index set X, a vector of values in semiring S is a function $f\colon X \to S$. Similarly, we will treat matrices as functions

$$A\colon Y \times X \to S\,,$$

and more generally *multidimensional matrices* or *tensors* as functions

$$T\colon Y \times X_1 \times X_2 \times \cdots \times X_n \to S\,.$$

Linear and multilinear forms. Matrices define sets of *multilinear forms* and, as discussed in Chapter 1, evaluation of such forms is one of the main themes of this thesis. Specifically, a matrix $A\colon Y \times X \to S$ defines a set of linear forms

$$\begin{aligned} A\colon (X \to S) &\to (Y \to S)\,,\\ (A(f))(y) &= \bigoplus_{x\in X} A(y,x) \odot f(x)\,, \end{aligned} \tag{2.1}$$

that is, A defines a function mapping a vector $f\colon X \to S$ to a vector $A(f)\colon Y \to S$. More concretely, these linear forms may be viewed as a computational task where we are given an input vector $f\colon X \to S$, and our task is to compute an output vector $e\colon Y \to S$ defined as

$$e(y) = \bigoplus_{x\in X} A(y,x) \odot f(x)\,.$$

We will often use post-fix notation for linear forms, that is, we write fA for $A(f)$.

Similarly, a multidimensional matrix $T\colon Y \times X_1 \times X_2 \times \cdots \times X_n \to S$ defines a set of multilinear forms

$$\begin{aligned} T\colon \prod_{i=1}^{n} (X_i \to S) &\to (Y \to S)\,,\\ (T(f_1,\ldots,f_n))(y) &= \bigoplus_{x_1\in X_1} \cdots \bigoplus_{x_n\in X_n} T(y,x_1,\ldots,x_n) \bigodot_{i=1}^{n} f_i(x_i)\,. \end{aligned}$$

Again, this can be viewed as a computational task with input vectors $f_i\colon X_i \to S$ and output $e\colon Y \to S$ specified by $e = T(f_1,\ldots,f_n)$.

Boolean matrices. For a Boolean matrix $A\colon Y \times X \to \{0,1\}$, we can consider the set of linear forms defined by A over a commutative semigroup $(S, \oplus)$, even when S does not have full semiring structure. First, we may assume that S has an identity element, as we can append one if it does not already exist. The semigroup $(\{0,1\}, \wedge)$ now has a natural left action $\odot$ on S, defined by

$$0 \odot x = 0\,, \qquad\qquad 1 \odot x = x\,.$$

With this extra structure the definition (2.1) is valid also over $(S, \oplus)$.

Models of computation. We will, in general, consider the evaluation of multilinear forms in an algebraic complexity setting where the semiring is not fixed beforehand, and furthermore, we are interested in the complexity of the task in terms of semiring operations [28]. Specifically, we will consider one of the following models, depending on the situation.

1. *Algebraic circuits*, or equivalently, *algebraic straight-line programs*: An algebraic circuit is a directed acyclic graph $D = (V, A)$, where each vertex is either
   (a) an *input gate* with in-degree zero, labelled with an input entry, or
   (b) an *operation gate* with non-zero in-degree, labelled with a semiring operation.

   Furthermore, some of the operation gates are labelled as *output gates* for specific output entries. The complexity measure is the number of edges in the graph D.
2. *Uniform algebraic circuits*: Uniform algebraic circuits are circuits that can be constructed on a uniform machine; the complexity measure is the running time of the construction algorithm.
3. *Uniform algebraic model*: In this model, we consider algorithms that run on a random-access machine with black-box access to semiring operations, that is, the machine can store semiring elements in its memory, each taking constant space, and perform semiring operations on them in constant time. The complexity measures are the space and time used by the algorithm.

In practice, there is little difference between the two uniform models, and all positive results of this thesis can be interpreted in both of them.

Chapter 3

Algorithm design

This chapter gives an overview of the positive results of this thesis, based on Papers I–IV.

3.1 Disjoint summation and meet-in-the-middle

This section is based on Papers I and II.

3.1.1 Introduction

Meet-in-the-middle. *Meet-in-the-middle* or *split and list* is an algorithmic technique for solving hard problems by essentially splitting the task into two or more parts. That is, the basic approach is to (a) split the problem into two or more equally-sized parts, (b) enumerate all solutions for the smaller parts, and (c) combine the solutions from the different parts via some fast algorithm.

The classical example of the meet-in-the-middle framework is the exact $2^{n/2}n^{O(1)}$ time algorithm for the subset sum problem, due to Horowitz and Sahni [74]. Given a list of n numbers, we

1. split the list into two lists of length $n/2$,
2. for both of these new lists, enumerate all $2^{n/2}$ different sums that can be obtained using elements of that list, and
3. use sorting and binary search to test whether there are two sub-results from different parts that sum to 0.

Further applications where this technique gives a halved exponential dependence on the input size include binary knapsack [74, 118], exact satisfiability and set cover [118], and (σ, ρ)-dominating sets [62]. Williams [133] uses

splitting into three parts and combination via matrix multiplication to reduce the exponential dependence on the input size from n to $\omega n/3$, where $\omega < 2.3727$ is the exponent of matrix multiplication [129].

Finding small subgraphs. In this section, we consider meet-in-the-middle in the context of algorithms for *counting small subgraphs*; that is, we have a pattern graph H with k vertices, and we want to count subgraphs of the input graph G that are isomorphic to H. In particular, we will look at the case where k is chosen to be a k-path. Note that the trivial algorithm for any pattern H simply tries all possible injective homomorphisms from H to G, giving running time $n^{k+O(1)}$.

As a starting point, one can consider the problem of simply finding a subgraph isomorphic to the pattern graph H; the complexity of this problem depends on the choice of H. For k-paths, the problem is fixed-parameter tractable [107]; a single-exponential dependence on k was obtained by Alon et al. [1], using a technique of colour-coding. Indeed, they establish that finding a copy of pattern graph H is fixed-parameter tractable in k whenever H has bounded tree-width. Subsequent work has further improved the running time for k-paths, both in deterministic [36, 89] and randomised [18, 95, 134] settings. On the other hand, finding a k-clique is W[1]-hard [52], but can be solved in time $n^{\omega k/3+O(1)}$ [108].

However, counting subgraphs isomorphic to H is more difficult than finding in general; indeed, the counting problem is #W[1]-hard even when H is a k-path [58] or a k-matching [47]. The k-clique algorithm of Nešetřil and Poljak [108] extends to counting copies of any fixed k-vertex pattern graph H in time $n^{\omega k/3+O(1)}$, and a standard polynomial encoding technique allows counting with positive integer edge weights at most W in time $n^{\omega k/3+O(1)}W$. An algorithm by Vassilevska Williams and Williams [130] further enables counting with arbitrary real node weights in time $n^{\omega k/3+O(1)}$.

For patterns H with suitable properties, running time $n^{\omega k/3+O(1)}$ can be improved upon; again, k-paths are a particular example. Indeed, three different algorithms for counting k-paths in time roughly proportional to $n^{k/2}$ were proposed almost simultaneously.

1. For patterns with an independent set of size s, Vassilevska Williams and Williams [130] give a counting algorithm with running time $2^s n^{k-s} n^{O(1)}$; when applied for k-paths, this gives an algorithm with running time $2^{k/2} n^{k/2+O(1)}$.
2. The group algebra techniques of Koutis and Williams [96] give an algorithm with running time $n^{k/2+O(1)}$; they enumerate all $k/2$-paths

and combine these via fast polynomial multiplication and multilinear monomial counting.

3. The similar meet-in-the-middle algorithm of Björklund et al. [16] gives running time $\binom{n}{k/2}n^{O(1)}$. Their algorithm counts $k/2$-paths for each subset of size $k/2$ and combines the results via a disjoint product of set functions; this is the approach illustrated in Example 1.3.

Fomin et al. [63] extend the last result to counting patterns of path-width p in time $\binom{n}{k/2}n^{2p+O(1)}$. Furthermore, the recent work of Björklund et al. [12] improves upon this by splitting the pattern graph into three parts of size $k/3$ and combining the parts via a disjoint product of three set functions. Specifically, their result gives running time $n^{0.4547k+2p+O(1)}$ for counting patterns of path-width p, assuming p is sufficiently small compared to k.

Papers I and II. Papers I and II consider another way to extend the meet-in-the-middle technique of Björklund et al. [16]; whereas the earlier results are based on the evaluation of disjoint products over rings, we consider the evaluation of disjoint products over semirings without additive inverses. There are two principal motivations for this work; first is the question whether the existence of additive inverses is necessary for the fast evaluation of disjoint products, and the second is the prospect of additional applications made possible by semiring encodings. Indeed, one such application is given in Example 1.3.

- *Section 3.1.2 introduces (n, p, q)-disjoint summation over commutative semigroups and discusses the core algorithmic results of Papers I and II.*
- *Section 3.1.3 discusses the meet-in-the-middle framework of Björklund et al. [16] and extensions over semirings.*

3.1.2 Ingredient: disjoint summation over semigroups

Disjoint summation. The *disjointness matrix* $D(\mathcal{E}, \mathcal{F})$ of two set families $\mathcal{E}, \mathcal{F} \subseteq 2^{[n]}$ is the matrix

$$D\colon \mathcal{E} \times \mathcal{F} \to S\,, \qquad\qquad D(Y, X) = [Y \cap X = \emptyset]\,.$$

This matrix defines the $(\mathcal{F}, \mathcal{E})$-*disjoint transform* over any commutative semigroup $(S, \oplus)$, defined as

$$\delta\colon (\mathcal{F} \to S) \to (\mathcal{E} \to S)\,, \qquad\qquad (f\delta)(Y) = \bigoplus_{X \cap Y = \emptyset} f(X)\,.$$

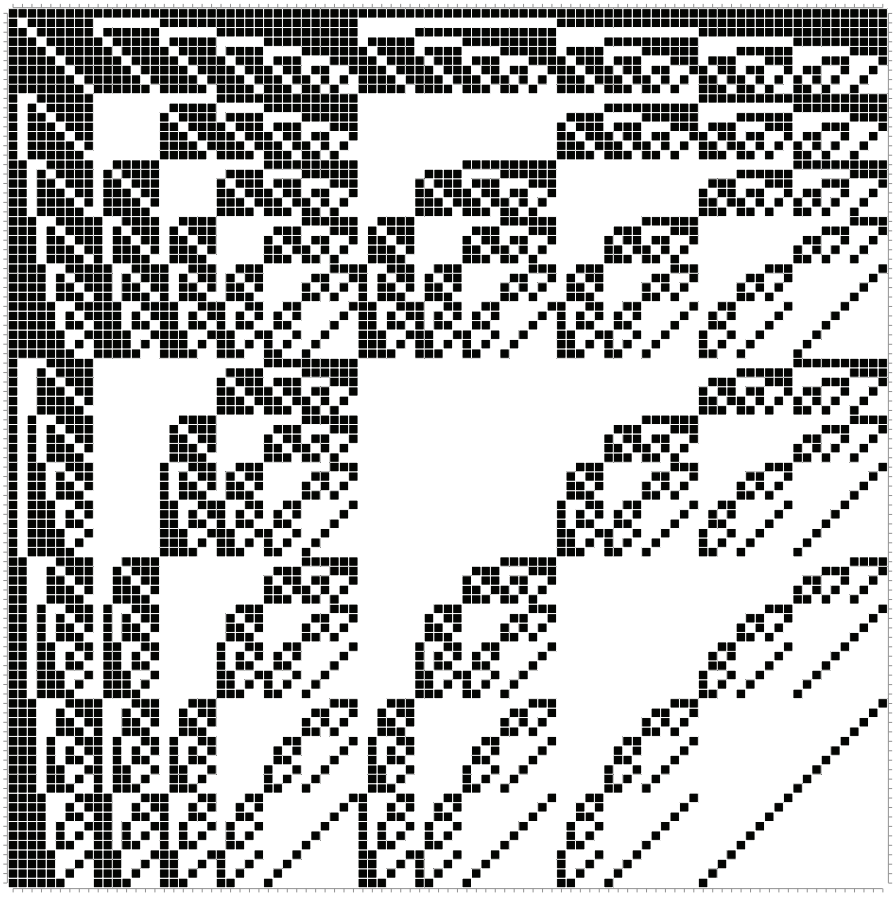


Figure 3.1: The disjointness matrix $D\left(\binom{[8]}{\downarrow 3}, \binom{[8]}{\downarrow 3}\right)$. The index sets are ordered by size, and lexicographically within the size.

Papers I and II study the evaluation of disjoint transform in the case where $\mathcal{F}$ and $\mathcal{E}$ are the complete hypergraphs $\binom{[n]}{\downarrow p}$ and $\binom{[n]}{\downarrow q}$, respectively; see Figure 3.1 We refer to this special case as (n,p,q)-*disjoint summation*. That is, given an input function $f\colon \binom{[n]}{\downarrow p} \to S$, the task is to compute the disjoint transform

$$\delta\colon \left(\binom{[n]}{\downarrow p} \to S\right) \to \left(\binom{[n]}{\downarrow q} \to S\right), \qquad (f\delta)(Y) = \bigoplus_{X \cap Y = \emptyset} f(X)\,.$$

Paper I shows that (n,p,q)-disjoint summation can be evaluated over any commutative semigroup in time that is almost linear in the combined input and output sizes. That is, the main theorem in the paper is as follows; see also Figure 3.2.

Theorem 3.1. *The (n,p,q)-disjoint summation can be evaluated over commutative semigroups in the uniform algebraic model in time*

$$\left[\binom{n}{p} + \binom{n}{q}\right] \cdot \min\{2^p, 2^q\} \cdot n^{O(1)}\,.$$

Controlling intersections. The earlier Paper II also gives an algorithm for (n,p,q)-disjoint summation, albeit one with slightly worse running time bounds than given in Theorem 3.1. However, the construction of Paper II allows a more finely tuned control over the intersections in summation terms.

Specifically, Paper II studies (n,p,q)-*intersection summation*, where we are given a value $g(Z,X)$ for each pair (Z,X) with $X \in \binom{[n]}{\downarrow p}$ and $Z \subseteq X$, and the task is to compute for each $Y \in \binom{[n]}{\downarrow q}$ the sum

$$\bigoplus_{X \in \binom{[n]}{\downarrow p}} f(X \cap Y, X)\,.$$

The main result of Paper II shows that this extension can be solved almost as efficiently as (n,p,q)-disjoint summation, as stated in the following theorem. See also Figure 3.3.

Theorem 3.2. *The (n,p,q)-intersection summation can be evaluated over commutative semigroups in the uniform algebraic model in time*

$$(n^p + n^q)n^{O(1)}\,.$$

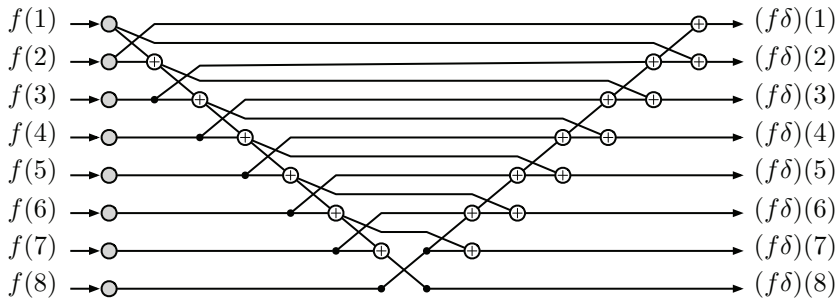


Figure 3.2: A $(8, 1, 1)$-disjoint summation circuit, based on the construction of Paper I. The inputs and outputs corresponding to the empty set are omitted.

Related work. Some special cases of (n, p, q)-disjoint summation over commutative semigroups are covered by earlier work; we summarise those results here.

1. If we omit the empty set, the $(n, 1, 1)$-disjoint summation corresponds to the matrix-vector multiplication task $x \mapsto \bar{I}x$, where $\bar{I}$ is the Boolean complement of the identity matrix; see Example 1.2. The optimal circuit for this task uses $3n - 6$ semigroup additions [34, 127]. Indeed, this construction is illustrated in Example 1.2(b).
2. For (n, n, n)-disjoint summation, Yates's algorithm [135] uses $2^{n-1}n$ semigroup additions, which is known to be optimal [87]. See Papers I and II for details.

If we restrict attention to the case where the ground semigroup S has full group structure, then it becomes possible to employ various algebraic techniques. Let us consider the *ℓ-intersection matrix* $N_\ell(\mathcal{E}, \mathcal{F})$ of two set families $\mathcal{E}, \mathcal{F} \subseteq 2^{[n]}$ is the matrix

$$N_\ell \colon \mathcal{E} \times \mathcal{F} \to S\,, \qquad N_\ell(Y, X) = [|Y \cap X| = \ell]\,.$$

This matrix defines the *ℓ-intersection summation* for set families $\mathcal{F}$ and $\mathcal{E}$; given an input function $f \colon \mathcal{F} \to S$, the task is to compute the *ℓ-intersection transform*

$$\iota_\ell \colon (\mathcal{F} \to S) \to (\mathcal{E} \to S)\,, \qquad (f\iota_\ell)(Y) = \bigoplus_{|X \cap Y| = \ell} f(X)\,.$$

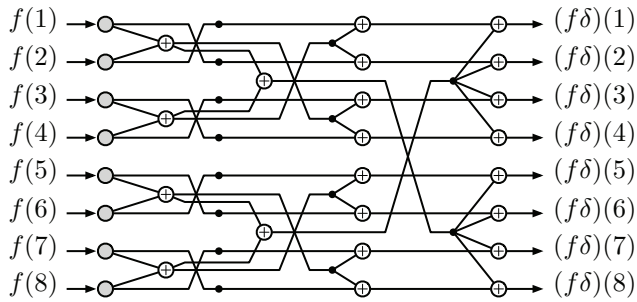


Figure 3.3: A $(8, 1, 1)$-disjoint summation circuit, based on the construction of Paper II. The inputs and outputs corresponding to the empty set are omitted.

Björklund et al. [14] show that for any set families $\mathcal{E}$ and $\mathcal{F}$, the ℓ-intersection summation for all $\ell = 0, 1, \ldots, n$ can be computed over any commutative group in the uniform algebraic model in time

$$(|{\downarrow}\mathcal{F}| + |{\downarrow}\mathcal{E}|)n^{O(1)}\,,$$

where $\downarrow\mathcal{F}$ denotes the *down-closure* of $\mathcal{F}$, that is, the set family

$$\left\{Z \in 2^{[n]} \colon Z \subseteq X \text{ for some } X \in \mathcal{F}\right\}.$$

This in particular implies that (n, p, q)-disjoint summation can also be evaluated in the uniform algebraic model in time $[\binom{n}{p} + \binom{n}{q}]n^{O(1)}$ over commutative groups.

3.1.3 Ingredient: meet-in-the-middle via disjoint products

Disjoint product. For applications, it is more often convenient to consider $(\mathcal{E}, \mathcal{F})$-*disjoint products* of two functions over a semiring $(S, \oplus, \odot)$. That is, given set families $\mathcal{F}$ and $\mathcal{G}$, and functions $f \colon \mathcal{F} \to S$ and $g \colon \mathcal{G} \to S$, the disjoint product $\Delta(g, f)$ is defined as

$$\Delta(g, f) = \bigoplus_{X,Y \colon X \cap Y = \emptyset} g(Y) \odot f(X)\,.$$

The disjoint product is closely connected to $(\mathcal{E}, \mathcal{F})$-disjoint transform; indeed, we have

$$\Delta(g, f) = \bigoplus_{Y} g(Y) \odot (f\delta)(Y)\,.$$

Thus, to evaluate $(\mathcal{E},\mathcal{F})$-disjoint product, it suffices to evaluate $(\mathcal{E},\mathcal{F})$-disjoint transform and $|\mathcal{E}|$ multiplications in S. Thus, the results of Björklund et al. [14] immediately give the following; see also the inclusion–exclusion techniques used by Björklund et al. [16].

Theorem 3.3 ([14, 16])**.** *The $(\mathcal{E},\mathcal{F})$-disjoint summation can be evaluated over rings in the uniform algebraic model in time*

$$(|{\downarrow}\mathcal{F}| + |{\downarrow}\mathcal{E}|)n^{O(1)}\,.$$

Again, we are interested in the case where $\mathcal{F} = \binom{[n]}{\downarrow p}$ and $\mathcal{E} = \binom{[n]}{\downarrow q}$; we refer to this case as (n,p,q)-*disjoint product*. Theorems 3.1 and 3.3 now give us the following corollary.

Corollary 3.4. *The (n,p,q)-disjoint product can be evaluated in the uniform algebraic model*

(i) *over rings in time* $\left[\binom{n}{p} + \binom{n}{q}\right] \cdot n^{O(1)}$*, and*

(ii) *over semirings in time* $\left[\binom{n}{p} + \binom{n}{q}\right] \cdot \min\{2^p, 2^q\} \cdot n^{O(1)}$.

Applications. We have already covered the basic idea of meet-in-the-middle via disjoint products for counting problems in Example 1.3; Corollary 3.4(i) provides the fast disjoint product algorithm required. In general, the technique works for counting any objects of size $k/2$, as long as the object in question can be split into two independent and disjoint parts; for example, in the k-path application, the only dependence between parts occurs at the middle vertex. To summarise, repeating some of the results already mentioned, application of Corollary 3.4(i) gives an algorithm with running time

1. $\binom{n}{k/2}n^{O(1)}$ for counting k-paths [16],
2. $\binom{n}{k/2}n^{2p+O(1)}$ for counting the number of subgraphs isomorphic to k-vertex graph input graph H of path-width p [63], and
3. $\binom{n}{pq/2}n^{O(1)}$ for counting p-packings of q-subsets [16].

These results can be extended to counting maximum-weight k-paths and p-packings of q-subsets. If the weights are integers with absolute value bounded by M, then we can encode the weights into the polynomial ring $\mathbb{Z}[X]$ and apply Theorem 3.4(i); see e.g. Björklund et al. [13]. This gives running time

1. $\binom{n}{k/2}n^{O(1)}M\log M$ for counting maximum-weight k-paths, and

2. $\binom{n}{pq/2} n^{O(1)} M \log M$ for counting maximum-weight p-packings of q-subsets.

For unbounded weights, the semiring encoding of Example 1.3 can be used along with Corollary 3.4(ii). In particular, the latter gives the following theorem of Paper I.

Theorem 3.5. *Maximum-weight objects can be counted in time*

(i) $\binom{n}{k/2} 2^{k/2} n^{O(1)}$ *for k-paths, and*

(ii) $\binom{n}{pq/2} 2^{pq/2} n^{O(1)}$ *for p-packings of q-subsets.*

Three-way disjoint product. Björklund et al. [12] extend the meet-in-the-middle via disjoint products by splitting the objects into three independent equally-sized parts; for example, a k-path can be split into three parts of length $k/3$, and the only interaction happens at the cut points. The parts are then assembled via the trilinear disjoint product of three functions $f, g, h\colon \binom{[n]}{\downarrow(k/3)} \to \mathbb{Z}$, given by

$$\Delta(f,g,h) = \sum_{\substack{X,Y,Z \\ X\cap Y = Y\cap X = X\cap Z = \emptyset}} f(X)g(Y)h(Z)\,.$$

Björklund et al. [12] show that the product $\Delta(f,g,h)$ can be evaluated over integers in time $n^{0.4547k+O(1)}$ for any constant k. This gives improved algorithms for the counting applications mentioned above, assuming k is constant.

3.2 Finding bounded tree-width graphs

This section is based on Paper III.

3.2.1 Introduction

Probabilistic graphical models. A *probabilistic graphical model* [93] is a presentation of a joint probability distribution, given by an undirected or directed graph with the individual variables as vertices; the *graph structure*, i.e. the edges of the graph represent the dependencies between the variables, and the *parameters* give the conditional probability distributions of the individual variables given their neighbours. The motivation for such models is that they allow presentation and processing of complex distributions in a succinct and efficient manner, as we only need to store and consider

variable dependencies given by the graph structure. Two principal examples of graphical models are *Markov networks*, based on undirected graphs, and *Bayesian networks* [113], based on acyclic directed graphs.

Bayesian networks. The work presented in this section is in particular motivated by applications to Bayesian networks. A Bayesian network represents a probability distribution as follows. The *network structure* is a directed acyclic graph $D = (V, A)$; for each $v \in V$, we have a random variable $\boldsymbol{X}_v$. The *parameters* of the Bayesian network are the conditional probabilities

$$\Pr\Big(\boldsymbol{X}_v = x_v \;\Big|\; \bigcap_{u \in A_v} \boldsymbol{X}_u = x_u\Big) \tag{3.1}$$

for each possible choice of values x_v and x_u for $u \in A_v$ in the ranges of the respective random variables, where A_v is the *parent set* of v in D, that is, the set of vertices u such that $(u, v) \in A$. The joint probability distribution is then given by

$$\Pr\Big(\bigcap_{v \in V} \boldsymbol{X}_v = x_v\Big) = \prod_{v \in V} \Pr\Big(\boldsymbol{X}_v = x_v \;\Big|\; \bigcap_{u \in A_v} \boldsymbol{X}_u = x_u\Big)\,.$$

From a computational perspective, there are two main tasks associated with the use of Bayesian network.

1. Given input data, usually based on the real-world phenomenon to be modelled, the *learning problem* concerns finding a Bayesian network, i.e. both the network structure and parameters that match the data well.
2. Once we have learned a Bayesian network, we use it to compute the conditional probabilities of a set of variables given some other set of variables; this is called the *inference problem*.

Both of these tasks are NP-hard in general [39, 40, 43]. However, if the network structure is restricted to a suitable graph class, i.e. the allowed dependencies between variables are further restricted, the inference problem can become tractable. For example, if the graph structure is required to be a tree, then both learning and inference can be done in polynomial time [41, 54]. More generally, inference in Bayesian networks is fixed-parameter tractable in the tree-width of the graph structure; see e.g. Dechter [50].

Learning network structures. In this work, we focus on learning the network structure. We view Bayesian network structure learning as a combinatorial problem by using an abstract score-based approach [110, 112].

In this framework, we are given a set of vertices V and scoring functions f_v for each $v \in V$, and the task to compute

$$\max_{D=(V,A)} \sum_{v \in V} f_v(A_v)\,,$$

where D ranges over all directed acyclic graphs on V. The idea is that the scoring functions f_v are pre-computed from the input data; see e.g. Cooper and Herskovits [44] and Heckerman et al. [72]. The fastest known exact algorithm for this problem is a Bellman–Held–Karp-style dynamic programming algorithm with running time $2^n n^{O(1)}$ [92, 111, 120, 121].

Learning with bounded tree-width. As noted before, inference in Bayesian networks is fixed-parameter tractable in the tree-width of the network structure. This observation motivates us to consider *learning bounded tree-width network structures*, as restricting the tree-width of the learned network structure also limits the time required for subsequent inference tasks. More specifically, this gives us a trade-off between the fit of the network and inference speed; decreasing the tree-width bound limits the space of possible models but guarantees faster inference on the resulting Bayesian network.

Formally, the problem is as follows. We are given the vertex set V and scoring functions f_v for each $v \in V$ as before, and additionally an integer tree-width bound $w \geq 1$, and the task is to compute

$$\max_{D=(V,A)} \sum_{v \in V} f_v(A_v)\,,$$

where D ranges over directed acyclic graphs of tree-width at most w. Here the tree-width of a directed acyclic graph is taken to be the tree-width of its *moralised graph*; we defer the technical discussion of this point to Section 3.2.2.

Unlike in the case of the inference problem, addition of the tree-width constraint does not make the structure learning problem fixed-parameter tractable. Indeed, the problem is NP-hard even for any fixed $w \geq 2$; see Paper III. Prior algorithmic work on learning bounded tree-width Bayesian network structures is limited to the heuristic search algorithm of Elidan and Gould [55]; for undirected graphical models, approximate and heuristic algorithms have been proposed [6, 35, 81, 123].

Paper III. Paper III presents an exact exponential algorithm for finding bounded tree-width Bayesian network structures. On a high level, the basic approach is to extend the Bellman–Held–Karp-style dynamic programming

algorithm for permutation problems into a dynamic programming over all tree decompositions by adding an extra branching step. More generally, the dynamic programming machinery of Paper III can also be used to solve optimisation problems that concern finding maximum-weight bounded tree-width graphs.

- *Section 3.2.2 presents definitions and basic theory of tree-width and tree decompositions.*
- *Section 3.2.3 gives an overview of a generic version of the dynamic programming machinery of Paper III.*

3.2.2 Ingredient: tree decompositions

Introduction. The basic idea is that the tree-width of a graph measures how tree-like the graph is; an equivalent concept was first introduced by Halin [71]. The concept was rediscovered independently by Robertson and Seymour [117] and Arnborg and Proskurowski [3], who both observed that certain NP-hard problems can be solved in polynomial time on graphs of fixed tree-width; the apparent algorithmic usefulness quickly prompted an extensive study of the concept [4, 21, 46].

Generally, if an NP-hard graph problem is polynomial-time solvable on trees, it is often also fixed-parameter tractable in the tree-width of the input graph. There are two principal ways to obtain fixed-parameter algorithms for problems parameterised by the tree-width w of the input graph.

1. Courcelle's Theorem [46] is a meta-theorem stating that recognising any graph property definable in monadic second-order logic of graphs is fixed-parameter tractable in w. This is sufficient to obtain fixed-parameter algorithms for most problems; however, algorithms given by Courcelle's Theorem have highly super-exponential dependence on w.
2. For faster algorithms, the underlying techniques of Courcelle's theorem can be fine-tuned for specific problems. This generally means using dynamic programming over *tree decompositions* of low width [21, 24]; such a decompositions can be constructed in fixed-parameter time for graphs of low tree-width, as we will discuss below. Essentially, access to a tree decomposition allows one to significantly limit possible interactions between partial solutions of problems, in a similar manner as one would do when solving the problem on trees. Furthermore, algebraic techniques are often applicable to obtain further speed-ups [25, 128].

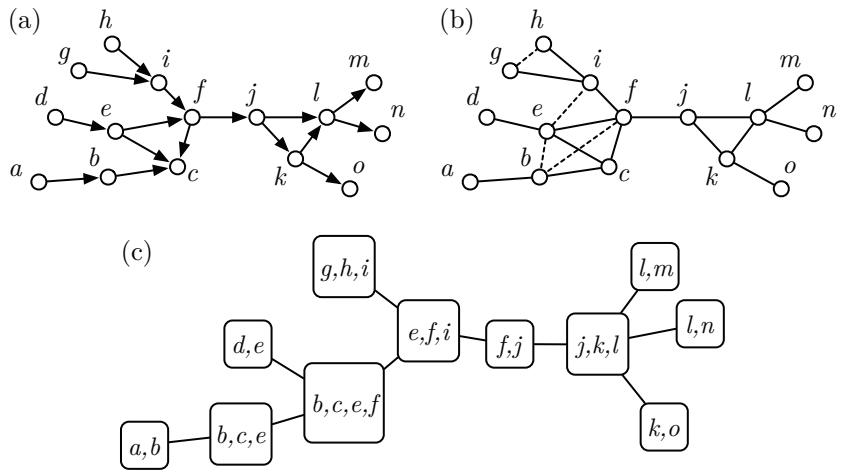


Figure 3.4: (a) A directed acyclic graph, (b) its moralised graph, with edges arising from v-structures highlighted, and (c) its tree decomposition.

Definitions and basic results. In this section, we review the basic definitions and theory of tree-width and tree decompositions. The definitions are essentially the ones introduced by Robertson and Seymour [117].

Definition 3.6. A *tree decomposition* of a graph $G = (V, E)$ is a pair $(\mathcal{X}, T)$, where $T = (I, F)$ is a tree and $\mathcal{X} = \{X_i : i \in I\}$ is an indexed family of subsets of V such that

(a) $\bigcup_{i \in I} X_i = V$,

(b) for each $e \in E$ there is $i \in I$ such that $e \subseteq X_i$, and

(c) for each $v \in V$ the subtree $T[\{i \in I : v \in X_i\}]$ is connected.

A *rooted tree decomposition* is a tree decomposition $(\mathcal{X}, T, r)$, where $r \in I$ is the root of the tree T. The *width* of a tree decomposition $(T, \mathcal{X})$ is $\min_{i \in I} |X_i| - 1$.

Definition 3.7. The *tree-width* of a graph $G = (V, E)$ is the minimum width of a tree decomposition of H.

Alternatively one may consider *path decompositions* and *path-width*. The definitions are analogous, the only difference is that we require the tree T of the decomposition to be a path.

It follows from the definition of tree-width that if G has a clique $K \subseteq V$, then K is contained in a set X_v for some $v \in V$. Another basic property of

tree decompositions is that the vertex sets X_i act as separators in G, in the sense of the following lemma.

Lemma 3.8 (Separation Property)**.** *Let $H = (V, \mathcal{E})$ be a graph with tree decomposition $(\mathcal{X}, T)$, and let i, j and k be nodes in T such that j is on the path from i to k. Then there is no edge $S \in \mathcal{E}$ such that $S \cap (X_i \setminus X_j) \neq \emptyset$ and $S \cap (X_k \setminus X_j) \neq \emptyset$.*

For algorithmic applications, it is convenient to consider tree decompositions of a specific type.

Definition 3.9. A *nice tree decomposition* is a rooted tree decomposition $(\mathcal{X}, T, r)$ such that each $i \in I$ is either

1. a *leaf* with no children and $|X_i| = 1$,
2. a *forget node* for some $v \in V$, with one child j such that $v \in X_j$ and $X_i = X_j \setminus \{v\}$,
3. a *introduce node* for some $v \in V$, with one child j such that $v \notin X_i$ and $X_i = X_j \cup \{v\}$, or
4. a *join node* with two children j and k such that $X_i = X_j = X_k$.

Any graph G of tree-width w has a nice tree decomposition of width $w + 1$; furthermore, a such tree decomposition can be constructed in time $f(w)n$ for some function f [22, 88]. As the function f grows fast in this case, it may be desirable in practice to use approximation algorithms with better or no dependence on w; see survey by Bodlaender [23].

Bounded tree-width Bayesian networks. Let $D = (V, A)$ be a directed graph. The *moralised graph* of D is the undirected graph $\mathcal{M}(G) = (V, E)$, where E consists of

1. edges $\{u, v\}$ for $(u, v) \in A$, and
2. edges $\{u, v\}$ for each *v-structure* $\{(u, w), (v, w)\} \subseteq A$.

We remark that the latter type of edges are sometimes called *marriage edges*; the graph $\mathcal{M}(G)$ is called moralised as all pairs that have common children are married [99].

For the purposes of our discussion of Bayesian networks, we define the *tree-width of a directed acyclic graph D* as the tree-width of the moralised graph $\mathcal{M}(D)$; see Figure 3.4. The reason for this definition is that the complexity of inference in Bayesian networks is controlled by the tree-width of the moralised graph [98, 99]; intuitively, if the parent set of v is A_v, then the inference algorithm must consider all subsets of $v \cup A_v$.

3.2.3 Ingredient: finding bounded tree-width graphs

Introduction. The main technical tool of Paper III is a dynamic programming algorithm for finding a tree decomposition on the base set that minimises a given cost function, or more generally, for evaluating certain types of multilinear forms defined in the terms of tree decompositions. Paper III uses this machinery to find bounded tree-width Bayesian network structures, and in this section we also apply it to maximum bounded tree-width subgraph problems.

We note that the basic idea we employ is conceptually quite similar to the Bellman–Held–Karp-style dynamic programming [7, 8, 73] for *permutation problems*. Following the formalisation of Koivisto and Parviainen [91], a permutation problem of degree d is the evaluation of an expression of the form

$$\bigoplus_{\sigma} \bigodot_{i=1}^{n} f_i(\{\sigma_1, \sigma_2, \ldots, \sigma_i\}, \sigma_{i-d+1}\sigma_{i-d+2}\cdots\sigma_i)\,, \tag{3.2}$$

where σ ranges over all permutations of $[n]$ and functions f_i are given as input. For example, travelling salesman with integer weights can be formulated as a permutation problem over semiring $(\mathbb{Z} \cup \{-\infty\}, \max, +)$, with $d = 2$ and weight $f_i(X, uv)$ being the weight of edge $\{u, v\}$.

Tree-decomposition problems. Let $\mathcal{T} = (\mathcal{X}, T, r)$ be a nice tree decomposition over a vertex set V; for the purposes of this section, we do not require that tree decompositions are associated with a fixed graph. The definition of nice tree decomposition implies that for any vertex $v \in V$ that is not in the root bag X_r, there is a unique forget node for v in the tree decomposition $\mathcal{T}$; we denote this node by $\mathrm{f}(\mathcal{T}, v)$.

Now let $f\colon \binom{V}{\downarrow w} \times v \to S$ be a function with values in semiring $(S, \oplus, \odot)$. For a tree decomposition $\mathcal{T}$ of width at most w, we define the *cost* of $\mathcal{T}$ as

$$\bigodot_{v \in V} f\left(X_{\mathrm{f}(\mathcal{T},v)}, v\right).$$

A *tree-decomposition problem of width w* asks us to optimise this cost over all nice tree decompositions, that is, to compute the multilinear form

$$\tau(f) = \bigoplus_{\mathcal{T}} \bigodot_{v \in V} f\left(X_{\mathrm{f}(\mathcal{T},v)}, v\right),$$

where $\mathcal{T} = (\mathcal{X}, T, r)$ ranges over nice tree decompositions of width at most w with $X_r = \emptyset$. Note in particular that the condition that the root bag is empty ensures that each vertex $v \in V$ has a forget node.

To evaluate the multilinear form $\tau(f)$, we will compute intermediate values $h(X,U)$ for disjoint sets $X,U \in 2^V$ such that $|X| \leq w+1$, defined by

$$h(X,U) = \bigoplus_{\mathcal{T}} \bigodot_{v \in U} f\big(X_{\mathrm{f}(\mathcal{T},v)}, v\big)\,,$$

where $\mathcal{T}$ ranges over tree decompositions on vertex set $X \cup U$ such that $X_r = X$; indeed, we have that $\tau(f) = h(\emptyset, V)$. The computation of the values $h(X,U)$ can now be done using the recurrence

$$\begin{aligned} h(X,\emptyset) &= 0\,, \\ h(X,U) &= h^{\mathrm{j}}(X,U) \oplus h^{\mathrm{i}}(X,U) \oplus h^{\mathrm{f}}(X,U)\,, \end{aligned} \tag{3.3}$$

where

$$\begin{aligned} h^{\mathrm{j}}(X,U) &= \bigoplus_{\substack{U_1 \cup U_2 = U \\ U_1, U_2 \neq \emptyset}} \Big[h(X,U_1) \odot h(X,U_2)\Big]\,, \\ h^{\mathrm{i}}(X,U) &= \bigoplus_{x \in X} h\Big(X \setminus \{x\}, U\Big)\,, \\ h^{\mathrm{f}}(X,U) &= \bigoplus_{x \in U} \Big[h\Big(X \cup \{x\}, U \setminus \{x\}\Big) \odot g\Big(X \setminus \{x\}, x\Big)\Big]\,. \end{aligned}$$

As special cases, we define $h^{\mathrm{j}}(X,U) = 0$ if $|U| = 1$ and $h^{\mathrm{f}}(X,U) = 0$ if $|X| = w+1$. To see that this recurrence works as intended, we observe that values $h^{\mathrm{j}}(X,U)$, $h^{\mathrm{i}}(X,U)$ and $h^{\mathrm{f}}(X,U)$ are restrictions of $h(X,U)$ where the root of the tree decompositions in the sum is required to be a join node, an introduce node and a forget node, respectively.

The recurrence (3.3) can be evaluated in the uniform algebraic model from bottom up in the usual dynamic programming manner. There are $O(\binom{n}{w+1}2^n w)$ index pairs (X,U) to consider, and the verbatim evaluation of (3.3) requires $\binom{n}{w}3^n n^{O(1)}$ time, as all partitions of U have to be considered when evaluating $h^{\mathrm{j}}(X,U)$. However, when operating over rings, fast subset convolution [13] can be used to speed this up to $\binom{n}{w}2^n n^{O(1)}$. Thus, we have the following theorem.

Theorem 3.10. *The multilinear form $\tau(f)$ can be evaluated in the uniform algebraic model*

(i) *in $\binom{n}{w}2^n n^{O(1)}$ time and space over rings, and*

(ii) *in $\binom{n}{w}3^n n^{O(1)}$ time and $\binom{n}{w}2^n n^{O(1)}$ space over semirings.*

Applications. Theorem 3.10 can be used to solve various bounded tree-width subgraph problems. In the simplest variant, we are given a graph $G = (V, E)$ and positive integers K and w, and the task is to decide whether there is a subgraph of G that has at least K edges and tree-width at most w; note that this problem is NP-hard even for fixed $w = 2$ [122]. To apply Theorem 3.10, we define $f(X, v)$ as the number of edges with one endpoint v and the other endpoint in X. Now the value $\tau(f)$ evaluated over the semiring $(\mathbb{Z} \cup \{-\infty\}, \max, +)$ gives us the weight of the maximum subgraph of G with tree-width at most w; indeed,

1. for each subgraph H of G with tree-width at most w, the sum $\tau(f)$ includes a tree decomposition that is compatible with H, and
2. for each tree decomposition $\mathcal{T}$ included in the sum $\tau(f)$, the value $\sum_{v \in V} f\big(X_{\mathrm{f}(\mathcal{T},v)}, v\big)$ is the number of edges in the maximum subgraph of G compatible with $\mathcal{T}$.

Furthermore, for bounded integers the subset convolution can be evaluated efficiently over the semiring $(\mathbb{Z} \cup \{-\infty\}, \max, +)$ [13], enabling us to use the faster version of Theorem 3.10. The algorithm also generalises to edge-weighted graphs, giving us the following theorem.

Theorem 3.11. *Given a graph $G = (V, E)$ with edge weights and positive integers K and w, we decide whether G has a subgraph of weight at least K and tree-width at most w*

(i) *in time and space $\binom{n}{w} 2^n n^{O(1)}$ when all edges have weight 1,*

(ii) *in time and space $\binom{n}{w} 2^n n^{O(1)} M \log M$ when weights are positive integers bounded by M, and*

(iii) *in time $\binom{n}{w} 3^n n^{O(1)}$ and space $\binom{n}{w} 2^n n^{O(1)}$.*

Bayesian network learning. Using the machinery established in this section, Paper III shows that learning bounded tree-width Bayesian network structures can be done in single-exponential time for any fixed $w \geq 2$. Specifically, we prove the following theorem.

Theorem 3.12. *Given a vertex set V, an integer $w \geq 1$ and a scoring functions f_v for each $v \in V$, we can find a directed acyclic graph $D = (V, A)$ of tree-width at most w maximising the score*

$$f(D) = \sum_{v \in V} f_v(A_v)$$

in time $3^n n^{w+O(1)}$.

The proof of Theorem 3.12 also requires another result from Paper III; Theorem 2 of the paper shows that if we are given a graph $G = (V, E)$, it is fixed-parameter tractable in the tree-width of G to find an optimal acyclic orientation of G that does not introduce new edges in moralisation. The algorithm of Theorem 3.12 essentially runs this algorithm on top of dynamic programming over tree decompositions;

1. we use dynamic programming over all nice tree decompositions to cover all possible moralised graphs of tree-width at most w, and
2. for each step of the dynamic programming algorithm, we use the orientation algorithm on the new root bag to select best orientation.

3.3 Cutting a few vertices from a graph

This section is based on Paper IV.

3.3.1 Introduction

Minimum and maximum cuts. *Minimum cut* and *maximum cut* are two fundamental cut problems; both of them ask for a partition of the vertex set of a graph $G = (V, E)$ into two parts X and Y, with the objective being to either minimise or maximise the number of edges crossing the partition. However, these problems exhibit very different complexity behaviours; indeed, minimum cut is polynomial-time solvable [65], while maximum cut is NP-hard [82].

However, if we add a size constraint to minimum cut, that is, we require that one side of the partition (X, Y) covers at least some fraction or at most some fraction of the vertices, then the problem becomes NP-hard [68]. This holds even for *minimum bisection*, which asks for a minimum cut (X, Y) with $||X| - |Y|| \leq 1$.

Small unbalanced cuts. In this section, we consider minimum cut problems with size constraints in a case where we require that either the cut itself or the smaller side of the partition is very small. In this setting, it makes sense to consider parameterised algorithms for the problem. That is, we look at variants of what we call *cutting a few vertices from a graph*, or for the sake of brevity, the (k, t)-*unbalanced cut* problem; we are given a graph $G = (V, E)$ and integers $k, t \geq 0$, and the task is to find a partition of V into disjoint non-empty sets X and Y such that $|X| \leq k$ and the cut (X, Y) has size at most t.

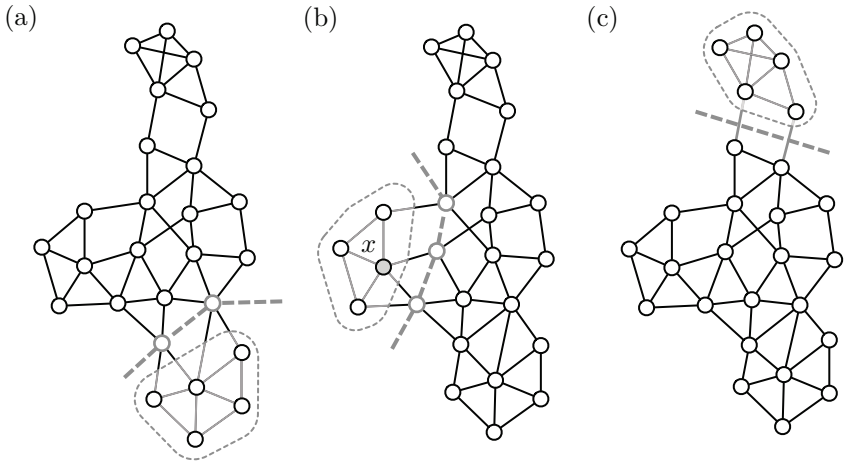


Figure 3.5: (a) Unbalanced vertex cut, (b) unbalanced vertex cut with terminal x, and (c) unbalanced edge cut.

There are several natural variants of the problem, as illustrated in Figure 3.5; we may

1. consider either edge cuts or vertex cuts,
2. require that X contains a distinguished vertex $x \in V$, called a *terminal*, which is given as a part of the input,
3. require that X has either size at most k or size exactly k, and
4. consider the complexity of the problem when the complexity parameter is (a) k alone, (b) t alone, or (c) both k and t.

Table 3.1 gives an overview of known fixed-parameter tractability and hardness results for the different variants of the problem, including the results from Paper IV.

Paper IV. Paper IV focuses on vertex-cut versions of the (k, t)-unbalanced cut problem; that is, the task is to find a partition (X, S, Y) such that $1 \leq |X| \leq k$, $|S| \leq t$ and S separates X from Y. For (k, t)-unbalanced vertex cut and its terminal variant, fixed-parameterised tractability when parameterised by both k and t can be established using standard two-colour colour-coding techniques [1, 30]. We omit detailed discussion of these results here; see Paper IV.

The main results of Paper IV shows that the (k, t)-unbalanced vertex cut problem is fixed-parameter tractable also when parameterised by t alone,

	Complexity parameter		
Variant	k only	t only	both k and t
vertex cut	**W[1]-hard**	**FPT**	FPT [106]
vertex cut, terminal	**W[1]-hard**	**W[1]-hard**	**FPT**
vertex cut, $\lvert X\rvert = k$	W[1]-hard [106]	W[1]-hard [106]	W[1]-hard [106]
edge cut	(solvable in polynomial time [2, 132])		
edge cut, terminal	FPT [100]	FPT [100]	FPT [100]
edge cut, $\lvert X\rvert = k$	W[1]-hard [29]	(open)	FPT [33]

Table 3.1: Parameterised complexity of (k, t)-unbalanced cut variants; for details, see Paper IV. All variants except the (k, t)-unbalanced edge cut are NP-hard when k and t are part of the input. Results indicated in boldface are from Paper IV.

as stated in Theorem 3.16 below. This result is obtained via the use of the *important separators* technique [106]. Specifically, we use the important separators to enable an efficient decomposition of unbalanced cuts based on the combinatorial lemmas of Paper IV.

► *Section 3.3.2 presents the basic theory of important separators and their use in Paper IV.*

3.3.2 Ingredient: important separators

Introduction. Important separators is a useful technique for parameterised graph cut problems, introduced by Marx [106]; in essence, given vertices s and t, important (s, t)-separators are small (s, t)-separators that are as far away from s as possible. The machinery was originally used to establish fixed-parameter tractability of multiway cuts and multicuts, but related techniques have since then been used in various guises to establish fixed-parameter tractability of other cut-type problems, such as directed feedback vertex set [37] and almost 2-SAT [115].

Paper IV uses important separators in the context of vertex cuts, but the machinery is also applicable to edge cuts almost without modifications. We refer to Lokshtanov and Marx [100] for a self-contained presentation of important edge cuts.

Definitions and basic theory. Let $G = (V, E)$ be a graph. For disjoint sets $X, Y \subseteq V$, an (X, Y)-*separator* is a vertex set $S \subseteq V \setminus (X \cup Y)$ such that there is no path between sets X and Y in $G[V \setminus S]$. An (X, Y)-separator S is *minimal* if no subset of S is an (X, Y)-separator. We write $R(X, S)$ for the set of vertices *reachable* from X in $G[V \setminus S]$, that is, for the set of vertices v such that there is a path from X to v in $G[V \setminus S]$.

Definition 3.13. A minimal (X, Y)-separator S is *important* if there is no (X, Y)-separator T such that $|T| \leq |S|$ and $R(X, S)$ is a proper subset of $R(X, T)$.

Important separators have the convenient property of being algorithmically easy to handle; Marx [106] showed that listing all important (X, Y)-separators up to size t is fixed-parameter tractable. The bound in the following lemma follows implicitly from Chen et al. [38].

Lemma 3.14. *There are at most 4^t important (X, Y)-separators of size at most t, and these separators can be listed in time $4^t n^{O(1)}$.*

Lemma 3.15 ([106])**.** *If there exists an (X, Y)-separator, then there is exactly one important (X, Y)-separator of minimum size, and it can be found in polynomial time.*

Application to unbalanced vertex cut. As we stated before, the main results of Paper IV uses important separators to establish fixed-parameter tractability of the (k, t)-unbalanced vertex cut problem. The exact statement of the theorem is as follows.

Theorem 3.16. *The (k, t)-unbalanced vertex cut problem can be solved in time $4^t n^{O(1)}$.*

The main ingredients of the proof of Theorem 3.16 are as follows. For full details, see Paper IV.

1. For (k, t)-unbalanced vertex cuts (X, S, Y) such that the smaller side X is inclusion-minimal among all (k, t)-unbalanced vertex cuts, Lemmas 6–9 of Paper IV give a combinatorial characterisation in terms of important separators. The proof of these lemmas rely heavily on the minimality of X and on the *sub-modularity* of the neighbourhood size function, i.e. the inequality
$$|N(A \cap B)| + |N(A \cup B)| \leq |N(A)| + |N(B)| \ .$$
2. When k is roughly equal to t or larger, the combinatorial lemmas can be used to find a (k, t)-unbalanced cut in time $4^t n^{O(1)}$ via Lemma 3.14.

3. On the other hand, when k is smaller than t, we can apply a fixed-parameter algorithm with parameters k and t; specifically, the colour-coding algorithm presented in Theorem 1 of Paper IV runs in time $2^{k+t}(k+t)^{O(\log(k+t))}n^{O(1)}$.

Chapter 4

Algebraic catalysts

In this chapter, we consider the evaluation of linear and multilinear forms in the presence of different *algebraic catalysts*, focusing in particular on the difference between rings and idempotent semirings. The chapter is partially based on Paper V. The presentation is also influenced by a follow-up work by the present author and others not included in this thesis, to which we refer as Find et al. [56].

4.1 Introduction

Many fast multilinear form evaluation algorithms rely on some *algebraic catalyst*, that is, some convenient algebraic property of the ground semiring. For example:

1. Fast matrix multiplication [45, 124, 125, 129] and subset convolution [13] over rings both use the existence of *additive inverses*.
2. Fourier transforms [42] use the existence of *roots of unity*.
3. Algebraic sieving techniques exploit *cancellation modulo* 2 and can be used, for example, to test Hamiltonicity in undirected graphs in time $1.657^n n^{O(1)}$ [11]; see also Björklund et al. [18, 20].
4. The Hasse diagram of a lattice L gives an optimal algorithm for the zeta transform on L over *commutative idempotent semigroups*, while equally fast algorithms are not known for rings [19, 87].

There are also algorithmic results that are essentially based on some matrix having a low rank over a suitable semiring; recall that the *rank* of a $n \times n$ matrix M over a semiring S is the smallest value r such that there is $n \times r$ matrix P and $r \times n$ matrix Q such that $M = PQ$. For example:

1. The (n, p, q)-disjointness matrix has rank at most $\binom{p+q}{p} 2^{o(p+q)} \log n$ over the Boolean semiring $(\{0,1\}, \vee, \wedge)$ [26]. Similar results hold for independent-union matrices of linear matroids [103], and algorithmic versions of these results have been developed and applied in the context of fixed-parameter algorithms [61, 97, 105–107].
2. The $2^{\Theta(n \log n)} \times 2^{\Theta(n \log n)}$ *matching connectivity matrix* of a complete graph on n vertices has rank $2^{n/2-1}$ over the field $(\mathbb{Z}_2, +, \cdot)$ [25, 49]; this result has been used to obtain fast Hamiltonicity testing algorithms in exact and bounded tree-width settings.

Given the above variety of algebraic catalysts that have been successfully employed in exact algorithmics, we are interested in how things change when specific algebraic catalysts are not available due to a constrained choice of semiring. While it is generally well understood that evaluation over semirings is strictly more difficult than over rings, that is, monotone models are weaker than non-monotone models, we will focus on the question whether idempotent addition can be more powerful algebraic catalyst than invertible addition in some cases. Specifically, we will consider this question within the context of linear form evaluation; the main question is then whether there are linear forms that are strictly easier to evaluate over commutative idempotent semigroups than over commutative groups.

- *Section 4.2 gives an overview of related circuit complexity results.*
- *Section 4.3 discusses intersection matrices and conditional complexity results implying that the corresponding linear forms are easier to evaluate over commutative idempotent semigroups than over commutative groups.*

4.2 Relative complexity of linear forms

Introduction. The circuit complexity of linear forms provides a convenient framework for systematic study of the relative power of different algebraic catalysts [56, 80]. Specifically, we take a fixed Boolean matrix $M \in \{0,1\}^{n \times n}$ and consider the circuit complexity of the linear maps $x \mapsto Mx$ in three circuit models over different commutative semigroups;

1. $+$-circuits, i.e. circuits over semigroup $(\mathbb{N}, +)$,
2. $\vee$-circuits, i.e. circuits over semigroup $(\{0,1\}, \vee)$, and
3. $\oplus$-circuits, i.e. circuits over group $(\{0,1\}, \oplus)$, where $\oplus$ is the exclusive or operation, or equivalently, addition modulo 2.

Note that these specific choices can be seen as prototypical examples of commutative semigroups with different properties. Indeed, a circuit for one of these models can be used to evaluate $x \mapsto Mx$ over any commutative semigroup with similar properties; (a) $+$-circuit works for any commutative semigroup, (b) $\vee$-circuit works for any commutative idempotent semigroup, and (c) $\oplus$-circuit works for any commutative group where elements are their own inverses.

Definitions and basic results. For matrix $M \in \{0,1\}^{n\times n}$, let $C_\vee(M)$, $C_+(M)$, and $C_\oplus(M)$ denote the minimum number of wires in an unbounded fan-in circuit for computing $x \mapsto Mx$ in the respective models. For $\mathsf{X}, \mathsf{Y} \in \{\vee, +, \oplus\}$, we define the *relative complexity* of X/Y as

$$\rho_{\mathsf{X}/\mathsf{Y}}(n) = \max_{M\in\{0,1\}^{n\times n}} C_{\mathsf{X}}(M)/C_{\mathsf{Y}}(M)\,.$$

We observe the following basic properties of these complexity measures.

1. As a $+$-circuit works for any commutative semigroup, we have

$$\begin{aligned}\rho_{\vee/+}(n) &= \rho_{\oplus/+}(n) = 1\,, \text{ and}\\ \rho_{+/\oplus}(n) &\geq \rho_{\vee/\oplus}(n)\,.\end{aligned}$$

2. For any $M \in \{0,1\}^{n\times n}$ and $\mathsf{X} \in \{\vee, +, \oplus\}$, we have

$$C_{\mathsf{X}}(M) = O(n^2/\log n)$$

by a result of Lupanov [104]. Thus for any $\mathsf{X}, \mathsf{Y} \in \{\vee, +, \oplus\}$, we have

$$\rho_{\mathsf{X}/\mathsf{Y}}(n) = O(n/\log n)\,.$$

3. In fact, standard counting arguments (see e.g. Jukna [79, Section 1.4]) and Lupanov's upper bound imply that for most matrices $M \in \{0,1\}^{n\times n}$, we have

$$C_{\mathsf{X}}(M) = \Theta(n^2/\log n)$$

for $\mathsf{X} \in \{\vee, +, \oplus\}$. This implies in particular that counting arguments and random matrices do not directly separate the different models.

Non-trivial separations. We will briefly summarise the known separation results below, based on Find et al. [56].

1. The relative complexity of $\vee/\oplus$ has been explicitly studied; in particular Gashkov and Sergeev [69] and Grinchuk and Sergeev [70] showed that $\rho_{\vee/\oplus} = \Omega(n/(\log^6 n \log\log n))$. Boyar and Find [27] improved the separation by showing that

$$\rho_{\vee/\oplus}(n) = \Omega(n/\log^2 n)\,, \text{ and thus}$$
$$\rho_{+/\oplus}(n) = \Omega(n/\log^2 n)\,.$$

   Furthermore, Find et al. [56] and Jukna and Sergeev [80] have subsequently given progressively simpler proofs for the same separation based on known complexity results.

2. The only known result on the relative complexity of $+/\vee$ is also due to Find et al. [56]. Specifically, they show that

$$\rho_{+/\vee}(n) = \Omega(n^{1/2}/\log^2 n)\,.$$

For $\rho_{\oplus/\vee}(n)$, no super-constant separation results are known. The main obstacle here is that proving lower bounds for non-monotone complexity has generally proven to be difficult, and no applicable lower bounds are known. However, Find et al. [56] conjecture that we also have

$$\rho_{+/\vee}(n) = n^{1-o(1)}\,, \text{ and}$$
$$\rho_{\oplus/\vee}(n) = n^{1-o(1)}\,.$$

The results of Paper V give some evidence in this direction, as we will discuss in Section 4.3.

4.3 Uniform algorithms for intersection matrices

This section is based on Paper V.

4.3.1 Introduction

In this section, we aim to provide some evidence for the conjectures that $\rho_{+/\vee}(n) = n^{1-o(1)}$ and $\rho_{\oplus/\vee}(n) = n^{1-o(1)}$, based on results of Paper V. We will consider *intersection matrices* of arbitrary set families. These matrices have small rank over idempotent semirings, and thus the corresponding linear forms, that is, *intersection transforms*, have small circuits over commutative idempotent semigroups.

Specifically, we will show that non-trivial intersection transforms over commutative rings could be used to implement a meet-in-the-middle style

algorithm for CNF-SAT with running time $2^{(1-\varepsilon)n}(n+m)^{O(1)}$ for $\varepsilon > 0$, where n is the number of variables and m is the number of clauses. This would violate the *strong exponential time hypothesis*, which asserts that brute force is essentially optimal for CNF-SAT.

However, it should be noted that there are two major caveats. Firstly, the result does not imply non-uniform circuit lower bounds, as even if the strong exponential time hypothesis holds, our result does not rule out the existence of small circuits that cannot be constructed efficiently. Secondly, the strong exponential time hypothesis is a far stronger assumption than even P $\neq$ NP, and it may very well turn out that it does not hold.

- *Section 4.3.2 introduces the strong exponential time hypothesis and related theory.*
- *Section 4.3.3 discusses intersection transforms and meet-in-the-middle for CNF-SAT.*

4.3.2 Ingredient: the strong exponential time hypothesis

Complexity of satisfiability. Even if we assume that P $\neq$ NP, we do not know whether we can solve

1. 3-SAT in time $2^{o(n)}(n+m)^{O(1)}$, that is, in sub-exponential time, or
2. CNF-SAT in time $2^{(1-\varepsilon)n}(n+m)^{O(1)}$ for some $\varepsilon > 0$.

Following the influential work of Impagliazzo, Paturi, and Zane [75, 76], the connection between these questions and the complexity of other hard problems have been investigated, as we will discuss further below in detail. In short, if we assume that the answer to either of these two questions is negative – these assumptions are known as the *exponential time hypothesis* [75] and the *strong exponential time hypothesis* [32, 76], respectively – we can then derive conditional lower bounds for other problems. These results can be seen as hardness results or new attacks on the complexity of CNF-SAT, depending on whether one agrees with the hypotheses or not.

Definitions. For $k \geq 2$, let s_k be the infimum of real numbers δ such that there exists an algorithm for k-SAT with running time $2^{\delta n}(n+m)^{O(1)}$. Note that 2-SAT is known to be in P and CNF-SAT can be solved in time $2^n(n+m)^{O(1)}$, and thus we have

$$0 = s_2 \leq s_3 \leq s_4 \leq \cdots \leq 1\,.$$

It follows that the limit $s_\infty = \lim_{k\to\infty} s_k$ exists and we have $0 \leq s_\infty \leq 1$.

Now the precise statements of the two hypotheses are as follows.

1. The *exponential time hypothesis* is the hypothesis that $s_3 > 0$.
2. The *strong exponential time hypothesis* is the hypothesis that $s_\infty = 1$.

In particular, the strong exponential time hypothesis implies that CNF-SAT cannot be solved in time $2^{(1-\varepsilon)n}(n+m)^{O(1)}$ for any $\varepsilon > 0$. However, note that the reverse does not hold, as it is possible in the light of current understanding that $s_\infty < 1$ but there is still no $2^{(1-\varepsilon)n}m^{O(1)}$ time algorithm for CNF-SAT for any $\varepsilon > 0$ [114].

Consequences of the strong exponential time hypothesis. In this work, we will restrict our attention to the strong exponential time hypothesis. One interpretation of this hypothesis is that it states that brute-force search over the whole solution space is essentially optimal for CNF-SAT, that is, exponential speed-up is not possible. Furthermore, it is possible to show that if the strong exponential time hypothesis holds, then brute force is also essentially optimal for various other problems. For example, if the strong exponential time hypothesis holds, then

1. hitting set and set splitting cannot be solved in time $2^{(1-\varepsilon)n}(n+m)^{O(1)}$ for any $\varepsilon > 0$, where n is the number of elements and m is the number set in the input [48],
2. $\oplus$CNF-SAT, that is, the problem of counting the number solutions modulo 2 of a CNF formula, cannot be solved in time $2^{(1-\varepsilon)n}(n+m)^{O(1)}$ for any $\varepsilon > 0$ [31], and
3. for any constant $k \geq 3$, k-dominating set cannot be solved in time $O(n^{(1-\varepsilon)k})$ for any $\varepsilon > 0$ [114].

It is also known that many fixed-parameter algorithms for problems parameterised by tree-width are optimal under the strong exponential time hypothesis [101]. See the cited papers and a survey by Lokshtanov et al. [102] for a complete picture of lower bounds based on the exponential time hypotheses.

4.3.3 Ingredient: Intersection matrices

Intersection matrices. Let $\mathcal{F} = \{F_1, F_2, \ldots, F_s\}$ and $\mathcal{E} = \{E_1, E_2, \ldots, E_t\}$ be collections of subsets of $[\ell]$; for the purposes of this section, it is convenient to allow duplicates in collections $\mathcal{F}$ and $\mathcal{E}$. The *intersection matrix* $N(\mathcal{E}, \mathcal{F})$ is the matrix

$$N\colon [t] \times [s] \to S\,, \qquad\qquad N(j,i) = [E_j \cap F_i \neq \emptyset]\,.$$

That is, the matrix $N(\mathcal{E},\mathcal{F})$ is the binary complement of the disjointness matrix $D(\mathcal{E},\mathcal{F})$.

The intersection matrix $N(\mathcal{E},\mathcal{F})$ defines the $(\mathcal{F},\mathcal{E})$-*intersection transform* over commutative semigroups $(S,\oplus)$, defined as

$$\iota\colon ([s]\to S)\to([t]\to S)\,,\qquad (f\iota)(j)=\bigoplus_{i\colon F_i\cap E_j\neq\emptyset} f(i)\,.$$

The trivial algorithm for intersection transform runs in time $O(st\ell)$ on the uniform algebraic model, assuming reasonable representation of $\mathcal{E}$ and $\mathcal{F}$.

Idempotent intersection transform. The matrix $N(\mathcal{E},\mathcal{F})$ has a low rank over any commutative semiring S with idempotent addition operation. Specifically, let us define two matrices

$$P\colon [s]\times[\ell]\to S\,,\qquad P(j,k)=[k\in E_j]\,,$$
$$Q\colon [\ell]\times[t]\to S\,,\qquad Q(k,i)=[k\in F_i]\,.$$

Clearly, $N(\mathcal{E},\mathcal{F})=PQ$ over S, and thus $N(\mathcal{E},\mathcal{F})$ has rank ℓ over S.

The low rank of the intersection matrix gives us a fast algorithm for the intersection transform in the uniform algebraic model over commutative idempotent semigroups. That is, given input function $f\colon[s]\to S$, we first compute for each $k\in[\ell]$ the sum

$$h(k)=\bigoplus_{i\colon k\in F_i} f(i)\,,$$

and then we recover the output for each $j\in[t]$ as

$$(f\iota)(j)=\bigoplus_{k\colon k\in E_j} h(k)\,.$$

This algorithm runs in time $O\big((s+t)\ell\big)$ in the uniform algebraic model.

General intersection transform. In contrast to the idempotent case, we will now show that fast intersection transforms over arbitrary commutative groups would violate the strong exponential time hypothesis. We essentially follow Williams [133], who considered the possibility of a meet-in-the-middle approach for CNF-SAT. We will show that a fast intersection transform could be used to efficiently implement the combination step of such a meet-in-the-middle algorithm.

For collections $\mathcal{F}=\{F_1,F_2,\ldots,F_s\}$ and $\mathcal{E}=\{E_1,E_2,\ldots,E_t\}$ of subsets of $[\ell]$, we say that (i,j) is a *covering pair for collections* $\mathcal{F}$ *and* $\mathcal{E}$ if $F_i\cup E_j=[\ell]$. The following lemma is a variant of a theorem of Williams [133, Theorem 5.1].

Lemma 4.1. *If we can, for all collections* $\mathcal{F} = \{F_1, F_2, \ldots, F_s\}$ *and* $\mathcal{E} = \{E_1, E_2, \ldots, E_t\}$ *of subsets of* $[\ell]$*,*

(i) *count the number of covering pairs for* $\mathcal{F}$ *and* $\mathcal{E}$ *in time* $(st)^{1-\varepsilon}\ell^{O(1)}$ *for some fixed* $\varepsilon > 0$*, then #CNF-SAT can be solved in time*

$$2^{(1-\varepsilon)n}(n+m)^{O(1)}, \text{ and}$$

(ii) *count the number of covering pairs for* $\mathcal{F}$ *and* $\mathcal{E}$ *modulo* 2 *in time* $(st)^{1-\varepsilon}\ell^{O(1)}$ *for some fixed* $\varepsilon > 0$*, then* $\oplus$*CNF-SAT can be solved in time*

$$2^{(1-\varepsilon)n}(n+m)^{O(1)}.$$

The basic idea of the proof is similar to the subset sum algorithm of Horowitz and Sahni [74], as discussed in Section 3.1.1. That is, given an input CNF formula φ with n variables and clause set $\mathcal{C}$ of size m, we

1. split the variables into two subsets of size $n/2$, which we will call *left* and *right* variables,
2. for any assignment x into left variables, we construct a set $L_x \subseteq \mathcal{C}$ of clauses satisfied by x, and similarly R_x for any assignment into right variables, and
3. observe that there is a one-to-one correspondence between covering pairs for families $\{L_x\}$ and $\{R_x\}$ and satisfying assignments of φ.

Theorem 4.2. *If we can evaluate the* $(\mathcal{F}, \mathcal{E})$*-intersection transform for all collections* $\mathcal{F} = \{F_1, F_2, \ldots, F_s\}$ *and* $\mathcal{E} = \{E_1, E_2, \ldots, E_t\}$ *of subsets of* $[\ell]$

(i) *over* $(\mathbb{Z}, +)$ *in time* $(st)^{1-\varepsilon}\ell^{O(1)}$ *for some fixed* $\varepsilon > 0$*, then #CNF-SAT can be solved in time*

$$2^{(1-\varepsilon)n}(n+m)^{O(1)}, \text{ and}$$

(ii) *over* $(\{0,1\}, \oplus)$ *in time* $(st)^{1-\varepsilon}\ell^{O(1)}$ *for some fixed* $\varepsilon > 0$*, then* $\oplus$*CNF-SAT can be solved in time*

$$2^{(1-\varepsilon)n}(n+m)^{O(1)}.$$

Theorem 4.2 is proven by a reduction to Lemma 4.1; in particular, we show that a fast algorithm for intersection summation can be used to count the number of covering pairs for $\mathcal{F}$ and $\mathcal{E}$. We will sketch the proof for part (i); part (ii) is identical except that the arithmetic is performed modulo 2.

1. For sets $X, Y \subseteq [\ell]$, we have that $X \cup Y = [\ell]$ if and only if

$$([\ell] \setminus X) \cap ([\ell] \setminus Y) = \emptyset\,.$$

   Thus, the number of covering pairs in $\mathcal{F} \times \mathcal{E}$ equals the number of disjoint pairs for $\bar{\mathcal{F}}$ and $\bar{\mathcal{E}}$, where $\bar{\mathcal{F}}$ and $\bar{\mathcal{E}}$ indicate collections obtained by taking the complement of each set in $\mathcal{F}$ and $\mathcal{E}$, respectively.

2. Assume that we have an $(st)^{1-\varepsilon}\ell^{O(1)}$ time algorithm for some $\varepsilon > 0$ for $(\mathcal{F}, \mathcal{E})$-intersection transform over $(\mathbb{Z}, +)$. Then we can also evaluate the $(\mathcal{F}, \mathcal{E})$-disjoint transform in the same time via the identity

$$(f\delta)(j) = \Big(\sum_{i=1}^{s} f(i)\Big) - (f\iota)(j)\,,$$

   and, by extension, the disjoint product

$$\Delta(e, f) = \sum_{i,j\colon F_i \cap E_j = \emptyset} f(i)e(j)$$

   of functions $f\colon [s] \to S$ and $e\colon [t] \to S$. Let $\mathbf{1}_{\mathcal{F}}$ and $\mathbf{1}_{\mathcal{E}}$ denote the constant functions with value 1 at each point. Now $\Delta(\mathbf{1}_{\mathcal{E}}, \mathbf{1}_{\mathcal{F}})$ counts the number of pairs (i, j) such that $F_i \cap E_j = \emptyset$.

It now follows from Theorem 4.2 that there are no efficiently constructible circuits for the intersection transform over non-idempotent commutative semigroups or commutative groups unless the strong exponential time hypothesis fails, as such circuits could be used to implement the transform over the groups $(\mathbb{Z}, +)$ or $(\{0, 1\}, \oplus)$. Specifically, we have the following, extended version of Theorem 2 of Paper V.

Corollary 4.3. *For any $\varepsilon > 0$, there is no algorithm that, given any collections $\mathcal{F} = \{F_1, F_2, \ldots, F_s\}$ and $\mathcal{E} = \{E_1, E_2, \ldots, E_t\}$, constructs*

(i) *$+$-circuit for $(\mathcal{F}, \mathcal{E})$-intersection transform in time $(st)^{1-\varepsilon}\ell^{O(1)}$, or*

(ii) *$\oplus$-circuit for $(\mathcal{F}, \mathcal{E})$-intersection transform in time $(st)^{1-\varepsilon}\ell^{O(1)}$,*

unless the strong exponential time hypothesis fails.

References

[1] Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *Journal of the ACM*, 42(4):844–856, 1995. doi:10.1145/210332.210337.

[2] Amitai Armon and Uri Zwick. Multicriteria global minimum cuts. *Algorithmica*, 46(1):15–26, 2006. doi:10.1007/s00453-006-0068-x.

[3] Stefan Arnborg and Andrzej Proskurowski. Linear time algorithms for NP-hard problems restricted to partial k-trees. *Discrete Applied Mathematics*, 23(1):11–24, 1989. doi:10.1016/0166-218X(89)90031-0.

[4] Stefan Arnborg, Jens Lagergren, and Detlef Seese. Easy problems for tree-decomposable graphs. *Journal of Algorithms*, 12(2):308–340, 1991. doi:10.1016/0196-6774(91)90006-K.

[5] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, New York, 2009.

[6] Francis R. Bach and Michael I. Jordan. Thin junction trees. In *15th Annual Conference on Neural Information Processing Systems (NIPS 2001)*. MIT Press, 2001.

[7] Richard Bellman. Combinatorial processes and dynamic programming. In *Combinatorial Analysis*, volume 10 of *Proceedings of Symposia in Applied Mathematics*, pages 217–249. AMS, 1960.

[8] Richard Bellman. Dynamic programming treatment of the travelling salesman problem. *Journal of the ACM*, 9(1):61–63, 1962. doi:10.1145/321105.321111.

[9] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, and Yunshan Zhu. Symbolic model checking without BDDs. In *5th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS 1999)*, pages 193–207. Springer, 1999. doi:10.1007/3-540-49059-0_14.

[10] Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh. *Handbook of Satisfiability*. IOS Press, 2009.

[11] Andreas Björklund. Determinant sums for undirected Hamiltonicity. In *51st IEEE Symposium on Foundations of Computer Science (FOCS 2010)*, pages 173–182. IEEE Computer Society, 2010. doi:10.1109/FOCS.2010.24.

[12] Andreas Björklund, Petteri Kaski, and Łukasz Kowalik. Counting thin subgraphs via packings faster than meet-in-the-middle time. arXiv:1306.4111 [cs.DS], 2013. To appear in *25th ACM/SIAM Symposium on Discrete Algorithms (SODA 2014)*.

[13] Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Fourier meets Möbius: fast subset convolution. In *39th ACM Symposium on Theory of Computing (STOC 2007)*, pages 67–74. ACM, 2007. doi:10.1145/1250790.1250801.

[14] Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. The fast intersection transform with applications to counting paths. arXiv:0809.2489 [cs.DS], 2008.

[15] Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Computing the Tutte polynomial in vertex-exponential time. In *49th IEEE Symposium on Foundations of Computer Science (FOCS 2008)*, pages 677–686. IEEE Computer Society, 2008. doi:10.1109/FOCS.2008.40.

[16] Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Counting paths and packings in halves. In *17th European Symposium on Algorithms (ESA 2009)*, pages 578–586. Springer, 2009. doi:10.1007/978-3-642-04128-0_52.

[17] Andreas Björklund, Thore Husfeldt, and Mikko Koivisto. Set partitioning via inclusion-exclusion. *SIAM Journal on Computing*, 39(2): 546–563, 2009. doi:10.1137/070683933.

[18] Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Narrow sieves for parameterized paths and packings. arXiv:1007.1161 [cs.DS], 2010.

[19] Andreas Björklund, Mikko Koivisto, Thore Husfeldt, Jesper Nederlof, Petteri Kaski, and Pekka Parviainen. Fast zeta transforms for lattices with few irreducibles. In *23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2012)*, pages 1436–1444. SIAM, 2012.

[20] Andreas Björklund, Petteri Kaski, and Łukasz Kowalik. Probably Optimal Graph Motifs. In *30th International Symposium on Theoretical Aspects of Computer Science (STACS 2013)*, pages 20–31, 2013. doi:10.4230/LIPIcs.STACS.2013.20.

[21] Hans L. Bodlaender. Dynamic programming on graphs with bounded treewidth. In *15th International Colloquium on Automata, Languages and Programming (ICALP 1988)*, pages 105–118. Springer Berlin Heidelberg, 1988. doi:10.1007/3-540-19488-6_110.

[22] Hans L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal of Computing*, 25:1305–1317, 1996. doi:10.1137/S0097539793251219.

[23] Hans L. Bodlaender. Discovering treewidth. In *31st Annual Conference on Current Trends in Theory and Practice of Informatics (SOFSEM 2005)*, pages 1–16. Springer, 2005. doi:10.1007/978-3-540-30577-4_1.

[24] Hans L. Bodlaender and Arie M.C.A. Koster. Combinatorial optimization on graphs of bounded treewidth. *The Computer Journal*, 51 (3):255–269, 2008. doi:10.1093/comjnl/bxm037.

[25] Hans L. Bodlaender, Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth. In *40th International Colloquium on Automata, Languages and Programming (ICALP 2013)*. 2013. doi:10.1007/978-3-642-39206-1_17.

[26] Béla Bollobás. On generalized graphs. *Acta Mathematica Hungarica*, 16(3–4):447–452, 1965. doi:10.1007/BF01904851.

[27] Joan Boyar and Magnus G. Find. Cancellation-free circuits in unbounded and bounded depth. In *19th International Symposium on Fundamentals of Computation Theory (FCT 2013)*, pages 159–170. Springer, 2013. doi:10.1007/978-3-642-40164-0_17.

[28] Peter Bürgisser, Michael Clausen, and M. Amin Shokrollahi. *Algebraic Complexity Theory*. Springer, 1997.

[29] Leizhen Cai. Parameterized complexity of cardinality constrained optimization problems. *The Computer Journal*, 51(1):102–121, 2008. doi:10.1093/comjnl/bxm086.

[30] Leizhen Cai, Siu Man Chan, and Siu On Chan. Random separation: A new method for solving fixed-cardinality optimization problems. pages 239–250, 2006. doi:10.1007/11847250_22.

[31] Chris Calabro, Russell Impagliazzo, Valentine Kabanets, and Ramamohan Paturi. The complexity of unique k-SAT: An isolation lemma for k-CNFs. *Journal of Computer and System Sciences*, 74(3): 386–393, 2008. doi:10.1016/j.jcss.2007.06.015.

[32] Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. The complexity of satisfiability of small depth circuits. In *4th International Workshop on Parameterized and Exact Computation (IWPEC 2009)*, pages 75–85. Springer Berlin Heidelberg, 2009. doi:10.1007/978-3-642-11269-0_6.

[33] Yixin Cao. A note on small cuts for a terminal. arXiv:1306.2578 [cs.DS], 2013.

[34] Alexander V. Chashkin. On the complexity of boolean matrices, graphs, and the boolean functions corresponding to them. *Diskretnaya Matematika*, 6(2):42–73, 1994. In Russian. English translation in *Discrete Mathematics and Applications* 4(3):229–257, 1994. doi:10.1515/dma.1994.4.3.229.

[35] Anton Chechetka and Carlos Guestrin. Efficient principled learning of thin junction trees. In *21st Annual Conference on Neural Information Processing Systems (NIPS 2007)*, pages 273–280. MIT Press, 2007.

[36] Jianer Chen, Songjian Lu, Sing-Hoi Sze, and Fenghui Zhang. Improved algorithms for path, matching, and packing problems. In *18th ACM/SIAM Symposium on Discrete Algorithms (SODA 2007)*, pages 298–307. SIAM, 2007.

[37] Jianer Chen, Yang Liu, Songjian Lu, Barry O'sullivan, and Igor Razgon. A fixed-parameter algorithm for the directed feedback vertex set problem. *Journal of the ACM*, 55(5):21:1–21:19, 2008. doi:10.1145/1411509.1411511.

[38] Jianer Chen, Yang Liu, and Songjian Lu. An improved parameterized algorithm for the minimum node multiway cut problem. *Algorithmica*, 55(1):1–13, 2009. doi:10.1007/978-3-540-73951-7_43.

[39] David Maxwell Chickering. Learning Bayesian networks is NP-Complete. In *Learning from Data: Artificial Intelligence and Statistics*

V, pages 121–130. Springer-Verlag, 1996. doi:10.1007/978-1-4612-2404-4_12.

[40] David Maxwell Chickering, David Heckerman, and Christopher Meek. Large-sample learning of Bayesian networks is NP-Hard. *Journal of Machine Learning Research*, 5:1287–1330, 2004.

[41] Yoeng-jin Chu and Tseng-hong Liu. On the shortest arborescence of a directed graph. *Scientia Sinica*, 14(10):1396–1400, 1965.

[42] James W. Cooley and John W. Tukey. An algorithm for the machine calculation of complex Fourier series. *Mathematics of Computation*, 19(90):297–301, 1965. doi:10.1090/S0025-5718-1965-0178586-1.

[43] Gregory F. Cooper. The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence*, 42: 393–405, 1990. doi:10.1016/0004-3702(90)90060-D.

[44] Gregory F. Cooper and Edward Herskovits. A Bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9:309–347, 1992. doi:10.1007/BF00994110.

[45] Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9(3):251–280, 1990. doi:10.1016/S0747-7171(08)80013-2.

[46] Bruno Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Information and Computation*, 85(1): 12–75, 1990. doi:10.1016/0890-5401(90)90043-H.

[47] Radu Curticapean. Counting matchings of size k is #W[1]-hard. In *40th International Colloquium on Automata, Languages and Programming (ICALP 2013)*, pages 171–181. Springer, 2013. doi:10.1007/978-3-642-39206-1_30.

[48] Marek Cygan, Holger Dell, Daniel Lokshtanov, Daniel Marx, Jesper Nederlof, Yoshio Okamoto, Ramamohan Paturi, Saket Saurabh, and Magnus Wahlstrom. On problems as hard as CNF-SAT. In *Proceedings of the 27th Conference on Computational Complexity (CCC 2012)*, pages 74–84. IEEE, 2012. doi:10.1109/CCC.2012.36.

[49] Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Fast hamiltonicity checking via bases of perfect matchings. In *45th annual ACM symposium on Symposium on Theory of Computing (STOC 2013)*, pages 301–310. ACM, 2013. doi:10.1145/2488608.2488646.

[50] Rina Dechter. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113:41–85, 1999. doi:10.1016/S0004-3702(99)00059-4.

[51] Reinhard Diestel. *Graph theory*. Springer, fourth edition, 2010.

[52] Rodney G. Downey and Michael R. Fellows. Parameterized computational feasibility. In *Feasible Mathematics II*, pages 219–244. Birkhauser, 1994. doi:10.1007/978-1-4612-2566-9_7.

[53] Rodney G. Downey and Michael R. Fellows. *Parameterized complexity*. Springer-Verlag, 1999.

[54] Jack R. Edmonds. Optimum branchings. *Journal of Research of the National Bureau of Standards*, 71B:233–240, 1967.

[55] Gal Elidan and Stephen Gould. Learning bounded treewidth Bayesian networks. *Journal of Machine Learning Research*, 9:2699–2731, 2008.

[56] Magnus G. Find, Mika Göös, Matti Järvisalo, Petteri Kaski, Mikko Koivisto, and Janne H. Korhonen. Separating OR, SUM, and XOR circuits. arXiv:1304.0513 [cs.CC], 2013.

[57] Jörg Flum and Martin Grohe. *Parameterized complexity theory*. Springer-Verlag, 2006.

[58] Jörg Flum and Martin Grohe. The parameterized complexity of counting problems. *SIAM Journal of Computing*, 33(4):892–922, 2004. doi:10.1137/S0097539703427203.

[59] Fedor V. Fomin and Petteri Kaski. Exact exponential algorithms. *Communications of the ACM*, 56(3):80–88, 2013. doi:10.1145/2428556.2428575.

[60] Fedor V. Fomin and Dieter Kratsch. *Exact exponential algorithms*. Springer, 2011.

[61] Fedor V. Fomin, Daniel Lokshtanov, and Saket Saurabh. Efficient Computation of Representative Sets with Applications in Parameterized and Exact Algorithms. arXiv:1304.4626 [cs.DS], 2013. To appear in *25th ACM/SIAM Symposium on Discrete Algorithms (SODA 2014)*.

[62] Fedor V. Fomin, Petr A. Golovach, Jan Kratochvíl, Dieter Kratsch, and Mathieu Liedloff. Sort and search: Exact algorithms for generalized domination. *Information Processing Letters*, 109(14):795–798, 2009. doi:10.1016/j.ipl.2009.03.023.

[63] Fedor V. Fomin, Daniel Lokshtanov, Venkatesh Raman, Saket Saurabh, and B.V. Raghavendra Rao. Faster algorithms for finding and counting subgraphs. *Journal of Computer and System Sciences*, 78(3):698–706, 2012. doi:10.1016/j.jcss.2011.10.001.

[64] Fedor V. Fomin, Petr A. Golovach, and Janne H. Korhonen. On the parameterized complexity of cutting a few vertices from a graph. In *38th International Symposium on Mathematical Foundations of Computer Science (MFCS 2013)*, pages 421–432. Springer, 2013. doi:10.1007/978-3-642-40313-2_38.

[65] Lester R. Ford and Delbert R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956. doi:10.4153/CJM-1956-045-5.

[66] Lance Fortnow. The status of the P versus NP problem. *Communications of the ACM*, 52(9):78–86, 2009. doi:10.1145/1562164.1562186.

[67] Michael R. Garey and David S. Johnson. *Computers and Intractability, A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979.

[68] Michael R. Garey, David S. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1(3):237–267, 1976. doi:10.1016/0304-3975(76)90059-1.

[69] Sergey B. Gashkov and Igor S. Sergeev. On the complexity of linear Boolean operators with thin matrices. *Journal of Applied and Industrial Mathematics*, 5:202–211, 2011. doi:10.1134/S1990478911020074.

[70] Mikhail I. Grinchuk and Igor S. Sergeev. Thin circulant matrices and lower bounds on complexity of some Boolean operators. *Diskretnyi Analiz i Issledovanie Operatsiy*, 18:38–53, 2011. In Russian.

[71] Rudolf Halin. S-functions for graphs. *Journal of Geometry*, 8(1–2): 171–186, 1976. doi:10.1007/BF01917434.

[72] David Heckerman, Dan Geiger, and David M. Chickering. Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, 20(3):197–243, 1995. doi:10.1023/A:1022623210503.

[73] Michael Held and Richard M. Karp. A dynamic programming approach to sequencing problems. *Journal of the Society for Industrial and Applied Mathematics*, 10(1):196–210, 1962. doi:10.1145/800029.808532.

[74] Ellis Horowitz and Sartaj Sahni. Computing partitions with applications to the knapsack problem. *Journal of the ACM*, 21(2):277–292, 1974. doi:10.1145/321812.321823.

[75] Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-SAT. *Journal of Computer and System Sciences*, 62(2):367–375, 2001. doi:10.1006/jcss.2000.1727.

[76] Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001. doi:10.1006/jcss.2001.1774.

[77] Matti Järvisalo, Petteri Kaski, Mikko Koivisto, and Janne H. Korhonen. Finding efficient circuits for ensemble computation. In *15th International Conference on Theory and Applications of Satisfiability Testing (SAT 2012)*, pages 369–382. Springer, 2012. doi:10.1007/978-3-642-31612-8_28.

[78] Matti Järvisalo, Daniel Le Berre, Olivier Roussel, and Laurent Simon. The international SAT solver competitions. *AI Magazine*, 33(1):89–92, 2012.

[79] Stasys Jukna. *Boolean Function Complexity: Advances and Frontiers*. Springer, 2012.

[80] Stasys Jukna and Igor S. Sergeev. Complexity of linear boolean operators. *Foundations and Trends in Theoretical Computer Science*, 9(1):1–123, 2013. doi:10.1561/0400000063.

[81] David Karger and Nathan Srebro. Learning Markov networks: Maximum bounded tree-width graphs. In *12th ACM/SIAM Symposium on Discrete Algorithms (SODA 2001)*, pages 392–401, 2001.

[82] Richard M. Karp. Reducibility among combinatorial problems. In *Symposium on the Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972. doi:10.1007/978-1-4684-2001-2_9.

[83] Richard M. Karp. Dynamic programming meets the principle of inclusion and exclusion. *Operations Research Letters*, 1(2):49–51, 1982. doi:10.1016/0167-6377(82)90044-X.

[84] Petteri Kaski, Mikko Koivisto, Janne H. Korhonen, and Igor S. Sergeev. Fast monotone summation over disjoint sets. Submitted.

[85] Petteri Kaski, Mikko Koivisto, and Janne H. Korhonen. Fast monotone summation over disjoint sets. In *7th International Symposium on Parameterized and Exact Computation (IPEC 2012)*, pages 159–170. Springer, 2012. doi:10.1007/978-3-642-33293-7_16.

[86] Henry A. Kautz and Bart Selman. Planning as satisfiability. In *10th European Conference on Artificial Intelligence (ECAI 1992)*, pages 359–363. Wiley, 1992.

[87] Robert Kennes. Computational aspects of the Möbius transformation of graphs. *IEEE Transactions on Systems, Man and Cybernetics*, 22 (2):201–223, 1992. doi:10.1109/21.148425.

[88] Ton Kloks. *Treewidth: computations and approximations*. Springer, 1994.

[89] Joachim Kneis, Daniel Mölle, Stefan Richter, and Peter Rossmanith. Divide-and-color. In *32nd International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2006)*, pages 58–67. Springer, 2006. doi:10.1007/11917496_6.

[90] Samuel Kohn, Allan Gottlieb, and Meryle Kohn. A generating function approach to the traveling salesman problem. In *Proceedings of the ACM Annual Conference (ACM 1977)*, pages 294–300. ACM, 1977. doi:10.1145/800179.810218.

[91] Mikko Koivisto and Pekka Parviainen. A space-time tradeoff for permutation problems. In *21st ACM/SIAM Symposium on Discrete Algorithms (SODA 2010)*, pages 484–492. SIAM, 2010.

[92] Mikko Koivisto and Kismat Sood. Exact Bayesian structure discovery in Bayesian networks. *Journal of Machine Learning Research*, 5: 549–573, 2004.

[93] Daphne Kollar and Nir Friedman. *Probabilistic graphical models: principles and techniques*. The MIT Press, 2009.

[94] Janne H. Korhonen and Pekka Parviainen. Exact learning of bounded tree-width Bayesian networks. In *16th International Conference on Artificial Intelligence and Statistics (AISTATS 2013)*, pages 370–378. JMLR, 2013.

[95] Ioannis Koutis. Faster algebraic algorithms for path and packing problems. In *35th International Colloquium on Automata, Languages*

and Programming (ICALP 2008), pages 575–586. Springer, 2008. doi:10.1007/978-3-540-70575-8_47.

[96] Ioannis Koutis and Ryan Williams. Limits and applications of group algebras for parameterized problems. In *36th International Colloquium on Automata, Languages and Programming (ICALP 2009)*, pages 653–664. Springer, 2009. doi:10.1007/978-3-642-02927-1_54.

[97] Stefan Kratsch and Magnus Wahlstrom. Representative sets and irrelevant vertices: New tools for kernelization. In *53rd IEEE Symposium on Foundations of Computer Science (FOCS 2012)*, pages 450–459. IEEE, 2012. doi:10.1109/FOCS.2012.46.

[98] Johan H. P. Kwisthout, Hans L. Bodlaender, and Linda C. van der Gaag. The necessity of bounded treewidth for efficient inference in Bayesian networks. In *19th European Conference on Artificial Intelligence (ECAI 2010)*, pages 237–242. IOS Press, 2010. doi:10.3233/978-1-60750-606-5-237.

[99] Steffen L. Lauritzen and David J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society. Series B (Methodological)*, 50(2):157–224, 1988.

[100] Daniel Lokshtanov and Dániel Marx. Clustering with local restrictions. *Information and Computation*, 222:278–292, 2013. doi:10.1016/j.ic.2012.10.016.

[101] Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Known algorithms on graphs of bounded treewidth are probably optimal. In *22nd ACM/SIAM Symposium on Discrete Algorithms (SODA 2011)*, pages 777–789. SIAM, 2011.

[102] Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Lower bounds based on the exponential time hypothesis. *Bulletin of the EATCS*, (105):41–72, 2011.

[103] László Lovász. Flats in matroids and geometric graphs. In *6th British Combinatorial Conference*, pages 45–86. Academic Press London, 1977.

[104] Oleg B. Lupanov. On rectifier and switching-and-rectifier schemes. In *Doklady Akademii Nauk SSSR*, volume 111, pages 1171–1174, 1956. In Russian.

[105] Dániel Marx. A parameterized view on matroid optimization problems. *Theoretical Computer Science*, 410(44):4471–4479, 2009. doi:10.1016/j.tcs.2009.07.027.

[106] Dániel Marx. Parameterized graph separation problems. *Theoretical Computer Science*, 351(3):394–406, 2006. doi:10.1016/j.tcs.2005.10.007.

[107] Burkhard Monien. How to find long paths efficiently. In *Analysis and Design of Algorithms for Combinatorial Problems*, volume 109 of *North-Holland Mathematics Studies*, pages 239–254. Elsevier, 1985. doi:10.1016/S0304-0208(08)73110-4.

[108] Jaroslav Nešetřil and Svatopluk Poljak. On the complexity of the subgraph problem. *Commentationes Mathematicae Universitatis Carolinae*, 26(2):415–419, 1985.

[109] Rolf Niedermeier. *Invitation to fixed-parameter algorithms*. Oxford University Press, 2006.

[110] Sebastian Ordyniak and Stefan Szeider. Algorithms and complexity results for exact Bayesian structure learning. In *26th Conference on Uncertainty in Artificial Intelligence (UAI 2010)*, pages 8–11. AUAI Press, 2010.

[111] Sascha Ott and Satoru Miyano. Finding optimal gene networks using biological constraints. In *14th International Conference on Genome Informatics*, pages 124–133. Universal Academy Press, 2003.

[112] Pekka Parviainen and Mikko Koivisto. Exact structure discovery in Bayesian networks with less space. In *25th Conference on Uncertainty in Artificial Intelligence (UAI 2009)*, pages 436–443. AUAI Press, 2009.

[113] Judea Pearl. *Probabilistic reasoning in intelligent systems*. Morgan Kaufmann, 1988.

[114] Mihai Pătraşcu and Ryan Williams. On the possibility of faster SAT algorithms. In *21st ACM/SIAM Symposium on Discrete Algorithms (SODA 2010)*, 2010.

[115] Igor Razgon and Barry O'Sullivan. Almost 2-sat is fixed-parameter tractable. *Journal of Computer and System Sciences*, 75(8):435–450, 2009. doi:10.1016/j.jcss.2009.04.002.

[116] Bruce Reed, Kaleigh Smith, and Adrian Vetta. Finding odd cycle transversals. *Operations Research Letters*, 32(4):299–301, 2004. doi:10.1016/j.orl.2003.10.009.

[117] Neil Robertson and Paul D. Seymour. Graph minors. II. algorithmic aspects of tree-width. *Journal of Algorithms*, 7(3):309–322, 1986. doi:10.1016/0196-6774(86)90023-4.

[118] Richard Schroeppel and Adi Shamir. A $T = O(2^{n/2})$, $S = O(2^{n/4})$ algorithm for certain NP-complete problems. *SIAM Journal of Computing*, 10(3):456–464, 1981. doi:10.1137/0210033.

[119] Igor S. Sergeev. On additive complexity of a sequence of matrices. arXiv:1209.1645 [cs.DS], 2012.

[120] Tomi Silander and Petri Myllymäki. A simple approach for finding the globally optimal Bayesian network structure. In *22nd Conference on Uncertainty in Artificial Intelligence (UAI 2006)*, pages 445–452. AUAI Press, 2006.

[121] Ajit P. Singh and Andrew W. Moore. Finding optimal bayesian networks by dynamic programming. Technical Report CMU-CALD-05-106, Carnegie Mellon University, 2005.

[122] Nathan Srebro. Maximum likelihood Markov networks: An algorithmic approach. Master's thesis, Massachusetts Institute of Technology, 2000.

[123] Nathan Srebro. Maximum likelihood bounded tree-width Markov networks. In *17th Conference on Uncertainty in Artificial Intelligence (UAI 2001)*, pages 504–511. AUAI Press, 2001.

[124] Andrew James Stothers. *On the complexity of matrix multiplication.* PhD thesis, The University of Edinburgh, 2010.

[125] Volker Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13(4):354–356, 1969. doi:10.1007/BF02165411.

[126] Robert Endre Tarjan and Anthony E. Trojanowski. Finding a maximum independent set. *SIAM Journal on Computing*, 6(3):537–546, 1977. doi:10.1137/0206038.

[127] Leslie G. Valiant. Negation is powerless for boolean slice functions. *SIAM Journal of Computing*, 15:531–535, 1986. doi:10.1137/0215037.

[128] Johan M.M. van Rooij, Hans L. Bodlaender, and Peter Rossmanith. Dynamic programming on tree decompositions using generalised fast subset convolution. In *17th European Symposium on Algorithms (ESA 2009)*, pages 566–577. Springer, 2009. doi:10.1007/978-3-642-04128-0_51.

[129] Virginia Vassilevska Williams. Multiplying matrices faster than Coppersmith-Winograd. In *44th ACM Symposium on Theory of Computing (STOC 2012)*, pages 887–898. ACM, 2012. doi:10.1145/2213977.2214056.

[130] Virginia Vassilevska Williams and Ryan Williams. Finding, minimizing, and counting weighted subgraphs. *SIAM Journal of Computing*, 42(3):831–854, 2013. doi:10.1137/09076619X.

[131] Vijay V. Vazirani. *Approximation Algorithms*. Springer, 2004.

[132] Toshimasa Watanabe and Akira Nakamura. Edge-connectivity augmentation problems. *Journal of Computer and System Sciences*, 35 (1):96–144, 1987. doi:10.1016/0022-0000(87)90038-9.

[133] Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theoretical Computer Science*, 348(2-3):357–365, 2005. doi:10.1016/j.tcs.2005.09.023.

[134] Ryan Williams. Finding paths of length k in $O^*(2^k)$ time. *Information Processing Letters*, 109(6):315–318, 2009. doi:10.1016/j.ipl.2008.11.004.

[135] Frank Yates. *The Design and Analysis of Factorial Experiments*. Imperial Bureau of Soil Science, 1937.

Paper I

Petteri Kaski, Mikko Koivisto, Janne H. Korhonen, and Igor S. Sergeev.

Fast monotone summation over disjoint sets.

Fast Monotone Summation over Disjoint Sets[☆]

Petteri Kaski[a], Mikko Koivisto[b], Janne H. Korhonen[b,*], Igor S. Sergeev[c]

[a] *Helsinki Institute for Information Technology HIIT & Department of Information and Computer Science, Aalto University, P.O. Box 15400, FI-00076 Aalto, Finland*
[b] *Helsinki Institute for Information Technology HIIT & Department of Computer Science, University of Helsinki, P.O. Box 68, FI-00014 University of Helsinki, Finland*
[c] *Lomonosov Moscow State University & Faculty of Mechanics and Mathematics, Department of Discrete Mathematics, Leninskie Gory, Moscow, 119991 Russia*

Keywords: algorithms, arithmetic complexity, commutative semigroup, disjoint summation, k-path, matrix permanent

1. Introduction

Let $(S, +)$ be a commutative semigroup and let $[n] = \{1, 2, \ldots, n\}$. For integers $0 \leq p, q \leq n$, the (n, p, q)-*disjoint summation* problem is as follows. Given a value $f(X) \in S$ for each set $X \subseteq [n]$ of size at most p, the task is to output the function e, defined for each set $Y \subseteq [n]$ of size at most q by

$$e(Y) = \sum_{X \cap Y = \emptyset} f(X)\,, \tag{1}$$

where $X \subseteq [n]$ ranges over sets of size at most p that are disjoint from Y.

We study the arithmetic complexity [6] of (n, p, q)-disjoint summation, with the objective of quantifying how many binary additions in S are sufficient to evaluate (1) for all Y. As additive inverses need not exist in S, this is equivalent to the monotone arithmetic circuit complexity, or equivalently, the monotone arithmetic straight-line program complexity of (n, p, q)-disjoint summation.

Our main result is the following. Let $C_{n,p,q}$ denote the minimum number of binary additions in S sufficient to compute (n, p, q)-disjoint summation, and let us write $\binom{n}{\downarrow k}$ for the sum $\binom{n}{0} + \binom{n}{1} + \cdots + \binom{n}{k}$ of binomial coefficients.

Theorem 1. $C_{n,p,q} \leq \left[p\binom{n}{\downarrow p} + q\binom{n}{\downarrow q}\right] \cdot \min\{2^p, 2^q\}$.

[☆] This paper is an extended version of a conference abstract by a subset of the present authors [10].

[*] Corresponding author. Postal address—see front matter; e-mail address—see below.

Email addresses: petteri.kaski@aalto.fi (Petteri Kaski), mikko.koivisto@cs.helsinki.fi (Mikko Koivisto), janne.h.korhonen@cs.helsinki.fi (Janne H. Korhonen), isserg@gmail.com (Igor S. Sergeev)

2. Related work and applications.

Circuit complexity. If we ignore the empty set, the special case of $(n, 1, 1)$-disjoint summation corresponds to the matrix-vector multiplication task $x \mapsto \bar{I}x$, where $\bar{I}$ is the complement of the identity matrix, that is, a matrix with zeroes on diagonal and ones elsewhere. Results from Boolean circuit complexity concerning this particular task can be transferred to our setting, implying that $(n, 1, 1)$-disjoint summation has complexity exactly $3n - 6$ [7, 14].

For the special case of (n, n, n)-disjoint summation, Yates [18] gave an algorithm evaluating (1) for all $Y \subseteq [n]$ using $2^{n-1}n$ additions in S; Knuth [12, §4.6.4] presents a modern exposition. Furthermore, this bound is tight in the monotone setting [11]. Yates's algorithm has been used as a component in $2^n n^{O(1)}$ time algorithms for graph k-colouring and related covering and packing problems [3].

In a setting where S is a group and we are allowed to take additive inverses in S, it is known that the (n, p, q)-disjoint summation can be evaluated with $O\big(p\binom{n}{\downarrow p} + q\binom{n}{\downarrow q}\big)$ operations in S via the principle of inclusion and exclusion. This has been employed to obtain improved counting algorithms for hard combinatorial problems, most prominently the $\binom{n}{k/2} n^{O(1)}$ algorithm for counting k-paths in a graph by Björklund et al. [4].

Maximisation. An immediate template for applications of (n, p, q)-disjoint summation is maximisation under constraints. If we choose the semigroup to be $(\mathbb{Z} \cup \{-\infty, \infty\}, \max)$, then (1) becomes

$$e(Y) = \max_{X \cap Y = \emptyset} f(X)\,.$$

This can be seen as precomputing the optimal p-subset X when the elements in a q-subset Y are "forbidden", in a setting where the set Y is either not known beforehand or Y will eventually go through almost all possible choices. Such scenario takes place e.g. in a recent work [8] on Bayesian network structure learning by branch and bound.

Semirings. Other applications of disjoint summation are enabled by extending the semigroup by a multiplication operation that distributes over addition. That is, we work over a semiring $(S, +, \cdot)$, and the task is to evaluate the sum

$$\sum_{X \cap Y = \emptyset} f(X) \cdot g(Y)\,, \tag{2}$$

where X and Y range over all disjoint pairs of subsets of $[n]$, of size at most p and q, respectively, and f and g are given mappings to S. We observe that the sum (2) equals $\sum_Y e(Y) \cdot g(Y)$, where Y ranges over all subsets of $[n]$ of size at most q and e is as in (1). Thus, an efficient way to compute e results in an efficient way to evaluate (2).

Counting k-paths. An application of the semiring setup is counting the maximum-weight k-edge paths from vertex s to vertex t in a given graph with real edge weights. Here we assume that we are only allowed to add and compare real numbers and these operations take constant time (cf. [16]). By straightforward Bellman–Held–Karp type dynamic programming [1, 2, 9] we can solve the problem in $\binom{n}{k}n^{O(1)}$ time. Our main result gives an improved algorithm that runs in time $2^{k/2}\binom{n}{k/2}n^{O(1)}$ for even k. The key idea is to solve the problem in halves. We guess a middle vertex v and define $w_1(X)$ as the maximum weight of a $k/2$-edge path from s to v in the graph induced by the vertex set $X \cup \{v\}$; we similarly define $w_2(X)$ for the $k/2$-edge paths from v to t. Furthermore, we define $c_1(X)$ and $c_2(X)$ as the respective numbers of paths of weight $w_1(X)$ and $w_2(X)$ and put $f(X) = (c_1(X), w_1(X))$ and $g(X) = (c_2(X), w_2(X))$. These values can be computed for all vertex subsets X of size $k/2$ in $\binom{n}{k/2}n^{O(1)}$ time. Now the expression (2) equals the number of k-edge paths from s to t with middle vertex v, when we define the semiring operations $\boxplus$ and $\boxdot$ in following manner: $(c, w) \boxdot (c', w') = (c \cdot c', w + w')$ and

$$(c, w) \boxplus (c', w') = \begin{cases} (c, w) & \text{if } w > w', \\ (c', w') & \text{if } w < w', \\ (c + c', w) & \text{if } w = w'. \end{cases}$$

For the more general problem of counting weighted subgraphs Vassilevska and Williams [15] give an algorithm whose running time, when applied to k-paths, is $O(n^{\omega k/3}) + n^{2k/3+O(1)}$, where $2 \leq \omega < 2.3727$ is the exponent of matrix multiplication [17].

Computing matrix permanent. A further application is the computation of the permanent of a $k \times n$ matrix (a_{ij}) over a noncommutative semiring, with $k \leq n$ an even integer, given by $\sum_\sigma a_{1\sigma(1)}a_{2\sigma(2)} \cdots a_{k\sigma(k)}$, where the sum is over all injective mappings σ from $[k]$ to $[n]$. We observe that the expression (2) equals the permanent if we let $p = q = k/2 = \ell$ and define $f(X)$ as the sum of $a_{1\sigma(1)}a_{2\sigma(2)} \cdots a_{\ell\sigma(\ell)}$ over all injective mappings σ from $\{1, 2, \ldots, \ell\}$ to X and, similarly, $g(Y)$ as the sum of $a_{\ell+1\sigma(\ell+1)}a_{\ell+2\sigma(\ell+2)} \cdots a_{k\sigma(k)}$ over all injective mappings σ from $\{\ell + 1, \ell + 2, \ldots, k\}$ to Y. Since the values $f(X)$ and $g(Y)$ for all relevant X and Y can be computed by dynamic programming with $O\bigl(k\binom{n}{\downarrow k/2}\bigr)$ operations in S, our main result yields an upper bound of $O\bigl(2^{k/2}k\binom{n}{\downarrow k/2}\bigr)$ operations in S for computing the permanent.

Thus we improve significantly upon a Bellman–Held–Karp type dynamic programming algorithm that computes the permanent with $O\bigl(k\binom{n}{\downarrow k}\bigr)$ operations in S, the best previous upper bound we are aware of for noncommutative semirings [5]. It should be noted, however, that algorithms using $O\bigl(k\binom{n}{\downarrow k/2}\bigr)$ operations in S are already known for noncommutative rings [5], and that faster algorithms using $O\bigl(k(n - k + 1)2^k\bigr)$ operations in S are known for commutative semirings [5, 13].

3. Evaluation of disjoint sums

Overview. In this section, we prove Theorem 1 by giving an inductive construction for the evaluation of (n,p,q)-disjoint summation. That is, we reduce (n,p,q)-disjoint summation into one $(n-1,p,q)$-disjoint summation and two $(n-2,p-1,q-1)$-disjoint summations. The key idea is to "pack" two elements of the ground set $[n]$ (say, 1 and n) into a new element $*$ and apply $(n-1,p,q)$-disjoint summation. We then complete this to (n,p,q)-disjoint summation using the two $(n-1,p-2,q-2)$-disjoint summations.

Preliminaries. For intervals of natural numbers, we write $[k,n] = \{k, k+1, \ldots, n\}$. For a set S and $k \le |S|$, we write $\binom{S}{k} = \{T \subseteq S \colon |T| = k\}$ and $\binom{S}{\downarrow k} = \binom{S}{0} \cup \binom{S}{1} \cup \cdots \cup \binom{S}{k}$. With these notations, the (n,p,q)-disjoint summation takes an input function $f \colon \binom{[n]}{\downarrow p} \to S$ to the output function $e \colon \binom{[n]}{\downarrow q} \to S$ defined by (1).

Base cases. When $p = 0$ or $q = 0$, the (n,p,q)-disjoint summation simplifies to a form permitting trivial evaluation. If $p = 0$, then we have $e(Y) = f(\emptyset)$ for all $Y \in \binom{[n]}{\downarrow q}$, and no additions are required. If $q = 0$, then there remains a single sum

$$e(\emptyset) = \sum_{X \in \binom{[n]}{\downarrow p}} f(X)\,,$$

which can be evaluated with $\binom{n}{\downarrow p} - 1$ additions in S.

If $p = n$ or $q = n$, we use Yates's algorithm to evaluate (n,p,q)-disjoint summation with $2^{n-1}n$ additions in S. For completeness, we repeat Yates's construction here. Given the input function $f \colon 2^{[n]} \to S$, we compute intermediate functions $a_i \colon 2^{[n]} \to S$ for $i = 0, 1, 2, \ldots, n$, where $a_0 = f$ and for any $Z \subseteq [n]$ and $i = 1, 2, \ldots, n$ the value $a_i(Z)$ is obtained from recurrence

$$a_i(Z) = \begin{cases} a_{i-1}(Z \cup \{i\}) + a_{i-1}(Z) & \text{if } i \notin Z, \\ a_{i-1}(Z \setminus \{i\}) & \text{if } i \in Z. \end{cases}$$

The output function $e \colon 2^{[n]} \to S$ can be recovered as $e = a_n$.

Inductive step. We may assume that $n \ge 2$ and $1 \le p, q \le n-1$. Let $*$ be an element not in $[n]$, and define $f_* \colon \binom{[2,n-1]\cup\{*\}}{\downarrow p} \to S$ as

$$\begin{aligned} f_*\big(X\big) &= f\big(X\big) && \text{for } X \in \tbinom{[2,n-1]}{\downarrow p}, \\ f_*\big(X \cup \{*\}\big) &= f\big(X \cup \{1\}\big) + f\big(X \cup \{n\}\big) && \text{for } X \in \tbinom{[2,n-1]}{p-1}, \\ f_*\big(X \cup \{*\}\big) &= f\big(X \cup \{1\}\big) + f\big(X \cup \{n\}\big) + f\big(X \cup \{1,n\}\big) && \text{for } X \in \tbinom{[2,n-1]}{\downarrow (p-2)}. \end{aligned}$$

We apply $(n-1,p,q)$-disjoint summation to input f_* to obtain output $e_*\colon \binom{[2,n-1]\cup\{*\}}{\downarrow q} \to S$ with

$$\begin{aligned} e_*\big(Y\big) &= \sum_{X\in\binom{[n]}{\downarrow p}\colon X\cap Y=\emptyset} f\big(X\big) && \text{for } Y\in\binom{[2,n-1]}{\downarrow q}, \\ e_*\big(Y\cup\{*\}\big) &= \sum_{X\in\binom{[2,n-1]}{\downarrow p}\colon X\cap Y=\emptyset} f\big(X\big) && \text{for } Y\in\binom{[2,n-1]}{\downarrow (q-1)}. \end{aligned} \tag{3}$$

In particular, we have $e(Y) = e_*(Y)$ for $Y \in \binom{[2,n-1]}{\downarrow q}$ and $e(Y\cup\{1,n\}) = e_*(Y\cup\{*\})$ for $Y\in\binom{[2,n-1]}{\downarrow (q-2)}$.

To complete e_* to e, we compute two $(n-2,p-1,q-1)$-disjoint summations over ground set $[2,n-1]$. Let

$$\begin{aligned} f_1\colon \tbinom{[2,n-1]}{\downarrow (p-1)} \to S\,, &\qquad f_1\big(X\big) = f\big(X\cup\{1\}\big)\,, \\ f_n\colon \tbinom{[2,n-1]}{\downarrow (p-1)} \to S\,, &\qquad f_n\big(X\big) = f\big(X\cup\{n\}\big)\,. \end{aligned}$$

Now applying $(n-2,p-1,q-1)$-disjoint summations to inputs f_1 and f_n yields output functions

$$e_1\colon \tbinom{[2,n-1]}{\downarrow (q-1)} \to S\,, \qquad e_1\big(Y\big) = \sum_{X\in\binom{[2,n-1]}{\downarrow (p-1)}\colon X\cap Y=\emptyset} f\big(X\cup\{1\}\big)\,, \tag{4}$$

$$e_n\colon \tbinom{[2,n-1]}{\downarrow (q-1)} \to S\,, \qquad e_n\big(Y\big) = \sum_{X\in\binom{[2,n-1]}{\downarrow (p-1)}\colon X\cap Y=\emptyset} f\big(X\cup\{n\}\big)\,. \tag{5}$$

By (3), (4) and (5), we can recover the output function $e\colon \binom{[n]}{\downarrow q} \to S$ as

$$\begin{aligned} e\big(Y\big) &= e_*\big(Y\big) && \text{for } Y\in\binom{[2,n-1]}{\downarrow q}, \\ e\big(Y\cup\{1\}\big) &= e_*\big(Y\cup\{*\}\big) + e_n\big(Y\big) && \text{for } Y\in\binom{[2,n-1]}{\downarrow (q-1)}, \\ e\big(Y\cup\{n\}\big) &= e_*\big(Y\cup\{*\}\big) + e_1\big(Y\big) && \text{for } Y\in\binom{[2,n-1]}{\downarrow (q-1)}, \\ e\big(Y\cup\{1,n\}\big) &= e_*\big(Y\cup\{*\}\big) && \text{for } Y\in\binom{[2,n-1]}{\downarrow (q-2)}. \end{aligned}$$

Analysis. It remains to show that computing the (n,p,q)-disjoint summation using the construction given above is sufficient to obtain the bound of Theorem 1. For $p=0$ or $q=0$, we have $C_{n,0,q}=0$ and $C_{n,p,0}=\binom{n}{\downarrow p}-1$. For $p=n$ or $q=n$, we have by Yates's algorithm that $C_{n,n,q}, C_{n,p,n} \le 2^{n-1}n$.

Otherwise, we have $p\ge 1$, $q\ge 1$ and $n\ge 2$. Let $r=\min\{p,q\}$. Computing f_* from f takes $2\binom{n-2}{\downarrow (p-2)}+\binom{n-2}{p-1}$ additions in S and computing e from e_*, e_1 and e_n takes $2\binom{n-2}{\downarrow (q-1)}$ additions in S. Thus, we have

$$C_{n,p,q} \le C_{n-1,p,q} + 2\cdot C_{n-2,p-1,q-1} + 2\tbinom{n-2}{\downarrow (p-2)} + \tbinom{n-2}{p-1} + 2\tbinom{n-2}{\downarrow (q-1)}. \tag{6}$$

By induction, it holds that

$$C_{n-1,p,q} \leq \left[p \binom{n-1}{\downarrow p} + q \binom{n-1}{\downarrow q} \right] \cdot 2^r \tag{7}$$

and

$$\begin{aligned} 2 \cdot C_{n-2,p-1,q-1} &\leq \left[(p-1) \binom{n-2}{\downarrow (p-1)} + (q-1) \binom{n-2}{\downarrow (q-1)} \right] \cdot 2^r \\ &\leq \left[(p-1) \binom{n-1}{\downarrow (p-1)} + (q-1) \binom{n-1}{\downarrow (q-1)} \right] \cdot 2^r \\ &= \left[p \binom{n-1}{\downarrow (p-1)} + q \binom{n-1}{\downarrow (q-1)} \right] \cdot 2^r - \left[\binom{n-1}{\downarrow (p-1)} + \binom{n-1}{\downarrow (q-1)} \right] \cdot 2^r . \end{aligned} \tag{8}$$

Inserting (7) and (8) into (6) and noting that an iterative application of Pascal's rule implies $\binom{n}{\downarrow k} = \binom{n-1}{\downarrow (k-1)} + \binom{n-1}{\downarrow k}$, we obtain

$$\begin{aligned} C_{n,p,q} &\leq \left[p \left[\binom{n-1}{\downarrow p} + \binom{n-1}{\downarrow (p-1)} \right] + q \left[\binom{n-1}{\downarrow q} + \binom{n-1}{\downarrow (q-1)} \right] \right] \cdot 2^r \\ &= \left[p \binom{n}{\downarrow p} + q \binom{n}{\downarrow q} \right] \cdot 2^r . \end{aligned}$$

Acknowledgements

We thank Jukka Suomela for useful discussions. The research was supported in part by the Academy of Finland, Grants 252083, 256287 (P.K.), and 125637, 218153, 255675 (M.K.), by the Helsinki Doctoral Programme in Computer Science – Advanced Computing and Intelligent Systems (J.K.), and by RFBR, grants 11–01–00508, 11–01–00792, and DMS RAS "Algebraic and combinatorial methods of mathematical cybernetics and information systems of new generation" programme, project "Problems of optimal synthesis of control systems" (I.S.).

References

[1] Richard Bellman. Combinatorial processes and dynamic programming. In *Combinatorial Analysis*, volume 10 of *Proceedings of Symposia in Applied Mathematics*, pages 217–249. AMS, 1960.

[2] Richard Bellman. Dynamic programming treatment of the travelling salesman problem. *Journal of the ACM*, 9(1):61–63, 1962.

[3] Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Fourier meets Möbius: fast subset convolution. In *39th ACM Symposium on Theory of Computing (STOC 2007)*, pages 67–74. ACM, 2007.

[4] Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Counting paths and packings in halves. In *17th European Symposium on Algorithms (ESA 2009)*, pages 578–586. Springer, 2009.

[5] Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Evaluation of permanents in rings and semirings. *Information Processing Letters*, 110(20):867–870, 2010.

[6] Peter Bürgisser, Michael Clausen, and M. Amin Shokrollahi. *Algebraic Complexity Theory*. Springer, 1997.

[7] Alexander V. Chashkin. On the complexity of boolean matrices, graphs, and the boolean functions corresponding to them. *Diskretnaya matematika*, 6(2):42–73, 1994. (In Russian. English translation in *Discrete Mathematics and Applications* 4(3):229–257, 1994).

[8] Cassio P. de Campos and Qiang Ji. Efficient structure learning of Bayesian networks using constraints. *Journal of Machine Learning Research*, 12: 663–689, 2011.

[9] Michael Held and Richard M. Karp. A dynamic programming approach to sequencing problems. *Journal of the Society for Industrial and Applied Mathematics*, 10(1):196–210, 1962.

[10] Petteri Kaski, Mikko Koivisto, and Janne H. Korhonen. Fast monotone summation over disjoint sets. In *7th International Symposium on Parameterized and Exact Computation (IPEC 2012)*, pages 159–170. Springer, 2012.

[11] Robert Kennes. Computational aspects of the Möbius transformation of graphs. *IEEE Transactions on Systems, Man and Cybernetics*, 22(2): 201–223, 1992.

[12] Donald E. Knuth. *The Art of Computer Programming*, volume 2, Seminumerical Algorithms. Addison–Wesley, 3rd edition, 1998.

[13] Ioannis Koutis and Ryan Williams. Limits and applications of group algebras for parameterized problems. In *36th International Colloquium on Automata, Languages and Programming (ICALP 2009)*, pages 653–664. Springer, 2009.

[14] Leslie G. Valiant. Negation is powerless for boolean slice functions. *SIAM Journal on Computing*, 15:531–535, 1986.

[15] Virginia Vassilevska and Ryan Williams. Finding, minimizing, and counting weighted subgraphs. In *41th ACM Symposium on Theory of Computing (STOC 2009)*, pages 455–464. ACM, 2009.

[16] Virginia Vassilevska, Ryan Williams, and Raphael Yuster. Finding heaviest H-subgraphs in real weighted graphs, with applications. *ACM Transactions on Algorithms*, 6(3):Art. 44, 2010.

[17] Virginia Vassilevska Williams. Multiplying matrices faster than Coppersmith-Winograd. In *44th ACM Symposium on Theory of Computing (STOC 2012)*, pages 887–898. ACM, 2012.

[18] Frank Yates. *The Design and Analysis of Factorial Experiments*. Imperial Bureau of Soil Science, 1937.

Paper II

II

Petteri Kaski, Mikko Koivisto, and Janne H. Korhonen.

Fast monotone summation over disjoint sets.

doi:10.1007/978-3-642-33293-7_16

Fast Monotone Summation over Disjoint Sets*

Petteri Kaski[1], Mikko Koivisto[2], and Janne H. Korhonen[2]

[1] Helsinki Institute for Information Technology HIIT & Department of Information and Computer Science, Aalto University, Finland
[2] Helsinki Institute for Information Technology HIIT & Department of Computer Science, University of Helsinki, Finland

Abstract. We study the problem of computing an ensemble of multiple sums where the summands in each sum are indexed by subsets of size p of an n-element ground set. More precisely, the task is to compute, for each subset of size q of the ground set, the sum over the values of all subsets of size p that are *disjoint* from the subset of size q. We present an arithmetic circuit that, without subtraction, solves the problem using $O((n^p + n^q) \log n)$ arithmetic gates, all monotone; for constant p, q this is within the factor $\log n$ of the optimal. The circuit design is based on viewing the summation as a "set nucleation" task and using a tree-projection approach to implement the nucleation. Applications include improved algorithms for counting heaviest k-paths in a weighted graph, computing permanents of rectangular matrices, and dynamic feature selection in machine learning.

1 Introduction

Weak Algebrisation. Many hard combinatorial problems benefit from *algebrisation*, where the problem to be solved is cast in algebraic terms as the task of evaluating a particular expression or function over a suitably rich algebraic structure, such as a multivariate polynomial ring over a finite field. Recent advances in this direction include improved algorithms for the k-path [25], Hamiltonian path [4], k-coloring [9], Tutte polynomial [6], knapsack [21], and connectivity [14] problems. A key ingredient in all of these advances is the exploitation of an algebraic catalyst, such as the existence of additive inverses for inclusion–exclusion, or the existence of roots of unity for evaluation/interpolation, to obtain fast evaluation algorithms.

Such advances withstanding, it is a basic question whether the catalyst is *necessary* to obtain speedup. For example, fast algorithms for matrix multiplication [11,13] (and combinatorially related tasks such as finding a triangle in a graph [1,17]) rely on the assumption that the scalars have a ring structure, which prompts the question whether a weaker structure, such as a semiring without

* This research was supported in part by the Academy of Finland, Grants 252083 (P.K.), 256287 (P.K.), and 125637 (M.K.), and by the Helsinki Doctoral Programme in Computer Science - Advanced Computing and Intelligent Systems (J.K.).

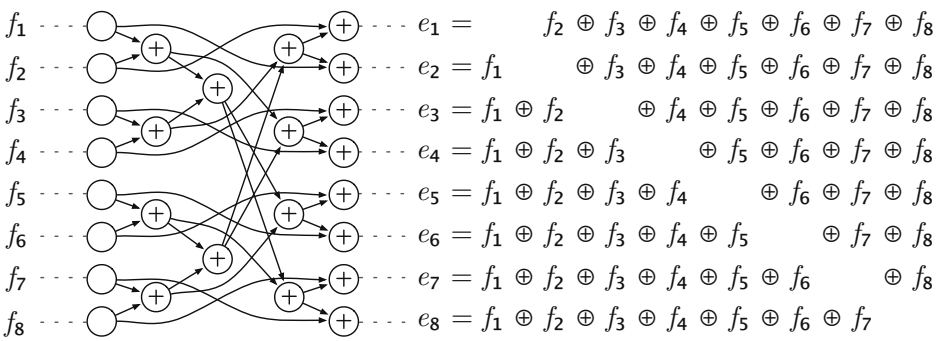
D.M. Thilikos and G.J. Woeginger (Eds.): IPEC 2012, LNCS 7535, pp. 159–170, 2012.

additive inverses, would still enable fast multiplication. The answer to this particular question is known to be negative [18], but for many of the recent advances such an analysis has not been carried out. In particular, many of the recent algebrisations have significant combinatorial structure, which gives hope for *positive* results even if algebraic catalysts are lacking. The objective of this paper is to present one such positive result by deploying *combinatorial* tools.

A Lemma of Valiant. Our present study stems from a technical lemma of Valiant [22] encountered in the study of circuit complexity over a monotone versus a universal basis. More specifically, starting from n variables $f_1, f_2, \ldots, f_n$, the objective is to use as few arithmetic operations as possible to compute the n sums of variables where the jth sum e_j includes all the other variables except the variable f_j, where $j = 1, 2, \ldots, n$.

If additive inverses are available, a solution using $O(n)$ arithmetic operations is immediate: first take the sum of all the n variables, and then for $j = 1, 2, \ldots, n$ compute e_j by subtracting the variable f_j.

Valiant [22] showed that $O(n)$ operations suffice also when additive inverses are *not* available; we display Valiant's elegant combinatorial solution for $n = 8$ below as an arithmetic circuit.



Generalising to Higher Dimensions. This paper generalises Valiant's lemma to higher dimensions using purely combinatorial tools. Accordingly, we assume that only very limited algebraic structure is available in the form of a commutative semigroup $(S, \oplus)$. That is, $\oplus$ satisfies the associative law $x \oplus (y \oplus z) = (x \oplus y) \oplus z$ and the commutative law $x \oplus y = y \oplus x$ for all $x, y, z \in S$, but nothing else is assumed.

By "higher dimensions" we refer to the input not consisting of n values ("variables" in the example above) in S, but rather $\binom{n}{p}$ values $f(X) \in S$ indexed by the p-subsets X of $[n] = \{1, 2, \ldots, n\}$. Accordingly, we also allow the output to have higher dimension. That is, given as input a function f from the p-subsets $[n]$ to the set S, the task is to output the function e defined for each q-subset Y of $[n]$ by

$$e(Y) = \bigoplus_{X : X \cap Y = \emptyset} f(X)\,, \tag{1}$$

where the sum is over all p-subsets X of $[n]$ satisfying the intersection constraint. Let us call this problem (p, q)-*disjoint summation*.

In analogy with Valiant's solution for the case $p = q = 1$ depicted above, an algorithm that solves the (p, q)-disjoint summation problem can now be viewed as a circuit consisting of two types of gates: *input gates* indexed by p-subsets X and *arithmetic gates* that perform the operation $\oplus$, with certain arithmetic gates designated as output gates indexed by q-subsets Y. We would like a circuit that has as few gates as possible. In particular, does there exist a circuit whose size for constant p, q is within a logarithmic factor of the lower bound $\Theta(n^p + n^q)$?

Main Result. In this paper we answer the question in the affirmative. Specifically, we show that a circuit of size $O\big((n^p + n^q) \log n\big)$ exists to compute e from f over an arbitrary commutative semigroup $(S, \oplus)$, and moreover, there is an algorithm that constructs the circuit in time $O\big((p^2 + q^2)(n^p + n^q) \log^3 n\big)$. These bounds hold uniformly for all p, q. That is, the coefficient hidden by O-notation does not depend on p and q.

From a technical perspective our main contribution is combinatorial and can be expressed as a solution to a specific *set nucleation* task. In such a task we start with a collection of "atomic compounds" (a collection of singleton sets), and the goal is to assemble a specified collection of "target compounds" (a collection of sets that are unions of the singletons). The assembly is to be executed by a straight-line program, where each operation in the program selects two *disjoint* sets in the collection and inserts their union into the collection. (Once a set is in the collection, it may be selected arbitrarily many times.) The assembly should be done in as few operations as possible.

Our main contribution can be viewed as a straight-line program of length $O\big((n^p + n^q) \log n\big)$ that assembles the collection $\{\{X : X \cap Y = \emptyset\} : Y\}$ starting from the collection $\{\{X\} : X\}$, where X ranges over the p-subsets of $[n]$ and Y ranges over the q-subsets of $[n]$. Valiant's lemma [22] in these terms provides an optimal solution of length $\Theta(n)$ for the specific case $p = q = 1$.

Applications. Many classical optimisation problems and counting problems can be algebrised over a commutative semigroup. A selection of applications will be reviewed in Sect. 3.

Related Work. "Nucleation" is implicit in the design of many fast algebraic algorithms, perhaps two of the most central are the fast Fourier transform of Cooley and Tukey [12] (as is witnessed by the butterfly circuit representation) and Yates's 1937 algorithm [26] for computing the product of a vector with the tensor product of n matrices of size 2×2. The latter can in fact be directly used to obtain a nucleation process for (p, q)-disjoint summation, even if an inefficient one. (For an exposition of Yates's method we recommend Knuth [19, §4.6.4]; take $m_i = 2$ and $g_i(s_i, t_i) = [s_i = 0 \text{ or } t_i = 0]$ for $i = 1, 2, \ldots, n$ to extract the following nucleation process implicit in the algorithm.) For all $Z \subseteq [n]$ and $i \in \{0, 1, \ldots, n\}$, let

$$a_i(Z) = \{X \subseteq [n] : X \cap [n - i] = Z \cap [n - i],\ X \cap Z \setminus [n - i] = \emptyset\}\,. \qquad (2)$$

Put otherwise, $a_i(Z)$ consists of X that agree with Z in the first $n - i$ elements of $[n]$ and are disjoint from Z in the last i elements of $[n]$. In particular, our objective

is to assemble the sets $a_n(Y) = \{X : X \cap Y = \emptyset\}$ for each $Y \subseteq [n]$ starting from the singletons $a_0(X) = \{X\}$ for each $X \subseteq [n]$. The nucleation process given by Yates' algorithm is, for all $i = 1, 2, \ldots, n$ and $Z \subseteq [n]$, to set

$$a_i(Z) = \begin{cases} a_{i-1}(Z \setminus \{n+1-i\}) & \text{if } n+1-i \in Z, \\ a_{i-1}(Z \cup \{n+1-i\}) \cup a_{i-1}(Z) & \text{if } n+1-i \notin Z. \end{cases} \tag{3}$$

This results in $2^{n-1}n$ disjoint unions. If we restrict to the case $|Y| \leq q$ and $|X| \leq p$, then it suffices to consider only Z with $|Z| \leq p+q$, which results in $O\big((p+q)\sum_{j=0}^{p+q}\binom{n}{j}\big)$ disjoint unions. Compared with our main result, this is not particularly efficient. In particular, our main result relies on "tree-projection" partitioning that enables a significant speedup over the "prefix-suffix" partitioning in (2) and (3).

We observe that "set nucleation" can also be viewed as a computational problem, where the output collection is given and the task is to decide whether there is a straight-line program of length at most ℓ that assembles the output using (disjoint) unions starting from singleton sets. This problem is known to be NP-complete even in the case where output sets have size 3 [15, Problem PO9]; moreover, the problem remains NP-complete if the unions are not required to be disjoint.

2 A Circuit for (p, q)-Disjoint Summation

Nucleation of p-Subsets with a Perfect Binary Tree. Looking at Valiant's circuit construction in the introduction, we observe that the left half of the circuit accumulates sums of variables (i.e., sums of 1-subsets of $[n]$) along what is a perfect binary tree. Our first objective is to develop a sufficient generalisation of this strategy to cover the setting where each summand is indexed by a p-subset of $[n]$ with $p \geq 1$.

Let us assume that $n = 2^b$ for a nonnegative integer b so that we can identify the elements of $[n]$ with binary strings of length b. We can view each binary string of length b as traversing a unique path starting from the root node of a perfect binary tree of height b and ending at a unique leaf node. Similarly, we may identify any node at level ℓ of the tree by a binary string of length ℓ, with $0 \leq \ell \leq b$. See Fig. 1(a) for an illustration. For $p = 1$ this correspondence suffices.

For $p > 1$, we are not studying individual binary strings of length b (that is, individual elements of $[n]$), but rather p-subsets of such strings. In particular, we can identify each p-subset of $[n]$ with a p-subset of leaf nodes in the binary tree. To nucleate such subsets it will be useful to be able to "project" sets upward in the tree. This motivates the following definitions.

Let us write $\{0,1\}^\ell$ for the set of all binary strings of length $0 \leq \ell \leq b$. For $\ell = 0$, we write ϵ for the empty string. For a subset $X \subseteq \{0,1\}^b$, we define the *projection of X to level ℓ* as

$$X|_\ell = \left\{x \in \{0,1\}^\ell \colon \exists y \in \{0,1\}^{b-\ell} \text{ such that } xy \in X\right\}. \tag{4}$$

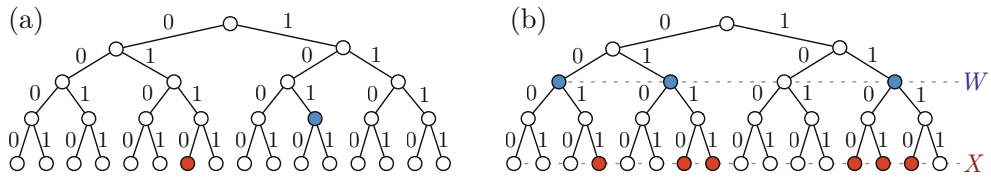


Fig. 1. Representing $\{0,1\}$-strings of length at most b as nodes in a perfect binary tree of height b. Here $b = 4$. (a) Each string traces a unique path down from the root node, with the empty string ϵ corresponding to the root node. The nodes at level $0 \leq \ell \leq b$ correspond to the strings of length ℓ. The red leaf node corresponds to 0110 and the blue node corresponds to 101. (b) A set of strings corresponds to a set of nodes in the tree. The set X is displayed in red, the set W in blue. The set W is the projection of the set X to level $\ell = 2$. Equivalently, $X|_\ell = W$.

That is, $X|_\ell$ is the set of length-ℓ prefixes of strings in X. Equivalently, in the binary tree we obtain $X|_\ell$ by lifting each element of X to its ancestor on level-ℓ in the tree. See Fig. 1(b) for an illustration. For the empty set we define $\emptyset|_\ell = \emptyset$.

Let us now study a set family $\mathcal{F} \subseteq 2^{\{0,1\}^b}$. The intuition here is that each member of $\mathcal{F}$ is a summand, and $\mathcal{F}$ represents the sum of its members. A circuit design must assemble (nucleate) $\mathcal{F}$ by taking disjoint unions of carefully selected subfamilies. This motivates the following definitions.

For a level $0 \leq \ell \leq b$ and a string $W \subseteq \{0,1\}^\ell$ let us define *the subfamily of $\mathcal{F}$ that projects to W* by

$$\mathcal{F}_W = \{X \in \mathcal{F} \colon X|_\ell = W\}\,. \tag{5}$$

That is, the family $\mathcal{F}_W$ consists of precisely those members $X \in \mathcal{F}$ that project to W. Again Fig. 1(b) provides an illustration: we select precisely those X whose projection is W.

The following technical observations are now immediate. For each $0 \leq \ell \leq b$, if $\emptyset \in \mathcal{F}$, then we have

$$\mathcal{F}_\emptyset = \{\emptyset\}\,. \tag{6}$$

Similarly, for $\ell = 0$ we have

$$\mathcal{F}_{\{\epsilon\}} = \mathcal{F} \setminus \{\emptyset\}\,. \tag{7}$$

For $\ell = b$ we have for every $W \in \mathcal{F}$ that

$$\mathcal{F}_W = \{W\}\,. \tag{8}$$

Now let us restrict our study to the situation where the family $\mathcal{F} \subseteq 2^{\{0,1\}^b}$ contains only sets of size at most p. In particular, this is the case in our applications. For a set U and an integer p, let us write $\binom{U}{p}$ for the family of all subsets of U of size p, and $\binom{U}{\downarrow p}$ for the family of all subsets of U with size at most p. Accordingly, for integers $0 \leq k \leq n$, let us use the shorthand $\binom{n}{\downarrow k} = \sum_{i=0}^{k} \binom{n}{i}$.

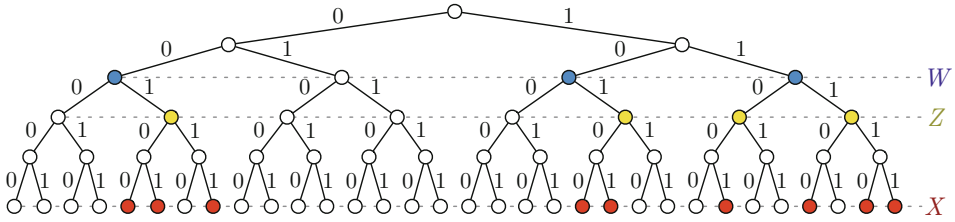


Fig. 2. Illustrating the proof of Lemma 1. Here $b = 5$. The set X (indicated with red nodes) projects to level $\ell = 2$ to the set W (indicated with blue nodes) and to level $\ell + 1 = 3$ to the set Z (indicated with yellow nodes). Furtermore, the projection of Z to level ℓ is W. Thus, each $X \in \mathcal{F}$ is included to $\mathcal{F}_W$ exactly from $\mathcal{F}_Z$ in Lemma 1.

The following lemma enables us to recursively nucleate any family $\mathcal{F} \subseteq \binom{\{0,1\}^b}{\downarrow p}$. In particular, we can nucleate the family $\mathcal{F}_W$ with W in level ℓ using the families $\mathcal{F}_Z$ with Z in level $\ell + 1$. Applied recursively, we obtain $\mathcal{F}$ by proceeding from the bottom up, that is, $\ell = b, b-1, \ldots, 1, 0$. The intuition underlying the lemma is illustrated in Fig. 2. We refer to the full version of this paper for the proof.

Lemma 1. *For all* $0 \leq \ell \leq b-1$, $\mathcal{F} \subseteq \binom{\{0,1\}^b}{\downarrow p}$, *and* $W \in \binom{\{0,1\}^\ell}{\downarrow p}$, *we have that the family* $\mathcal{F}_W$ *is a disjoint union* $\mathcal{F}_W = \bigcup \left\{ \mathcal{F}_Z \colon Z \in \binom{\{0,1\}^{\ell+1}}{\downarrow p}_W \right\}$.

A Generalisation: (p, q)-Intersection Summation. It will be convenient to study a minor generalisation of (p, q)-disjoint summation. Namely, instead of insisting on disjointness, we allow nonempty intersections to occur with "active" (or "avoided") q-subsets A, but require that elements in the intersection of each p-subset and each A are "individualized." That is, our input is not given by associating a value $f(X) \in S$ to each set $X \in \binom{[n]}{\downarrow p}$, but is instead given by associating a value $g(I, X) \in S$ to each pair (I, X) with $I \subseteq X \in \binom{[n]}{\downarrow p}$, where I indicates the elements of X that are "individualized." In particular, we may insist (by appending to S a formal identity element if such an element does not already exist in S) that $g(I, X)$ vanishes unless I is empty. This reduces (p, q)-disjoint summation to the following problem:

Problem 1. (**(p, q)-intersection summation**) Given as input a function g that maps each pair (I, X) with $I \subseteq X \in \binom{[n]}{\downarrow p}$ and $|I| \leq q$ to an element $g(I, X) \in S$, output the function $h \colon \binom{[n]}{\downarrow q} \to S$ defined for all $A \in \binom{[n]}{\downarrow q}$ by

$$h(A) = \bigoplus_{X \in \binom{[n]}{\downarrow p}} g(A \cap X, X)\,. \tag{9}$$

The Circuit Construction. We proceed to derive a recursion for the function h using Lemma 1 to carry out nucleation of p-subsets. The recursion proceeds

from the bottom up, that is, $\ell = b, b-1, \ldots, 1, 0$ in the binary tree representation. (Recall that we identify the elements of $[n]$ with the elements of $\{0,1\}^b$, where n is a power of 2 with $n = 2^b$.) The intermediate functions h_ℓ computed by the recursion are "projections" of (9) using (5). In more precise terms, for $\ell = b, b-1, \ldots, 1, 0$, the function $h_\ell \colon \binom{\{0,1\}^b}{\downarrow q} \times \binom{\{0,1\}^\ell}{\downarrow p} \to S$ is defined for all $W \in \binom{\{0,1\}^\ell}{\downarrow p}$ and $A \in \binom{\{0,1\}^b}{\downarrow q}$ by

$$h_\ell(A, W) = \bigoplus_{X \in \binom{\{0,1\}^b}{\downarrow p}_W} g(A \cap X, X)\,. \tag{10}$$

Let us now observe that we can indeed recover the function h from the case $\ell = 0$. Indeed, for the empty string ϵ, the empty set $\emptyset$ and every $A \in \binom{\{0,1\}^b}{\downarrow q}$ we have by (6) and (7) that

$$h(A) = h_0(A, \{\epsilon\}) \oplus h_0(A, \emptyset)\,. \tag{11}$$

It remains to derive the recursion that gives us h_0. Here we require one more technical observation, which enables us to narrow down the intermediate values $h_\ell(A, W)$ that need to be computed to obtain h_0. In particular, we may discard the part of the active set A that extends outside the "span" of W. This observation is the crux in deriving a succinct circuit design.

For $0 \le \ell \le b$ and $w \in \{0,1\}^\ell$, we define the *span* of w by

$$\langle w \rangle = \left\{ x \in \{0,1\}^b \colon \exists z \in \{0,1\}^{b-\ell} \text{ such that } wz = x \right\}.$$

In the binary tree, $\langle w \rangle$ consists of the leaf nodes in the subtree rooted at w. Let us extend this notation to subsets $W \subseteq \{0,1\}^\ell$ by $\langle W \rangle = \bigcup_{w \in W} \langle w \rangle$. The following lemma shows that it is sufficient to evaluate $h_\ell(A, W)$ only for $W \in \binom{\{0,1\}^\ell}{\downarrow p}$ and $A \in \binom{\{0,1\}^b}{\downarrow q}$ such that $A \subseteq \langle W \rangle$. We omit the proof; please refer to the full version of this paper for details.

Lemma 2. *For all $0 \le \ell \le b$, $W \in \binom{\{0,1\}^\ell}{\downarrow p}$, and $A \in \binom{\{0,1\}^b}{\downarrow q}$, we have*

$$h_\ell(A, W) = h_\ell(A \cap \langle W \rangle, W)\,. \tag{12}$$

We are now ready to present the recursion for $\ell = b, b-1, \ldots, 1, 0$. The base case $\ell = b$ is obtained directly based on the values of g, because we have by (8) for all $W \in \binom{\{0,1\}^b}{\downarrow p}$ and $A \in \binom{\{0,1\}^b}{\downarrow q}$ with $A \subseteq W$ that

$$h_b(A, W) = g(A, W)\,. \tag{13}$$

The following lemma gives the recursive step from $\ell + 1$ to ℓ by combining Lemma 1 and Lemma 2. Again, we defer the details of the proof to the full version of this paper.

Lemma 3. *For* $0 \leq \ell \leq b-1$, $W \in \binom{\{0,1\}^\ell}{\downarrow p}$, *and* $A \in \binom{\{0,1\}^b}{\downarrow q}$ *with* $A \subseteq \langle W \rangle$, *we have*

$$h_\ell(A, W) = \bigoplus_{Z \in \binom{\{0,1\}^{\ell+1}}{\downarrow p}_W} h_{\ell+1}(A \cap \langle Z \rangle, Z)\,. \tag{14}$$

The recursion given by (13), (14), and (12) now defines an arithmetic circuit that solves (p, q)-intersection summation.

Size of the circuit. By (13), the number of input gates in the circuit is equal to the number of pairs (I, X) with $I \subseteq X \in \binom{\{0,1\}^b}{\downarrow p}$ and $|X| \leq q$, which is

$$\sum_{i=0}^{p} \sum_{j=0}^{q} \binom{2^b}{i} \binom{i}{j}\,. \tag{15}$$

To derive an expression for the number of $\oplus$-gates, we count for each $0 \leq \ell \leq b-1$ the number of pairs (A, W) with $W \in \binom{\{0,1\}^\ell}{\downarrow p}$, $A \in \binom{\{0,1\}^b}{\downarrow q}$, and $A \subseteq \langle W \rangle$, and for each such pair (A, W) we count the number of $\oplus$-gates in the subcircuit that computes the value $h_\ell(A, W)$ from the values of $h_{\ell+1}$ using (14).

First, we observe that for each $W \in \binom{\{0,1\}^\ell}{\downarrow p}$ we have $|\langle W \rangle| = 2^{b-\ell}\,|W|$. Thus, the number of pairs (A, W) with $W \in \binom{\{0,1\}^\ell}{\downarrow p}$, $A \in \binom{\{0,1\}^b}{\downarrow q}$, and $A \subseteq \langle W \rangle$ is

$$\sum_{i=0}^{p} \sum_{j=0}^{q} \binom{2^\ell}{i} \binom{i2^{b-\ell}}{j}\,. \tag{16}$$

For each such pair (A, W), the number of $\oplus$-gates for (14) is $\left|\binom{\{0,1\}^{\ell+1}}{\downarrow p}_W\right| - 1$.

Lemma 4. *For all* $0 \leq \ell \leq b-1$, $W \in \binom{\{0,1\}^\ell}{\downarrow p}$, *and* $|W| = i$, *we have*

$$\left|\binom{\{0,1\}^{\ell+1}}{\downarrow p}_W\right| = \sum_{k=0}^{p-i} \binom{i}{k} 2^{i-k}\,. \tag{17}$$

Proof. A set $Z \in \binom{\{0,1\}^{\ell+1}}{\downarrow p}_W$ can contain either one or both of the strings $w0$ and $w1$ for each $w \in W$. The set Z may contain both elements for at most $p-i$ elements $w \in W$ because otherwise $|Z| > p$. Finally, for each $0 \leq k \leq p-i$, there are $\binom{i}{k} 2^{i-k}$ ways to select a set $Z \in \binom{\{0,1\}^{\ell+1}}{\downarrow p}_W$ such that Z contains $w0$ and $w1$ for exactly k elements $w \in W$.

Finally, for each $A \in \binom{\{0,1\}^b}{\downarrow q}$ we require an $\oplus$-gate that is also designated as an output gate to implement (11). The number of these gates is

$$\sum_{j=0}^{q} \binom{2^b}{j}\,. \tag{18}$$

The total number of $\oplus$-gates in the circuit is obtained by combining (15), (16), (17), and (18). The number of $\oplus$-gates is thus

$$\begin{aligned}
&\sum_{i=0}^{p}\sum_{j=0}^{q}\binom{2^b}{i}\binom{i}{j}+\sum_{\ell=0}^{b-1}\sum_{i=0}^{p}\sum_{j=0}^{q}\binom{2^\ell}{i}\binom{i2^{b-\ell}}{j}\left(\sum_{k=0}^{p-i}\binom{i}{k}2^{i-k}-1\right)+\sum_{j=0}^{q}\binom{2^b}{j}\\
&\le\sum_{\ell=0}^{b}\sum_{i=0}^{p}\sum_{j=0}^{q}\binom{2^\ell}{i}\binom{i2^{b-\ell}}{j}3^i\le\sum_{\ell=0}^{b}\sum_{i=0}^{p}\sum_{j=0}^{q}\frac{(2^\ell)^i}{i!}\frac{i^j(2^{b-\ell})^j}{j!}3^i\\
&\le\sum_{\ell=0}^{b}\sum_{i=0}^{p}\sum_{j=0}^{q}\frac{(2^\ell)^{\max(p,q)}}{i!}\frac{i^j(2^{m-\ell})^{\max(p,q)}}{j!}3^i\\
&=n^{\max(p,q)}(1+\log_2 n)\sum_{i=0}^{p}\sum_{j=0}^{q}\frac{i^j3^i}{i!j!}\,.
\end{aligned}$$

The remaining double sum is bounded from above by a constant, and thus the circuit defined by (13), (14), and (12) has size $O((n^p+n^q)\log n)$, where the constant hidden by the O-notation does not depend on p and q.

The circuit can be constructed in time $O\big((p^2+q^2)(n^p+n^q)\log^3 n\big)$. We omit the details.

3 Concluding Remarks and Applications

We have generalised Valiant's [22] observation that negation is powerless for computing simultaneously the n different disjunctions of all but one of the given n variables: now we know that, in our terminology, subtraction is powerless for (p,q)-disjoint summation for any constant p and q. (Valiant proved this for $p=q=1$.) Interestingly, requiring p and q be constants turns out to be essential, namely, when subtraction is available, an inclusion–exclusion technique is known [5] to yield a circuit of size $O\big(p\binom{n}{\downarrow p}+q\binom{n}{\downarrow q}\big)$, which, in terms of p and q, is exponentially smaller than our bound $O\big((n^p+n^q)\log n\big)$. This gap highlights the difference of the algorithmic ideas behind the two results. Whether the gap can be improved to polynomial in p and q is an open question.

While we have dealed with the abstract notions of "monotone sums" or semigroup sums, in applications they most often materialise as maximisation or minimisation, as described in the next paragraphs. Also, in applications local terms are usually combined not only by one (monotone) operation but two different operations, such as "min" and "+". To facilitate the treatment of such applications, we extend the semigroup to a semiring $(S,\oplus,\odot)$ by introducing a product operation "$\odot$". Now the task is to evaluate

$$\bigoplus_{X,Y:X\cap Y=\emptyset} f(X)\odot g(Y)\,, \tag{19}$$

where X and Y run through all p-subsets and q-subsets of $[n]$, respectively, and f and g are given mappings to S. We immediately observe that the expression

(19) is equal to $\bigoplus_Y e(Y) \odot g(Y)$, where the sum is over all q-subsets of $[n]$ and e is as in (1). Thus, by our main result, it can be evaluated using a circuit with $O((n^p + n^q) \log n)$ gates.

Application to k-paths. We apply the semiring formulation to the problem of counting the maximum-weight k-edge paths from vertex s to vertex t in a given edge-weighted graph with real weights, where we assume that we are only allowed to add and compare real numbers and these operations take constant time (cf. [24]). By straightforward Bellman–Held–Karp type dynamic programming [2,3,16] (or, even by brute force) we can solve the problem in $\binom{n}{\downarrow k} n^{O(1)}$ time. However, our main result gives an algorithm that runs in $n^{k/2+O(1)}$ time by solving the problem in halves: Guess a middle vertex v and define $f_1(X)$ as the number of maximum-weight $k/2$-edge paths from s to v in the graph induced by the vertex set $X \cup \{v\}$; similarly define $g_1(X)$ for the $k/2$-edge paths from v to t. Furthermore, define $f_2(X)$ and $g_2(X)$ as the respective maximum weights and put $f(X) = (f_1(X), f_2(X))$ and $g(X) = (g_1(X), g_2(X))$. These values can be computed for all vertex subsets X of size $k/2$ in $\binom{n}{k/2} n^{O(1)}$ time. It remains to define the semiring operations in such a way that the expression (19) equals the desired number of k-edge paths; one can verify that the following definitions work correctly: $(c, w) \odot (c', w') = (c \cdot c', w + w')$ and

$$(c, w) \oplus (c', w') = \begin{cases} (c, w) & \text{if } w > w', \\ (c', w') & \text{if } w < w', \\ (c + c', w) & \text{if } w = w'. \end{cases}$$

Thus, the techniques of the present paper enable solving the problem essentially as fast as the fastest known algorithms for the special case of counting *all* the k-paths, for which quite different techniques relying on subtraction yield $\binom{n}{k/2} n^{O(1)}$ time bound [7]. On the other, for the more general problem of counting weighted subgraphs Vassilevska and Williams [23] give an algorithm whose running time, when applied to k-paths, is $O(n^{\omega k/3} + n^{2k/3+c})$, where $\omega < 2.3727$ is the exponent of matrix multiplication and c is a constant; this of course would remain worse than our bound even if $\omega = 2$.

Application to Matrix Permanent. Consider the problem of computing the permanent of a $k \times n$ matrix (a_{ij}) over a *noncommutative semiring*, with $k \leq n$ and even for simplicity, given by $\sum_\sigma a_{1\sigma(1)} a_{2\sigma(2)} \cdots a_{k\sigma(k)}$, where the sum is over all injective mappings σ from $[k]$ to $[n]$. We observe that the expression (19) equals the permanent if we let $p = q = k/2 = \ell$ and define $f(X)$ as the sum of $a_{1\sigma(1)} a_{2\sigma(2)} \cdots a_{\ell\sigma(\ell)}$ over all injective mappings σ from $\{1, 2, \ldots, \ell\}$ to X and, similarly, $g(Y)$ as the sum of $a_{\ell+1\sigma(\ell+1)} a_{\ell+2\sigma(\ell+2)} \cdots a_{k\sigma(k)}$ over all injective mappings σ from $\{\ell+1, \ell+2, \ldots, k\}$ to Y. Since the values $f(X)$ and $g(Y)$ for all relevant X and Y can be computed by dynamic programming in $\binom{n}{k/2} n^{O(1)}$ time, our main result yields the time bound $n^{k/2+O(1)}$ for computing the permanent.

Thus we improve significantly upon a Bellman–Held–Karp type dynamic programming algorithm that computes the permanent in $\binom{n}{\downarrow k} n^{O(1)}$ time, the best

previous upper bound we are aware of for noncommutative semirings [8]. It should be noted, however, that essentally as fast algorithms are already known for *noncommutative rings* [8], and that faster, $2^k n^{O(1)}$ time, algorithms are known for *commutative semirings* [8,20].

Application to Feature Selection. The extensively studied feature selection problem in machine learning asks for a subset X of a given set of available features A so as to maximise some objective function $f(X)$. Often the size of X can be bounded from above by some constant k, and sometimes the selection task needs to be solved repeatedly with the set of available features A changing dynamically across, say, the set $[n]$ of all features. Such constraints take place in a recent work [10] on Bayesian network structure learning by branch and bound: the algorithm proceeds by forcing some features, I, to be included in X and some other, E, to be excluded from X. Thus the key computational step becomes that of maximising $f(X)$ subject to $I \subseteq X \subseteq [n] \setminus E$ and $|X| \leq k$, which is repeated for varying I and E. We observe that instead of computing the maximum every time from scratch, it pays off precompute a solution to (p, q)-disjoint summation for all $0 \leq p, q \leq k$, since this takes about the same time as a single step for $I = \emptyset$ and any fixed E. Indeed, in the scenario where the branch and bound search proceeds to exclude each and every subset of k features in turn, but no larger subsets, such precomputation decreases the running time bound quite dramatically, from $O(n^{2k})$ to $O(n^k)$; typically, n ranges from tens to some hundreds and k from 2 to 7. Admitted, in practice, one can expect the search procedure match the said scenario only partially, and so the savings will be more modest yet significant.

Acknowledgment. We thank Jukka Suomela for useful discussions.

References

1. Alon, N., Yuster, R., Zwick, U.: Finding and counting given length cycles. Algorithmica 17(3), 209–223 (1997)
2. Bellman, R.: Combinatorial processes and dynamic programming. In: Combinatorial Analysis. Proceedings of Symposia in Applied Mathematics, vol. 10, pp. 217–249. ACM (1960)
3. Bellman, R.: Dynamic programming treatment of the travelling salesman problem. J. ACM 9(1), 61–63 (1962)
4. Björklund, A.: Determinant sums for undirected Hamiltonicity. In: 51st Annual IEEE Symposium on Foundations of Computer Science (FOCS 2010), pp. 173–182. IEEE Computer Society, Los Alamitos (2010)
5. Björklund, A., Husfeldt, T., Kaski, P., Koivisto, M.: Fourier meets möbius: fast subset convolution (manuscript submitted for publication)
6. Björklund, A., Husfeldt, T., Kaski, P., Koivisto, M.: Computing the Tutte polynomial in vertex-exponential time. In: 49th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2008), pp. 677–686. IEEE Computer Society, Los Alamitos (2008)
7. Björklund, A., Husfeldt, T., Kaski, P., Koivisto, M.: Counting Paths and Packings in Halves. In: Fiat, A., Sanders, P. (eds.) ESA 2009. LNCS, vol. 5757, pp. 578–586. Springer, Heidelberg (2009)

8. Björklund, A., Husfeldt, T., Kaski, P., Koivisto, M.: Evaluation of permanents in rings and semirings. Information Processing Letters 110(20), 867–870 (2010)
9. Björklund, A., Husfeldt, T., Koivisto, M.: Set partitioning via inclusion-exclusion. SIAM Journal on Computing 39(2), 546–563 (2009)
10. de Campos, C.P., Ji, Q.: Efficient structure learning of bayesian networks using constraints. Journal of Machine Learning Research 12, 663–689 (2011)
11. Cohn, H., Kleinberg, R., Szegedy, B., Umans, C.: Group-theoretic algorithms for matrix multiplication. In: 46th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2005), pp. 379–388. IEEE Computer Society, Los Alamitos (2005)
12. Cooley, J.W., Tukey, J.W.: An algorithm for the machine calculation of complex Fourier series. Mathematics of Computation 19, 297–301 (1965)
13. Coppersmith, D., Winograd, S.: Matrix multiplication via arithmetic progressions. Journal of Symbolic Computation 9(3), 251–280 (1990)
14. Cygan, M., Nederlof, J., Pilipczuk, M., Pilipczuk, M., van Rooij, J.M.M., Wojtaszczyk, J.O.: Solving connectivity problems parameterized by treewidth in single exponential time (2011) (manuscript)
15. Garey, M.R., Johnson, D.S.: Computers and Intractability, A Guide to the Theory of NP-Completeness. W.H. Freeman and Company (1979)
16. Held, M., Karp, R.M.: A dynamic programming approach to sequencing problems. Journal of the Society for Industrial and Applied Mathematics 10(1), 196–210 (1962)
17. Itai, A., Rodeh, M.: Finding a minimum circuit in a graph. SIAM Journal on Computing 7(4), 413–423 (1978)
18. Kerr, L.R.: The effect of algebraic structure on the computational complexity of matrix multiplications. Ph.D. thesis, Cornell University (1970)
19. Knuth, D.E.: The Art of Computer Programming, 3rd edn. Seminumerical Algorithms, vol. 2. Addison–Wesley, Upper Saddle River (1998)
20. Koutis, I., Williams, R.: Limits and Applications of Group Algebras for Parameterized Problems. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009, Part I. LNCS, vol. 5555, pp. 653–664. Springer, Heidelberg (2009)
21. Lokshtanov, D., Nederlof, J.: Saving space by algebraization. In: 2010 ACM International Symposium on Theory of Computing (STOC 2010), pp. 321–330. ACM, New York (2010)
22. Valiant, L.G.: Negation is powerless for boolean slice functions. SIAM Journal on Computing 15, 531–535 (1986)
23. Vassilevska, V., Williams, R.: Finding, minimizing, and counting weighted subgraphs. In: 2009 ACM International Symposium on Theory of Computing (STOC 2009), pp. 455–464. ACM, New York (2009)
24. Vassilevska, V., Williams, R., Yuster, R.: Finding heaviest H-subgraphs in real weighted graphs, with applications. ACM Transactions on Algorithms 6(3), Art. 44, 23 (2010)
25. Williams, R.: Finding paths of length k in $O^*(2^k)$ time. Information Processing Letters 109(6), 315–318 (2009)
26. Yates, F.: The Design and Analysis of Factorial Experiments. Imperial Bureau of Soil Science, Harpenden, England (1937)

Paper III

Janne H. Korhonen and Pekka Parviainen.

Learning bounded tree-width Bayesian networks.

III

Exact Learning of Bounded Tree-width Bayesian Networks

Janne H. Korhonen
University of Helsinki
Dept. of Computer Science and HIIT
Helsinki, Finland

Pekka Parviainen
Royal Institute of Technology
CSC and Scilifelab
Stockholm, Sweden

Abstract

Inference in Bayesian networks is known to be NP-hard, but if the network has bounded tree-width, then inference becomes tractable. Not surprisingly, learning networks that closely match the given data and have a bounded tree-width has recently attracted some attention. In this paper we aim to lay groundwork for future research on the topic by studying the exact complexity of this problem. We give the first non-trivial exact algorithm for the NP-hard problem of finding an optimal Bayesian network of tree-width at most w, with running time $3^n n^{w+O(1)}$, and provide an implementation of this algorithm. Additionally, we propose a variant of Bayesian network learning with "super-structures", and show that finding a Bayesian network consistent with a given super-structure is fixed-parameter tractable in the tree-width of the super-structure.

1 INTRODUCTION

1.1 Bayesian network learning

Bayesian networks are used widely to represent joint probability distributions. Typically, the first step in using a Bayesian network to model some problem is *learning* the network from the input data. That is, we have to learn a directed acyclic graph (DAG) and the parameters associated with each variable, so that the model describes the original data "well". Learning the parameters given a structure is an easy task, so in the recent years research has mostly focused on learning the structure. One of the main approaches to structure learning is so-called score-based methods (Cooper and Herskovits, 1992; Heckerman et al., 1995), where the idea is to assign each possible structure a score based on how well it fits the data and try to find a structure that maximises the score.

In this paper, we study the Bayesian structure learning as a combinatorial problem using an abstract score-based framework. In this framework, we are given a node set N of size n and for each node $v \in N$ and each parent set $S \subseteq N \setminus \{v\}$ a local score $f_v(S)$. The goal is to find a DAG A that maximises the sum

$$f(A) = \sum_{v \in N} f_v(A_v)\,,$$

where A_v is the parent set of v, *i.e.*, the set of nodes u such that there is an arc from u to v in A. This problem is NP-hard (Chickering, 1996; Chickering et al., 2004), the best known exact algorithm being a Bellman–Held–Karp style dynamic programming that runs in time $2^n n^{O(1)}$ (Silander and Myllymäki, 2006).

1.2 Learning with bounded tree-width

Once the network has been learned, we want to use it to compute conditional probabilities of some sets of variables given some other sets of variables. This *inference problem* in Bayesian networks is also NP-hard (Cooper, 1990). However, if the network (or more precisely its structure) has low *tree-width*, exact inference is tractable even for large networks. Thus, learning models of bounded tree-width enables us to limit the time required for inference. Specifically, we have a trade-off between the fit of the network and the speed of inference, since if the "true" network has high tree-width, bounding the tree-width can lead to under-fitting.

More formally, given local scores f_v as before and a constant w, we want to find a DAG A that maximises the score $f(A)$ among the DAGs of tree-width at most w. Here the tree-width of a DAG is defined as the tree-width of its moralised graph (Elidan and Gould, 2008); the *moralised graph* of a DAG A is an undirected

Appearing in Proceedings of the 16^{th} International Conference on Artificial Intelligence and Statistics (AISTATS) 2013, Scottsdale, AZ, USA. Volume 31 of JMLR: W&CP 31.

graph that includes an edge $\{u, v\} \in E$ for every arc $uv \in A$ and an edge $\{u, w\} \in E$ for every pair of arcs $uv \in A$ and $wv \in A$. Defined this way, the tree-width of a network matches its inference complexity[1].

While there have been some studies on learning undirected models with bounded tree-width using approximation algorithms (Karger and Srebro, 2001; Srebro, 2001), heuristics (Bach and Jordan, 2002), and PAC-learning (Chechetka and Guestrin, 2008), the corresponding problem for Bayesian networks remains poorly understood. The only result of this vein we are aware of is a heuristic algorithm for learning bounded tree-width Bayesian network structures by Elidan and Gould (2008). Our main motivation for the work presented in this paper is to fill this gap and lay groundwork for future investigations of the topic. Specifically, we aim to establish basic theoretical results for learning bounded tree-width Bayesian network structures, especially in the sense of *exact* algorithmics.

Unfortunately, learning Bayesian network structures remains difficult when the tree-width is bounded. While learning an optimal *tree*, i.e., a Bayesian network with tree-width 1, can be done in polynomial time (Chow and Liu, 1968), a straightforward reduction from a corresponding result for Markov networks shows that finding an optimal Bayesian network of tree-width at most w is NP-hard for any *fixed* $w \geq 2$; see Section 2.3.

1.3 Learning in exponential time

Since learning bounded tree-width Bayesian networks is NP-hard, the natural question from the perspective of exact algorithmics is to study exponential-time algorithms for the problem. As our main result, we obtain a single-exponential time algorithm for bounded tree-width Bayesian structure learning.

Theorem 1. *Given a node set N of size n, an integer w and scoring functions f_v for each node $v \in N$, we can find a DAG A with tree-width at most w maximising score $f(A) = \sum_{v \in N} f_v(A_v)$ in time $3^n n^{w+O(1)}$ and space $2^n n^{w+O(1)}$.*

The proof of Theorem 1 is given in Section 5.

Somewhat disappointingly we are not able to match the $2^n n^{O(1)}$ algorithm for unrestricted Bayesian network structure learning. Indeed, it seems to us that the added restriction of the bounded tree-width makes the problem more challenging.

On the practical side, we have implemented the algorithm of Theorem 1, and it works well for small n and w. We also experimented with using this implementation to find small bounded tree-width networks for real-world data; see Section 6.

Although the exponentiality hinders the application of the algorithm for all but a small number of nodes, we argue that having even an exponential exact algorithm for the problem is essential for further investigations of the topic. Principally, it provides a baseline against which approximation algorithms and heuristics can be tested, and it may also prove to be useful as a component of such algorithms. Furthermore, being able to generate examples of optimal bounded tree-width networks enables explorative studies of their properties.

We also note that in the recent years there has been a lot of interest in exponential time algorithms for learning the structure of a Bayesian network (Ott and Miyano, 2003; Singh and Moore, 2005; Silander and Myllymäki, 2006) and related tasks, like computing posterior probabilities of structural features (Koivisto and Sood, 2004; Koivisto, 2006; Tian and He, 2009; Kang et al., 2010; Parviainen and Koivisto, 2011). Most of the algorithms run in $2^n n^{O(1)}$ time but some of them have running time $3^n n^{O(1)}$ which matches our algorithm. In line with our experiments, other algorithms with running time $3^n n^{O(1)}$ been implemented and tested successfully with networks of up to 20 nodes (Tian and He, 2009; Kang et al., 2010; Parviainen and Koivisto, 2011).

1.4 Learning with super-structures

The dynamic programming algorithm of Theorem 1 implicitly contains a subroutine that, given an undirected graph G, finds an optimal DAG whose moralised graph is a subgraph of G. Indeed, this problem is fixed-parameter tractable with regard to the tree-width of the graph G, as formalised in the following theorem, whose proof we give in Section 4.

Theorem 2. *For any fixed w, given an n-vertex graph $G = (N, E)$ of tree-width at most w and scoring functions f_v for each node $v \in N$, we can find a DAG A whose moralised graph is a subgraph of G maximising the score in time and space $O(n)$.*

Specifically, the running time of our algorithm is $O\big((w+1)! \cdot w \cdot 3^w \cdot n\big)$ if we are given a suitable tree-decomposition of G. As it is usual with algorithms based on tree-decompositions, the bottle-neck is the construction of a tree-decomposition from G; see Section 2.1.

This observation is related to the super-structure approach for learning Bayesian networks, presented by Perrier et al. (2008), where we are given an undirected graph G, called the super-structure, and the goal is to

[1] To avoid confusion, we point out that this definition differs from the definition of tree-width for directed graphs given by Johnson et al. (2001).

find the highest-scoring DAG such that there is, for each arc in the DAG, a corresponding undirected edge in G, i.e. the *skeleton* of the DAG is a subgraph of the super-structure. Recently Ordyniak and Szeider (2010) have shown that in this setting, if both the tree-width and the maximum degree of the super-structure are bounded by constants, an optimal DAG can be found in linear time. However, the tree-width of the super-structure alone does not bound the running time of the algorithm or the tree-width of the resulting DAG, and Ordyniak and Szeider in fact show that learning optimal Bayesian network with given super-structure is $W[1]$-hard when the complexity parameter is the tree-width of the super-structure. Intuitively, the reason for this is that the tree-width of a DAG is defined to be the tree-width of its moralised graph, and moralising can introduce edges that are not present in the super-structure.

2 PRELIMINARIES

2.1 Tree-width

For an undirected graph $G = (V, E)$, we use the convention that E is a set of two-element subsets of V. We write $\{u, v\}$ for an edge between nodes u and v. We also denote $n = |V|$.

A *tree-decomposition* of an undirected graph $G = (V, E)$ is a pair $(\mathcal{X}, T)$, where $\mathcal{X} = \{X_1, X_2, \ldots, X_m\}$ is a collection of subsets of V and T is a tree on $\{1, 2, \ldots, m\}$, such that

1. $\bigcup_{i=1}^{m} X_i = V$,
2. for all edges $\{u, v\} \in E$ there exist i with $u \in X_i$ and $v \in X_i$, and
3. for all i, j and k, if j is on the (unique) path from i to k in T, then $X_i \cap X_k \subseteq X_j$.

The *width* of a tree-decomposition $(\mathcal{X}, T)$ is defined as $\max_i |X_i| - 1$. The *tree-width* of an undirected graph G is the minimum width over all possible tree-decompositions of G. In a sense, the tree-width of a graph describes how close the graph is to a tree; graphs of tree-width 1 coincide with trees. For a fixed w and graph G with tree-width w, a tree-decomposition of width w can be found in time $O(n)$ (Bodlaender, 1996).

A *nice tree-decomposition* of a graph $G = (V, E)$ is a tree-decomposition $(\mathcal{X}, T)$ along with a fixed root node r for T such that each node $i \in \{1, 2, \ldots, m\}$ is either

1. a *leaf* with no children and $|X_i| = 1$,
2. a *forget* node that has one child j and $X_i = X_j \setminus \{v\}$ for some $v \in X_j$,
3. a *introduce* node that has one child j and $X_i = X_j \cup \{v\}$ for some $v \notin X_j$, or
4. a *join* node that has two children j and k and $X_i = X_j = X_k$.

For fixed w, if we are given a tree-decomposition of G with width w, we can construct a nice tree-decomposition of width w and a linear number of nodes in time $O(n)$ (Kloks, 1994). Thus, if we are given a graph of tree-width w, we can also obtain a nice tree-decomposition of width w and a linear number of nodes in time $O(n)$, since some tree-decomposition of width w can be found in linear time, as noted above.

We will in particular need the following basic property of the tree-decompositions.

Lemma 3 (Separation Property)**.** *Let $G = (V, E)$ be a graph with tree-decomposition $(\mathcal{X}, T)$, and let i, j and k be nodes in T such that j is on the path from i to k. Then there is no edge $\{u, v\} \in E$ with $v \in X_i \setminus X_j$ and $u \in X_k \setminus X_j$.*

Finally, we give a well-known alternate characterisation of tree-width. The family of *k-trees* is defined inductively as follows.

1. A $(k+1)$-clique is a k-tree.
2. If $G = (V, E)$ is a k-tree and $C \subseteq V$ is a k-clique, then graph obtained by adding a new vertex v and an edge uv for each $u \in C$ is a k-tree.

The k-trees are maximal graphs with tree-width k, that is, a graph has tree-width k if and only if it is a subgraph of some k-tree; see e.g. van Leeuwen (1990).

2.2 Bayesian Networks

Aside from the definitions given in Sections 1.2 and 1.4, we will use the following conventions when discussing Bayesian networks and the structure learning problem.

A directed graph is a pair (N, A), where N is the node set and $A \subseteq N \times N$ is the arc set. We write uv for arc $(u, v) \in A$. When there is no ambiguity about the node set, we identify a directed graph by its arc set. Throughout this paper, we denote $n = |N|$.

A node u is said to be a *parent* of node v if the arc set contains an arc from u to v, that is, $uv \in A$. If u is a parent of v, then v is a *child* of u. We denote the set of the parents of v in A by A_v.

As a consequence of the definition of the tree-width of of a DAG (see Section 1.2), we have that if the tree-width of a DAG is w, then the in-degree must be bounded by w, as $\{v\} \cup A_v$ is a clique in the moralised

graph M, and a graph with k-clique has tree-width at least $k-1$. The reverse does not hold, and a graph with maximum in-degree Δ can have tree-width larger than Δ.

In the structure learning problem, we are given the local scores $f_v(S)$ for each node $v \in N$ and each parent set $S \subseteq N \setminus \{v\}$ as input. The output is a DAG that has maximal score. As noted above, parent sets S of size more than w are not eligible when we want the DAG to have tree-width at most w. Thus, we assume that the input consist only of the scores for parent sets of size at most w and has size $O\left(n\binom{n}{w}\right)$. For structure learning with super-structures, we may further assume that we are given scores only for parent sets that are compatible with the super-structure, in which case the input has size $O(n2^w)$. We will not, however, consider the representation of the input in more detail, and in the rest of the paper we will assume that we can access scores $f_v(S)$ in constant time. This does not have significant effect on the analysis.

Finally, we define functions $\hat{f}_v$ by $\hat{f}_v(S) = \max_{T \subseteq S} f_v(T)$. That is, $\hat{f}_v(S)$ is the highest local score when the parents of node v are chosen from S. For any set X, the values $\hat{f}_v(S)$ for *all* sets $S \subseteq X$ can be computed from the values of f_v in $O\big(|X|2^{|X|}\big)$ time and $O\big(2^{|X|}\big)$ space using dynamic programming (Ott and Miyano, 2003).

2.3 Hardness

Srebro (2000) has shown that the problem of finding a subgraph of an input graph G with tree-width at most w and maximum number of edges is NP-hard for any fixed $w \geq 2$. The NP-hardness of learning bounded tree-width Bayesian network structures follows by a straightforward reduction.

Theorem 4. *Finding an optimal Bayesian network structure with tree-width at most w under a given scoring function is NP-hard for any fixed $w \geq 2$.*

Proof. Let $G = (V, E)$ be an undirected graph. Let $N = V \cup E$, and define a score function on N by setting $f_e(\{v, u\}) = 1$ for each edge $e = \{v, u\} \in E$, and let $f_v(S) = 0$ for any other node $v \in N$ and potential parent set $S \subseteq N \setminus \{v\}$. This transformation can be computed in polynomial time.

Now we note that there is a subgraph (V, F) of G with $|F| = m$ and tree-width at most w if and only if there is a DAG D on N with $f(D) = m$ and tree-width at most w; furthermore, if we are given one, we can compute the other in polynomial time. Since finding the maximum bounded tree-width subgraph is known to be NP-hard, the claim follows. □

3 DECOMPOSING DAGS

In this section, we establish results that will be used to prove the correctness of the algorithms we give later on. The intuitive idea is that if (N, A) is a DAG of low tree-width, then there is a small set $X \subseteq N$ whose removal will split A into two or more connected components. We can exploit this property by finding optimal DAGs that can act as these separate components, and then "glue" these DAGs together at X. We now proceed to formalise these ideas.

Let N be a set of nodes and let $X \subseteq N$ with $|X| = k$. For a permutation $\sigma = \sigma_1\sigma_2 \ldots \sigma_k$ of X and a set $S \subseteq X$, we say that a DAG A is a (σ, S)-*DAG on* N if the node set of A is N, it holds that A is *compatible* with σ, that is, $A \cup \{\sigma_p\sigma_{p+1} \colon p = 1, 2, \ldots, k-1\}$ is acyclic, and for each $v \in X \setminus S$ we have that $A_v = \emptyset$. For a (σ, S)-DAG A on N, we define the S-*score* of A as $f_S(A) = \sum_{v \in S \cup (N \setminus X)} f_v(A_v)$. That is, the nodes in X that are required to have empty parent sets do not contribute to the score.

In the following, we assume that N is some node set, and X, N_1 and N_2 are subsets of N such that $N_1 \cup N_2 = N$ and $N_1 \cap N_2 = X$. Furthermore, we assume that σ is a permutation of X and $S \subseteq X$.

Lemma 5. *Let $Z \subseteq S$. If A is a (σ, Z)-DAG on N_1 and B is a $(\sigma, S \setminus Z)$-DAG on N_2, then $A \cup B$ is a (σ, S)-DAG on N. Furthermore, we have*

$$f_S(A \cup B) = f_Z(A) + f_{S \setminus Z}(B)\,.$$

Proof. The claim follows almost directly from the definitions; the only non-trivial step to verify is that $A \cup B$ is in fact acyclic. To see that $A \cup B$ is acyclic, assume that there is a directed cycle C in $A \cup B$. Since both A and B are acyclic, there must be a node σ_i on cycle C. But since both A and C are compatible with σ, each maximal segment of C that consists only of edges in A or only of edges in B goes from a node σ_j to a node σ_ℓ for $j < \ell$, and thus the cycle cannot return to σ_i. □

For a (σ, S)-DAG A on N, we say that *a decomposition of A over N_1 and N_2* is a pair (B, C), where B is a (σ, Z)-DAG on N_1 and C is a $(\sigma, S \setminus Z)$-DAG on N_2 such that $A = B \cup C$ and $Z \subseteq S$. Note that if A has such a decomposition (B, C), then by Lemma 5 we have $f_S(A) = f_Z(B) + f_{S \setminus Z}(C)$.

Lemma 6. *Suppose that A is an (σ, S)-DAG on N and suppose there are no arcs in A between $N_1 \setminus X$ and $N_2 \setminus X$, and no $v \in N$, $u \in N_1 \setminus X$ and $w \in N_2 \setminus X$ such that $uv \in A$ and $wv \in A$. Then there is a decomposition of A over N_1 and N_2.*

Proof. Let $Z = \{v \in S \colon A_v \subseteq N_1\}$. Then the DAGs $B = \{uv \colon v \in Z \cup (N_1 \setminus X)\}$ and $C =$

$\{uv \colon v \in (S \setminus Z) \cup (N_2 \setminus X)\}$ clearly have the desired properties. □

Lemma 7. *Suppose that $\mathcal{A}$ is a family of (σ, S)-DAGs on N and that for each $Z \subseteq S$, we have that $\mathcal{B}_Z$ is a family of (σ, Z)-DAGs on N_1 and $\mathcal{C}_Z$ a family of (σ, Z)-DAGs on N_2. If each $A \in \mathcal{A}$ has a decomposition (B, C) over N_1 and N_2 such that $B \in \mathcal{B}_Z$ and $C \in \mathcal{C}_{S \setminus Z}$ for some $Z \subseteq S$, and for each $Z \subseteq S$ and DAGs $B \in \mathcal{B}_Z$ and $C \in \mathcal{C}_{S \setminus Z}$ it holds that $B \cup C \in \mathcal{A}$, then*

$$\max_{A \in \mathcal{A}} f_S(A) = \max_{Z \subseteq S} \Big(\max_{B \in \mathcal{B}_Z} f_Z(B) + \max_{C \in \mathcal{C}_{S \setminus Z}} f_{S \setminus Z}(C) \Big) .$$

Proof. Fix $A \in \mathcal{A}$. Since A decomposes into $B \in \mathcal{B}_Z$ and $C \in \mathcal{C}_{S \setminus Z}$ for some $Z \subseteq S$, we have

$$\begin{aligned} f_S(A) &= f_Z(B) + f_{S \setminus Z}(C) \\ &\leq \max_{B \in \mathcal{B}_Z} f_Z(B) + \max_{C \in \mathcal{C}_{S \setminus Z}} f_{S \setminus Z}(C) \\ &\leq \max_{Z \subseteq S} \Big(\max_{B \in \mathcal{B}_Z} f_Z(B) + \max_{C \in \mathcal{C}_{S \setminus Z}} f_{S \setminus Z}(C) \Big) . \end{aligned}$$

On the other hand, since $B \cup C \in \mathcal{A}$ for all $B \in \mathcal{B}_Z$ and $C \in C_{S \setminus Z}$, there is a DAG in $\mathcal{A}$ with S-score

$$\max_{Z \subseteq S} \Big(\max_{B \in \mathcal{B}_Z} f_Z(B) + \max_{C \in \mathcal{C}_{S \setminus Z}} f_{S \setminus Z}(C) \Big) . \qquad \square$$

4 LEARNING WITH SUPER-STRUCTURES

We prove Theorem 2 first. The proof of this theorem will act as a preliminary for the proof of Theorem 1 in the next section.

To prove Theorem 2, we use dynamic programming on the tree-decomposition of the underlying super-structure. We will assume that parent sets that are not compatible with the super-structure graph G have score of $-\infty$ and will thus not be picked by the algorithm. That is, if for $v \in V$ we have that $S \subseteq V \setminus \{v\}$ contains a node u such that $\{u, v\} \notin E$, or nodes u and s such that $\{u, s\} \notin E$, then $f_v(S) = -\infty$.

Let $G = (V, E)$ be the super-structure graph and let $(\mathcal{X}, T)$ be a nice tree-decomposition of G with root r and $\mathcal{X} = \{X_1, X_2, \ldots, X_m\}$. For $1 \leq i \leq m$, we denote by V_i the union of bags below X_i in the tree.

Intuitively, our algorithm will proceed by computing for each node i in T an optimal (σ, S)-DAG on V_i for each permutation σ of X_i and $S \subseteq X_i$. We will show that these can be computed by starting from the leaves of T and proceeding upwards in the tree. Finally, in the root r, we will have the optimal (σ, X_i)-DAG on V for each permutation of X_i; taking the one with the best score gives us the desired optimal Bayesian network.

To formalise the intuition given above, let $i \in \{1, 2, \ldots, m\}$ and let $k = |X_i|$. For a permutation $\sigma = \sigma_1 \sigma_2 \ldots \sigma_k$ of X_i and $S \subseteq X_i$, we define

$$g_i(\sigma, S) = \max_A f_S(A) , \tag{1}$$

where A ranges over (σ, S)-DAGs on V_i such that the moralised graph of A is a subgraph of $G[V_i]$. It follows immediately from this definition that the best DAG on V has score $\max_\sigma g_r(\sigma, X_r)$, where σ ranges over the permutations of the root bag X_r. Furthermore, we note that it suffices to show how these scores can be computed, as the optimal DAG can then be recovered using standard techniques; see e.g., Silander and Myllymäki (2006).

The values $g_i(\sigma, X_i)$ can be computed using dynamic programming on the tree-decomposition, starting from the leaf nodes and going up in the tree. There are four cases to consider, depending on the type of node X_i.

Leaf: $X_i = \{v\}$ for $v \in N$. Then we have $g_i(v, \emptyset) = 0$ and $g_i(v, \{v\}) = f_v(\emptyset)$.

Forget: Node i has a child j and $X_i = X_j \setminus \{v\}$ for $v \in X_j$. Now $V_i = V_j$, and directly by definition we have that

$$g_i(\sigma, S) = \max_{\eta\tau = \sigma} g_j(\eta v \tau, S \cup \{v\}) \tag{2}$$

for all permutations σ of X_i and $S \subseteq X_i$. Computing (2) directly for all σ and S takes $O(k 2^k k!)$ time.

Introduce: Node i has a child j and $X_i = X_j \cup \{v\}$ for $v \notin V_j$. First, we compute values $\hat{f}_u(S)$ for $u \in X_i$ and $S \subseteq X_i \setminus \{u\}$, which takes $O(k 2^k)$ time as noted in Section 2.2. Now suppose that $\sigma = \eta v \tau$ is a permutation of X_i and $S \subseteq X_i$. Denote by $P_{\sigma,u}$ the set of elements of X_i that appear before u in σ. Then we have that

$$g_i(\sigma, S) = \max_{Z \subseteq S \setminus \{v\}} \Big(g_j(\eta\tau, Z) + \sum_{u \in S \setminus Z} \hat{f}_u(P_{\sigma,u}) \Big) . \tag{3}$$

To verify that (3) is correct, consider any (σ, S)-DAG A on V_i. Since by Lemma 3 there are no edges $\{u, v\} \in E$ for $u \in V_i \setminus X_i$, Lemma 6 implies that A interpreted as a $(\eta\tau, S \setminus \{v\})$-DAG has a decomposition (B, C) over $V_i \setminus \{v\}$ and X_i, where B is a $(\eta\tau, Z \setminus \{v\})$-DAG on $V_i \setminus \{v\}$ and C is a $(\sigma, S \setminus Z)$-DAG on X_i for some $Z \subseteq S$. Furthermore, the moralised graphs of both B and C are subgraphs of G. Finally, we note that the maximum score for a (σ, Z)-DAG on X_i is $\sum_{u \in Z} \hat{f}_u(P_{\sigma,u})$. The correctness of (3) now follows from Lemma 7.

Evaluating (3) for all σ and S can be done in time $O(k 3^k k!)$.

Join: Node i has children j and ℓ, and $X_i = X_j = X_\ell$. By Lemma 3, there are no edges between $V_j \setminus X_i$ and

$V_\ell \setminus X_i$ in G. Thus any (σ, S)-DAG A on V_i has a decomposition (B, C) over $V_j \setminus X_i$ and $V_\ell \setminus X_i$. Since moralised graphs of both B and C are subgraphs of G, Lemma 7 implies that

$$g_i(\sigma, S) = \max_{Z \subseteq S} \big(g_j(\sigma, Z) + g_\ell(\sigma, S \setminus Z) \big) . \qquad (4)$$

Evaluating (4) for all σ and S takes $O(3^k k!)$ time.

Summing the running times over all nodes in T, we obtain the following.

Theorem 8. *Given a graph G, a nice tree-decomposition $(\mathcal{X}, T)$ of G, and scoring functions f_v for each node v, we can find a DAG A whose moralised graph is a subgraph of G maximising the score in time $O\big((w+1)! \cdot w \cdot 3^w \cdot n\big)$, where w is the tree-width of G.*

As noted in Section 2.1, a nice tree-decomposition of the super-structure graph G can be obtained from G in $O(n)$ time. Thus, we obtain Theorem 2 as a corollary.

5 EXACT LEARNING WITH BOUNDED TREE-WIDTH

We will now proceed to prove Theorem 1 by giving a dynamic programming algorithm for the problem. This algorithm is based on the same ideas as the super-structure algorithm in Section 4, but here we perform dynamic programming over all possible tree-decompositions of width w. In the following, let w be a fixed tree-width bound.

As noted in Section 2.1, each graph of tree-width w is a subgraph of a w-tree. It follows that each graph $G = (V, E)$ of tree-width w has a rooted tree-decomposition $(\mathcal{X}, T)$ such that each bag X_i has size $w+1$, and for adjacent i and j we have that $|X_i \cap X_j| = w$. By applying an obvious transformation to $(\mathcal{X}, T)$, we have that G also has a rooted tree-decomposition $(\mathcal{Y}, Q)$ such that each bag has size $w+1$ and each node i in Q is either

1. a *leaf* with no children,
2. a *swap* node that has one child j such that $Y_i = (Y_j \setminus \{u\}) \cup \{v\}$ for some $u \in Y_j$ and $v \notin Y_j$, or
3. a *join* node that has two children j and ℓ such that $Y_i = Y_j = Y_\ell$.

Furthermore, by the construction we can assume that for a join node i with children j and ℓ, both V_j and V_ℓ contain vertices not in X_i. We will call a tree-decomposition satisfying these conditions *fat*. Thus, each graph G of tree-width w has a fat tree-decomposition of width w.

Let now N be a node set and let $X \subseteq N$. For a permutation σ of X and $S \subseteq X$, we say that a (σ, S)-DAG A on N is *rooted* if A has a fat tree-decomposition with root r and $X_r = X$. Furthermore, we say that A is *join-rooted* or *swap-rooted* if there is a fat tree-decomposition where the root node is of the corresponding type.

Now for $X \subseteq N$ with $|X| = w+1$, a permutation σ of X, and sets $S \subseteq X$ and $M \supseteq X$, we want to compute

$$g(\sigma, S, M) = \max_A f_S(A) ,$$

where A ranges over rooted (σ, S)-DAGs on M with tree-width at most w. Computing these values is sufficient for finding an optimal DAG of tree-width w, as the optimal DAG is rooted at some $X \subseteq N$ with $|X| = w+1$, thus has score $\max_{X,\sigma} g(\sigma, X, N)$, where X ranges over $(w+1)$-subsets of N and σ ranges over permutations of X.

We will now show that these values can be computed using dynamic programming, starting from sets $M \subseteq N$ with $|M| = w+1$. For any set M with $|M| = w+1$ and a permutation σ of M, we note that a (σ, S)-DAG A on M has a fat tree-decomposition whose root r is a leaf node with $X_r = M$. Thus, any (σ, S)-DAG on M is rooted, and we have that

$$g(\sigma, S, M) = \sum_{u \in S} \hat{f}_u(P_{\sigma,u}) ,$$

as σ completely specifies the order of nodes in any (σ, S)-DAG on M.

On the other hand, if $M \subseteq N$ with $|M| > w+1$, then the optimal rooted (σ, S)-DAG on M can be either join-rooted or swap-rooted. Therefore, we compute values

$$J(\sigma, S, M) = \max_B f_S(B) ,$$

where B ranges over join-rooted (σ, S)-DAGs on M with tree-width at most w, and

$$K(\sigma, S, M) = \max_C f_S(C) ,$$

where C ranges over swap-rooted (σ, S)-DAGs on M with tree-width at most w. Then we have that

$$g(\sigma, S, M) = \max\big\{ K(\sigma, S, M), J(\sigma, S, M) \big\} .$$

The one special case is the sets M with $|M| = w+2$, as then there cannot be a join-rooted (σ, S)-DAG on M. Thus, for M with $|M| = w+2$, we have $g(\sigma, S, M) = K(\sigma, S, M)$.

Join. First, we show how values $J(\sigma, S, M)$ can be computed. In the following, let M_1 and M_2 be sets such that $M_1 \cup M_2 = M$ and $M_1 \cap M_2 = X$. Furthermore, assume that $M_1 \neq X$ and $M_2 \neq X$.

Lemma 9. *If A is a rooted (σ, Z)-DAG on M_1 and B is a rooted $(\sigma, S \setminus Z)$-DAG on M_2, then $A \cup B$ is a join-rooted (σ, S)-DAG on M. Moreover, if A and B have tree-width at most w, so does $A \cup B$.*

Proof. The claim follows from Lemma 5 and from the observation that we can obtain the desired tree-decomposition for $A \cup B$ by adding the tree-decompositions of A and B as the children of a new root node r with $X_r = X$. □

Lemma 10. *If A is a join-rooted (σ, S)-DAG on M, then there are sets M_1 and M_2 such that $M_1 \cup M_2 = M$, $M_1 \cap M_2 = X$, $M_1 \neq X$, $M_2 \neq X$ and A has a decomposition (B, C) over M_1 and M_2 such that both B and C are rooted at X. Moreover, if A has tree-width at most w, so does B and C.*

Proof. Let $(\mathcal{X}, T)$ be a tree-decomposition of A with root r such that $X_r = X$ and r is a join node with children i and j. Let $M_1 = V_i$ and $M_2 = V_j$. Now by Lemma 3 and Lemma 6, A has decomposition (B, C) over M_1 and M_2. Noticing that the subtree of $(\mathcal{X}, T)$ rooted at i is a tree-decomposition of B and the subtree of $(\mathcal{X}, T)$ rooted at j is a tree-decomposition of C completes the proof, as both of these have width w and root bag X. □

For fixed M_1 and M_2, Lemma 9 implies that we can apply Lemma 7 similarly as in the join case of the super-structure algorithm, obtaining that the S-score of the best join-rooted (σ, S)-DAG that decomposes over M_1 and M_2 is

$$h(M_1, M_2) = \max_{Z \subseteq S} \big(g(\sigma, Z, M_1) + g(\sigma, S \setminus Z, M_2) \big).$$

As the optimal join-rooted (σ, S)-DAG on M decomposes over some M_1 and M_2 by Lemma 10, we have that

$$J(\sigma, S, M) = \max_{\substack{M_1 \cap M_2 = X \\ M_1 \cup M_2 = M \\ M_1, M_2 \neq X}} h(M_1, M_2) \,. \tag{5}$$

Evaluating (5) directly for fixed σ, S and M can be done in time

$$O\big(n \cdot 2^{|M \setminus X|} \cdot 2^{|S|}\big) = O\big(n \cdot 2^{|M| + |S| - (w+1)}\big) \,.$$

Swap. We now show how values $K(\sigma, S, M)$ can be computed. The following lemmas are analogous to Lemmas 9 and 10, and we omit their proofs.

Lemma 11. *Let $Y \subseteq M$ with $|Y| = w$, and $u \in M \setminus Y$ and $v \notin M$. Furthermore, let $\sigma = \eta v \tau$ be a permutation of $Y \cup \{v\}$ and $\gamma = \zeta u \rho$ be a permutation of $Y \cup \{u\}$ such that $\eta\tau = \zeta\rho$. If A is a (σ, S_1)-DAG on $Y \cup \{v\}$ and B is a rooted (γ, S_2)-DAG on M, then $A \cup B$ is a swap-rooted $(\sigma, S_1 \cup S_2)$-DAG on $M \cup \{v\}$. If B has tree-width at most w, then so does $A \cup B$.*

Lemma 12. *Let A be a swap-rooted (σ, S)-DAG on M. Then there are nodes $v \in X$ and $u \in M \setminus X$ such that, when we let $\sigma = \eta v \tau$ and $Y = (X \setminus \{v\}) \cup \{u\}$, there is a permutation $\gamma = \zeta u \rho$ of Y with $\zeta\rho = \eta\tau$, a (σ, S_1)-DAG B on X and a rooted (γ, S_2)-DAG C on $M \setminus \{u\}$ such that $A = B \cup C$. Furthermore, the tree-width of C is at most the tree-width of A.*

For $Y \subseteq M$ with $|Y| = w$ and a permutation γ of Y, we first compute an auxiliary function F defined by

$$F(\gamma, Z, M) = \max_A f_Z(A) \,,$$

where A ranges over rooted (σ, S)-DAGs on M such that σ is a permutation of a set $X \subseteq M$, for some $v \in M \setminus Y$ we have $X = Y \cup \{v\}$ and $S = Z \cup \{v\}$, and $\sigma = \eta v \tau$ with $\eta\tau = \gamma$. It follows directly from the definition that we can evaluate F as

$$F(\gamma, Z, M) = \max_{v \in M \setminus Y} \max_{\eta\tau = \gamma} g\big(\eta v \tau, Z \cup \{v\}, M\big) \,. \tag{6}$$

For a fixed $v \in X$, applying Lemmas 11 and 7 in a similar fashion as in the introduce case of the super-structure algorithm, we have that the maximum score of a swap-rooted (σ, S)-DAG on M with tree-width at most w such that the new node in the root bag is v is

$$\kappa(v) = \max_{Z \subseteq S \setminus \{v\}} \Big(F(\sigma, Z, M \setminus \{v\}) + \sum_{u \in S \setminus Z} \hat{f}_u(P_{\sigma,u}) \Big) \,.$$

It then follows from Lemma 12 that the maximum score can be obtained by optimising over v, that is, we have

$$K(\sigma, S, M) = \max_{v \in X} \kappa(v) \,. \tag{7}$$

Finally, we note that evaluating (6) for fixed γ, Z and M can be done in time $O\big(w^2\big)$, and when the required values of F have been evaluated beforehand, (7) can be evaluated in time $O\big(2^{|S|} w\big)$.

The total number of tuples (σ, S, M) for all $X \subseteq N$ is $(w+1)!\binom{n}{w+1} 2^n$. By summing the running times over all these tuples and estimating $\binom{n}{w+1} \leq n^{w+1}/(w+1)!$, we have that the total running time of our exact algorithm is $3^n n^{w + O(1)}$. Furthermore, we note that we need to store all values of g during the dynamic programming, meaning that the total space requirement of the algorithm is $2^n n^{w+O(1)}$. Thus, we have proven Theorem 1.

Remark 13. It is possible to recover a tree-decomposition for width w for the optimal DAG without extra computational cost in an obvious way.

Remark 14. By omitting the join step from the algorithm described in this section we can obtain a $2^n n^{w+O(1)}$ time algorithm for finding networks of bounded *path-width*.

6 EXPERIMENTS

To complement our theoretical results, we constructed a proof-of-concept implementation of the dynamic programming algorithm of Theorem 1 and tested it on real-world data sets[2]. In this section, we will discuss the performance of our implementation and provide examples of bounded tree-width networks on real-world data sets.

The implementation was made in Python, using cython compiler[3] to compile the most computationally demanding parts of the code to C, and the experiments were run under Linux on blade servers with 2.53-GHz processors and 32 GB of memory. We tested our implementation on two datasets, ADULT (15 variables, 32,561 samples) and HOUSING (14 variables, 506 samples), downloaded from the UCI machine learning repository (Frank and Asuncion, 2010). We discretised all variables into binary variables, and as the local scores, we used BDeu scores with equivalent sample size 1.

Finding optimal networks of tree-width at most 2 with our algorithm took 13,086 and 3,220 seconds for ADULT and HOUSING respectively; the reported times are user times measured using `time` and they include the dynamic programming itself and the recovery of the optimal network structure, but exclude the time needed for computing the local scores. We also benchmarked the implementation on various variable subsets of the aforementioned datasets with $w = 1$, $w = 2$, and $w = 3$, and the results were in line with theoretical bounds of Theorem 1. For $n = 14$ and $w = 3$, our implementation ran into problems with memory requirements, but this is mostly caused by an inefficient choice of data structure for storing dynamic programming values indexed by triples (σ, S, M), and these limits should be circumventable by a good choice of data structures and some careful algorithm engineering.

An example of optimal bounded tree-width networks for HOUSING found is shown in Figure 1. The network with unbounded tree-width is quite complex, with tree-width at least 6, so the networks with tree-width 1 and 2 are rough approximations. Indeed, the optimal network has score -3080, while the scores for bounded tree-width networks are -3295 for $w = 2$ and -3479 for $w = 1$. The optimal network with tree-width 2 has 23 arcs, meaning that is relatively dense, as a tree-width 2 network on 14 nodes can have at most 25 arcs. The most connected node is NOX, with 9 neighbours, in contrast to the optimal unbounded network, where NOX has only 5 neighbours. This hints that the more complex structure may allow representing dependencies indirectly. Overall, however, we do not feel that these examples suggest any hitherto unknown features of bounded tree-width networks as a model class. A more thorough study is warranted in the future.

[2]The implementation is available at `http://www.cs.helsinki.fi/u/jazkorho/aistats-2013/`.

[3]`http://www.cython.org`

Acknowledgements

We thank Mikko Koivisto for useful comments. The research was supported in part by the Helsinki Doctoral Programme in Computer Science - Advanced Computing and Intelligent Systems (J.K.) and by the Finnish Doctoral Programme in Computational Sciences (P.P.).

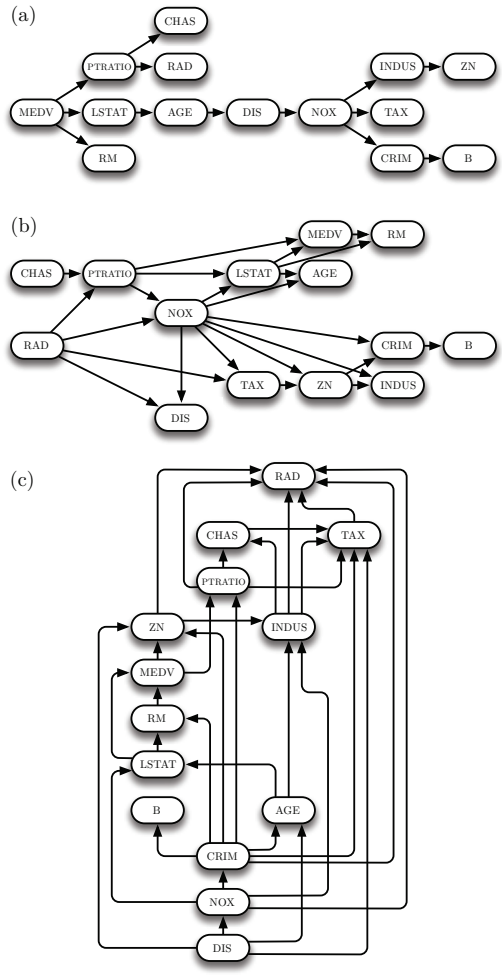


Figure 1: An optimal network for HOUSING for tree-width bound (a) $w = 1$, (b) $w = 2$, and (c) unbounded tree-width.

References

F. R. Bach and M. I. Jordan. Thin junction trees. In *Advances in Neural Information Processing Systems 14 (NIPS)*. MIT Press, 2002.

H. L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal of Computing*, 25:1305–1317, 1996.

A. Chechetka and C. Guestrin. Efficient principled learning of thin junction trees. In *Advances in Neural Information Processing Systems 20 (NIPS)*, pages 273–280. MIT Press, 2008.

D. M. Chickering. Learning Bayesian networks is NP-Complete. In *Learning from Data: Artificial Intelligence and Statistics V*, pages 121–130. Springer-Verlag, 1996.

D. M. Chickering, D. Heckerman, and C. Meek. Large-sample learning of Bayesian networks is NP-Hard. *Journal of Machine Learning Research*, 5:1287–1330, 2004.

C. K. Chow and C. N. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, 14(3): 462–467, 1968.

G. F. Cooper. The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence*, 42:393–405, 1990.

G. F. Cooper and E. Herskovits. A Bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9:309–347, 1992.

G. Elidan and S. Gould. Learning bounded treewidth Bayesian networks. *Journal of Machine Learning Research*, 9:2699–2731, 2008.

A. Frank and A. Asuncion. UCI machine learning repository, 2010.

D. Heckerman, D. Geiger, and D. M. Chickering. Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, 20(3): 197–243, 1995.

T. Johnson, N. Robertson, P.D. Seymour, and R. Thomas. Directed tree-width. *Journal of Combinatorial Theory, Series B*, 82(1):138–154, 2001.

E. Y. Kang, I. Shpitser, and E. Eskin. Respecting Markov equivalence in computing posterior probabilities of causal graphical features. In *24th AAAI Conference on Artificial Intelligence (AAAI)*, pages 1175–1180. AAAI Press, 2010.

D. Karger and N. Srebro. Learning Markov networks: Maximum bounded tree-width graphs. In *12th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 392–401, 2001.

T. Kloks. *Treewidth: computations and approximations*. Springer, 1994.

M. Koivisto. Advances in exact Bayesian structure discovery in Bayesian networks. In *22nd Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 241–248. AUAI Press, 2006.

M. Koivisto and K. Sood. Exact Bayesian structure discovery in Bayesian networks. *Journal of Machine Learning Research*, 5:549–573, 2004.

S. Ordyniak and S. Szeider. Algorithms and complexity results for exact Bayesian structure learning. In *26th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 8–11, 2010.

S. Ott and S. Miyano. Finding optimal gene networks using biological constraints. *Genome Informatics*, 14:124–133, 2003.

P. Parviainen and M. Koivisto. Ancestor relations in the presence of unobserved variables. In *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD)*, pages 581–596. Springer, 2011.

E. Perrier, S. Imoto, and S. Miyano. Finding optimal Bayesian network given a super-structure. *Journal of Machine Learning Research*, 9:2251–2286, 2008.

T. Silander and P. Myllymäki. A simple approach for finding the globally optimal Bayesian network structure. In *22nd Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 445–452. AUAI Press, 2006.

A. P. Singh and A. W. Moore. Finding optimal Bayesian networks by dynamic programming. Technical Report CMU-CALD-05-106, Carnegie Mellon University, June 2005.

N. Srebro. Maximum likelihood Markov networks: An algorithmic approach. Master's thesis, Massachusetts Institute of Technology, 2000.

N. Srebro. Maximum likelihood bounded tree-width Markov networks. In *17th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 504–511. AUAI Press, 2001.

J. Tian and R. He. Computing posterior probabilities of structural features in Bayesian networks. In *25th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 538–547. AUAI Press, 2009.

J. van Leeuwen. Graph algorithms. In *Handbook of theoretical computer science, Vol. A*, pages 525–631. Elsevier, 1990.

Paper IV

Fedor V. Fomin, Petr A. Golovach, and Janne H. Korhonen.

On the parameterized complexity of cutting a few vertices from a graph.

IV

doi:10.1007/978-3-642-40313-2_38

On the Parameterized Complexity of Cutting a Few Vertices from a Graph

Fedor V. Fomin[1], Petr A. Golovach[1], and Janne H. Korhonen[2]

[1] Department of Informatics, University of Bergen
Norway
[2] Helsinki Institute for Information Technology HIIT
& Department of Computer Science, University of Helsinki
Finland

Abstract. We study the parameterized complexity of separating a small set of vertices from a graph by a small vertex-separator. That is, given a graph G and integers k, t, the task is to find a vertex set X with $|X| \leq k$ and $|N(X)| \leq t$. We show that

- the problem is fixed-parameter tractable (FPT) when parameterized by t but W[1]-hard when parameterized by k, and
- a terminal variant of the problem, where X must contain a given vertex s, is W[1]-hard when parameterized either by k or by t alone, but is FPT when parameterized by $k + t$.

We also show that if we consider edge cuts instead of vertex cuts, the terminal variant is NP-hard.

1 Introduction

We investigate two related problems that concern separating a small vertex set from a graph $G = (V, E)$. Specifically, we consider finding a vertex set X of size at most k such that

1. X is separated from the rest of V by a small cut (e.g. *finding communities in a social network*, cf. [14]), or
2. X is separated from the rest of V by a small cut and contains a specified terminal vertex s (e.g. *isolating a dangerous node*, cf. [11,13]).

We focus on *parameterized complexity* of the *vertex-cut versions* of these problems.

Parameterized Vertex Cuts. Our interest in the vertex-cut version stems from the following parameterized separation problem, studied by Marx [16]. Let $N(X)$ denote the vertex-neighborhood of X.

Cutting k Vertices
Input: Graph $G = (V, E)$, integers $k \geq 1$, $t \geq 0$
Parameter 1: k
Parameter 2: t
Question: Is there a set $X \subseteq V$ such that $|X| = k$ and $|N(X)| \leq t$?

K. Chatterjee and J. Sgall (Eds.): MFCS 2013, LNCS 8087, pp. 421–432, 2013.

In particular, Marx showed that Cutting k Vertices is W[1]-hard even when parameterized by both k and t. We contrast this result by investigating the parameterized complexity of the two related separation problems with relaxed requirement on the size of the separated set X.

Cutting at Most k Vertices
Input: Graph $G = (V, E)$, integers $k \geq 1$, $t \geq 0$
Parameter 1: k
Parameter 2: t
Question: Is there a non-empty set $X \subseteq V$ such that $|X| \leq k$ and $|N(X)| \leq t$?

Cutting at Most k Vertices with Terminal
Input: Graph $G = (V, E)$, terminal vertex s, integers $k \geq 1$, $t \geq 0$
Parameter 1: k
Parameter 2: t
Question: Is there a non-empty set $X \subseteq V$ such that $s \in X$, $|X| \leq k$ and $|N(X)| \leq t$?

We show that these closely related problems exhibit quite different complexity behaviors. In particular, we show that Cutting at Most k Vertices is fixed-parameter tractable (FPT) when parameterized by the size of the separator t, while we need both k and t as parameters to obtain an FPT algorithm for Cutting at Most k Vertices with Terminal. A full summary of the parameterized complexity of these problems and our results is given in Table 1.

The main algorithmic contribution of our paper is the proof that Cutting at most k vertices is FPT when parameterized by t (Theorem 2). To obtain this result, we utilize the concept of *important separators* introduced by Marx [16]. However, a direct application of important separators—guess a vertex contained in the separated set, and find a minimal set containing this vertex that can be separated from the remaining graph by at most t vertices—does not work. Indeed, pursuing this approach would bring us to essentially solving Cutting at most k vertices with terminal, which is W[1]-hard when parameterized by t. Our FPT algorithm is based on new structural results about unique important

Table 1. Parameterized complexity of Cutting k Vertices, Cutting at Most k Vertices, and Cutting at Most k Vertices with Terminal

Parameter	Cutting k Vertices	Cutting $\leq k$ Vertices	Cutting $\leq k$ Vertices with Terminal
k	W[1]-hard, [16]	W[1]-hard, Thm 3	W[1]-hard, Thm 3
t	W[1]-hard, [16]	FPT, Thm 2	W[1]-hard, Thm 5
k and t	W[1]-hard, [16]	FPT, Thm 1	FPT, Thm 1

separators of minimum size separating pairs of vertices. We also observe that it is unlikely that Cutting at most k vertices has a polynomial kernel.

Edge Cuts. Although our main focus is on vertex cuts, we will also make some remarks on the edge-cut versions of the problems. In particular, the edge-cut versions again exhibit a different kind of complexity behavior. Let $\partial(X)$ denote the edge-boundary of X.

Cutting at Most k Vertices by Edge-Cut
Input: Graph $G = (V, E)$, integers $k \geq 1$, $t \geq 0$
Parameter 1: k
Parameter 2: t
Question: Is there a non-empty set $X \subseteq V$ such that $|X| \leq k$ and $|\partial(X)| \leq t$?

Cutting k Vertices by Edge-Cut with Terminal
Input: Graph $G = (V, E)$, terminal vertex s, integers $k \geq 1$, $t \geq 0$
Parameter 1: k
Parameter 2: t
Question: Is there a set $X \subseteq V$ such that $s \in X$, $|X| \leq k$ and $|\partial(X)| \leq t$?

Results by Watanabe and Nakamura [19] imply that Cutting at most k vertices by edge-cut can be done in polynomial time even when k and t are part of the input; more recently, Armon and Zwick [2] have shown that this also holds in the edge-weighted case. Lokshtanov and Marx [15] have proven that Cutting at most k vertices by edge-cut with terminal is fixed-parameter tractable when parameterized by k or by t; see also [5]. We complete the picture by showing that Cutting at most k vertices by edge-cut with terminal is NP-hard (Theorem 6). The color-coding techniques we employ in Theorem 1 also give a simple algorithm with running time $2^{k+t+o(k+t)} \cdot n^{O(1)}$.

Related edge-cut problems have received attention in the context of approximation algorithms. In contrast to Cutting at most k vertices by edge-cut, finding a minimum-weight edge-cut that separates exactly k vertices is NP-hard. Feige et al. [9] give a PTAS for $k = O(\log n)$ and an $O(k/\log n)$-approximation for $k = \Omega(\log n)$; Li and Zhang [14] give an $O(\log n)$-approximation. Approximation algorithms have also been given for unbalanced s-t-cuts, where s and t are specified terminal vertices and the task is to find an edge cut $(X, V \setminus X)$ with $s \in X$ and $t \in V \setminus S$ such that (a) $|X| \leq k$ and weight of the cut is minimized [11,14], or (b) weight of the cut is at most w and $|X|$ is minimized [13].

2 Basic Definitions and Preliminaries

Graph Theory. We follow the conventions of Diestel [7] with graph-theoretic notations. We only consider finite, undirected graphs that do not contain loops or multiple edges. The vertex set of a graph G is denoted by $V(G)$ and the edge

set is denoted by $E(G)$, or simply by V and E, respectively. Typically we use n to denote the number of vertices of G and m the number of edges.

For a set of vertices $U \subseteq V(G)$, we write $G[U]$ for the subgraph of G induced by U, and $G-U$ for the graph obtained form G by the removal of all the vertices of U, i.e., the subgraph of G induced by $V(G) \setminus U$. Similarly, for a set of edges A, the graph obtained from G by the removal of all the edges in A is denoted by $G - A$.

For a vertex v, we denote by $N_G(v)$ its (*open*) *neighborhood*, that is, the set of vertices which are adjacent to v. The *degree* of a vertex v is $d_G(v) = |N_G(v)|$. For a set of vertices $U \subseteq V(G)$, we write $N_G(U) = \cup_{v \in U} N_G(v) \setminus U$ and $\partial_G(U) = \{uv \in E(G) \mid u \in U, v \in V(G) \setminus U\}$. We may omit subscripts in these notations if there is no danger of ambiguity.

Submodularity. We will make use of the well-known fact that given a graph G, the mapping $2^V \to \mathbb{Z}$ defined by $U \mapsto |N(U)|$ is *submodular*. That is, for $A, B \subseteq V$ we have

$$|N(A \cap B)| + |N(A \cup B)| \leq |N(A)| + |N(B)| \ . \tag{1}$$

Important Separators. Let G be a graph. For disjoint sets $X, Y \subseteq V$, a vertex set $S \subseteq V \setminus (X \cup Y)$ is a (*vertex*) (X,Y)-*separator* if there is no path from X to Y in $G - S$. An *edge* (X,Y)-*separator* $A \subseteq E$ is defined analogously. Note that we do not allow deletion of vertices in X and Y, and thus there are no vertex (X,Y)-separators if X and Y are adjacent. As our main focus is on the vertex-cut problems, all separators are henceforth vertex-separators unless otherwise specified.

We will make use of the concept of *important separators*, introduced by Marx [16]. A vertex v is *reachable* from a set $X \subseteq V$ if G has a path that joins a vertex of X and v. For any sets S and $X \subseteq V \setminus S$, we denote the set of vertices reachable from X in $G - S$ by $R(X,S)$. An (X,Y)-separator S is *minimal* if no proper subset of S is an (X,Y)-separator. For (X,Y)-separators S and T, we say that T *dominates* S if $|T| \leq |S|$ and $R(X,S)$ is a proper subset of $R(X,T)$. For singleton sets, we will write x instead of $\{x\}$ in the notations defined above.

Definition 1 ([16]). *An (X,Y)-separator S is* important *if it is minimal and there is no other (X,Y)-separator dominating S.*

In particular, this definition implies that for any (X,Y)-separator S there exists an important (X,Y)-separator T with $|T| \leq |S|$ and $R(X,T) \supseteq R(X,S)$. If S is not important, then at least one of the aforementioned relations is proper.

The algorithmic usefulness of important separators follows from the fact that the number of important separators of size at most t is bounded by t alone, and furthermore, these separators can be listed efficiently. Moreover, minimum-size important separators are unique and can be found in polynomial time. That is, we will make use of the following lemmas.

Lemma 1 ([6]). *For any disjoint sets $X, Y \subseteq V$, the number of important (X, Y)-separators of size at most t is at most 4^t, and all important (X, Y)-separators of size at most t can be listed in time $4^t \cdot n^{O(1)}$.*

Lemma 2 ([16]). *For any sets $X, Y \subseteq V$, if there exists an (X, Y)-separator, then there is exactly one important (X, Y)-separator of minimum size. This separator can be found in polynomial time.*

Parameterized Complexity. We will briefly review the basic notions of parameterized complexity, though we refer to the books of Downey and Fellows [8], Flum and Grohe [10], and Niedermeier [18] for a detailed introduction. Parameterized complexity is a two-dimensional framework for studying the computational complexity of a problem; one dimension is the input size n and another one is a parameter k. A parameterized problem is *fixed-parameter tractable* (or FPT) if it can be solved in time $f(k) \cdot n^{O(1)}$ for some function f, and in the class XP if it can be solved in time $O\left(n^{f(k)}\right)$ for some function f.

Between FPT and XP lies the class W[1]. One of basic assumptions of the parameterized complexity theory is the conjecture that W[1] $\neq$ FPT, and it is thus held to be unlikely that a W[1]-hard problem would be in FPT. For exact definition of W[1], we refer to the books mentioned above. We mention only that Indpendent Set and Clique parameterized by solution size are two fundamental problems that are known to be W[1]-complete.

The basic way of showing that a parameterized problem is unlikely to be fixed-parameter tractable is to prove W[1]-hardness. To show that a problem is W[1]-hard, it is enough to give a *parameterized reduction* from a known W[1]-hard problem. That is, let A, B be parameterized problems. We say that A is (uniformly many-one) FPT-*reducible* to B if there exist functions $f, g : \mathbb{N} \to \mathbb{N}$, a constant $c \in \mathbb{N}$ and an algorithm $\mathcal{A}$ that transforms an instance (x, k) of A into an instance $(x', g(k))$ of B in time $f(k)|x|^c$ so that $(x, k) \in A$ if and only if $(x', g(k)) \in B$.

Cutting Problems with Parameters k and t. In the remainder of this section, we consider Cutting at most k vertices, Cutting at most k vertices with terminal, and Cutting at most k vertices by edge-cut with terminal with parameters k and t. We first note that if there exists a solution for one of the problems, then there is also a solution in which X is connected; indeed, it suffices to take any maximal connected $Y \subseteq X$. Furthermore, we note finding a connected set X with $|X| = k$ and $|N(X)| \leq t$ is fixed-parameter tractable with parameters k and t due to a result by Marx [16, Theorem 13], and thus Cutting at most k vertices is also fixed-parameter tractable with parameters k and t.

We now give a simple color-coding algorithm [1,4] for the three problems with parameters k and t, in particular improving upon the running time of the aforementioned algorithm for Cutting at most k vertices.

Theorem 1. Cutting at most k vertices, Cutting at most k vertices with terminal, *and* Cutting at most k vertices by edge-cut with terminal *can be solved in time* $2^{k+t} \cdot (k+t)^{O(\log(k+t))} \cdot n^{O(1)}$.

Proof. We first consider a 2-colored version of Cutting at most k vertices. That is, we are given a graph G where each vertex is either colored red or blue (this is not required to be a proper coloring), and the task is to find a connected red set X with $|X| \leq k$ such that $N(X)$ is blue and $|N(X)| \leq t$. If such a set exists, it can be found in polynomial time by trying all maximal connected red sets.

Now let $G = (V, E)$ be a graph. Assume that there is a set X with $|X| \leq k$ and $|N(X)| \leq t$; we may assume that X is connected. It suffices to find a coloring of V such that X is colored red and $N(X)$ is colored blue. This can be done by coloring each vertex v either red or blue independently and uniformly at random. Indeed, this gives a desired coloring with probability at least $2^{-(k+t)}$, which immediately yields a $2^{k+t} \cdot n^{O(1)}$ time randomized algorithm for Cutting at Most k Vertices.

This algorithm can be derandomized in standard fashion using universal sets (compare with Cai et al. [4]). Recall that *a* (n, ℓ)*-universal set* is a collection of binary vectors of length n such that for each index subset of size ℓ, each of the 2^ℓ possible combinations of values appears in some vector of the set. A construction of Naor et al. [17] gives a (n, ℓ)-universal set of size $2^\ell \cdot \ell^{O(\log \ell)} \log n$ that can be listed in linear time. It suffices to try all colorings induced by a $(n, k+t)$-universal set obtained trough this construction.

The given algorithm works for Cutting at most k vertices with terminal with obvious modifications. That is, given a coloring, we simply check if the terminal s is red and its connected red component is a solution. This also works for Cutting at most k vertices by edge-cut with terminal, as we have $|N(X)| \leq |\partial(X)|$. □

3 Cutting at Most k Vertices Parameterized by t

In this section we show that Cutting at Most k Vertices is fixed-parameter tractable when parameterized by the size of the separator t only. Specifically, we will prove the following theorem.

Theorem 2. Cutting at Most k Vertices *can be solved in time* $4^t \cdot n^{O(1)}$.

The remainder of this section consists of the proof of Theorem 2. Note that we may assume $\frac{3}{4}t < k < n - t$. Indeed, if $k \leq ct$ for a fixed constant $c < 1$, then we can apply the algorithm of Theorem 1 to solve Cutting at Most k Vertices in time $4^t n^{O(1)}$. On the other hand, if $k \geq n - t$, then any vertex set X of size k is a solution, as $|N(X)| \leq n - k \leq t$.

We start by guessing a vertex $u \in V$ that belongs to a solution set X if one exists; specifically, we can try all choices of u. We cannot expect to necessarily find a solution X that contains the chosen vertex u, even if the guess is correct,

as the terminal variant is W[1]-hard. We will nonetheless try; turns out that the only thing that can prevent us from finding a solution containing u is that we find a solution *not* containing u.

With u fixed, we compute for each $v \in V \setminus (\{u\} \cup N(u))$ the unique minimum important (u, v)-separator S_v. This can be done in polynomial time by Lemma 2. Let V_0 be set of those v with $|S_v| \leq t$, and denote $R(v) = R(v, S_v)$. Finally, let $\mathcal{X}$ be a set family consisting of those $R(v)$ for $v \in V_0$ that are inclusion-minimal, i.e., if $R(v) \in \mathcal{X}$, then there is no $w \in V_0$ such that $R(w) \subsetneq R(v)$. Note that we can compute the sets V_0, $R(v)$ and $\mathcal{X}$ in polynomial time.

There are now three possible cases that may occur.

1. If $V_0 = \emptyset$, we conclude that we have no solution containing u.
2. If there is $v \in V_0$ such that $|R(v)| \leq k$, then $X = R(v)$ gives a solution, and we stop and return a YES-answer.
3. Otherwise, $\mathcal{X}$ is non-empty and for all sets $A \in \mathcal{X}$ we have $|A| > k$.

We only have to consider the last case, as otherwise we are done. We will show that in that case, the sets $A \in \mathcal{X}$ can be used to find a solution X containing u if one exists. For this, we need the following structural results about the sets $R(v)$.

Lemma 3. *For any $v, w \in V_0$, if $w \in R(v)$ then $R(w) \subseteq R(v)$.*

Proof. Let $A = R(v)$ and $B = R(w)$. Since $S_v = N(A)$ is a (u, v)-separator of minimum size, we must have $|N(A \cup B)| \geq |N(A)|$. By (1), we have

$$|N(A \cap B)| \leq |N(A)| + |N(B)| - |N(A \cup B)| \leq |N(B)| \ .$$

Because $w \in A$, the set $N(A \cap B)$ is a (u, w)-separator. Thus, if $B \neq A \cap B$, then $N(A \cap B)$ is a (u, w)-separator that witnesses that S_w is not an important separator. But this is not possible by the definition of S_w, so we have $B = A \cap B \subseteq A$. □

Lemma 4. *Any distinct $A, B \in \mathcal{X}$ are disjoint.*

Proof. Assume that $A, B \in \mathcal{X}$ are distinct and intersect. Then there is $v \in A \cap B$. Since $v \in A$, the set $N(A)$ is a (u, v)-separator of size at most t, and $v \in V_0$. Recall that $\mathcal{X}$ contains inclusion-minimal sets $R(w)$ for $w \in V_0$. But by Lemma 3, $R(v)$ is a proper subset of both A and B, which is not possible by the definition of $\mathcal{X}$. □

Now assume that the input graph G has a solution for Cutting at Most k Vertices containing u. In particular, then there is an inclusion-minimal set $X \subseteq V$ with $u \in X$ satisfying $|X| \leq k$ and $|N(X)| \leq t$. Let us fix one such set X.

Lemma 5. *For all $A \in \mathcal{X}$, the set A is either contained in $X \cup N(X)$ or does not intersect it.*

Proof. Suppose that there is a set $A \in \mathfrak{X}$ that intersects both $X \cup N(X)$ and its complement. Let $Y = V \setminus (X \cup N(X))$.

Now let $v \in A \cap Y$. By Lemma 3, we have $R(v) = A$. If $|N(A \cup Y)| > |N(Y)|$ then it follows from (1) that

$$|N(A \cap Y)| \leq |N(A)| + |N(Y)| - |N(A \cup Y)| < |N(A)| \, .$$

However, this would imply that $N(A \cap Y)$ is a (u, v)-separator smaller than $S_v = N(A)$.

Thus, we have $|N(A \cup Y)| \leq |N(Y)|$. But $X' = X \setminus (A \cup Y \cup N(A \cup Y))$ is a proper subset of X; furthermore, any vertex of $N(X')$ that is not in $N(A \cup Y)$ is also in $N(X) \setminus N(Y)$, so we have $|N(X')| \leq |N(X)| \leq t$. This is in contradiction with the minimality of X. □

Lemma 6. *Let Z be the union of all $A \in \mathfrak{X}$ that do not intersect $X \cup N(X)$. Then $Z \neq \emptyset$ and there is an important (Z, u)-separator S of size at most t such that $|R(u, S)| + |S| \leq k + t$.*

Proof. Let $S = N(X)$. Consider an arbitrary $v \in V \setminus (X \cup S)$; such vertex exists, since $k + t < n$. Since S separates v from u, the set $R(v)$ is well-defined.

Suppose now that $R(v)$ is not contained in $R(v, S)$. Let $B = R(u, S_v)$. Since S_v is a minimum-size (u, v)-separator we have $|N(B)| = |N(R(v))|$. But $N(X \cup B)$ also separates u and v, so we have $|N(X \cup B)| \geq |N(R(v))| = |N(B)|$. By (1), we have

$$|N(X \cap B)| \leq |N(X)| + |N(B)| - |N(X \cup B)| \leq |N(X)| \leq t \, .$$

But since $R(v)$ is not contained $R(v, S)$, it follows that $X \cap B$ is a proper subset of X, which contradicts the minimality of X.

Thus we have $R(v) \subseteq R(v, S)$. It follows that $R(v, S)$ contains a set $A \in \mathfrak{X}$, which implies that $Z \neq \emptyset$ and $v \in R(A, S) \subseteq R(Z, S)$. Furthermore, since $v \in V \setminus (X \cup S)$ was chosen arbitrarily, we have that $R(Z, S) = V \setminus (X \cup S)$.

If S is an important (Z, u)-separator, we are done. Otherwise, there is an important (Z, u)-separator T with $|T| \leq |S|$ and $R(Z, S) \subseteq R(Z, T)$. But then we have $|T| \leq t$, and $R(u, T) \cup T \subseteq X \cup S$, that is, $|R(u, T) \cup T| \leq k + t$. □

Recall now that we may assume $|A| > k$ for all $A \in \mathfrak{X}$. Furthermore, we have $|X \cup N(X)| \leq k + t < \left(2 + \frac{1}{3}\right) k$ and the sets $A \in \mathfrak{X}$ are disjoint by Lemma 4. Thus, at most two sets $A \in \mathfrak{X}$ fit inside $X \cup N(X)$ by Lemma 5. This means that if we let Z be the union of all $A \in \mathfrak{X}$ that do not intersect $X \cup N(X)$, then as we have already computed $\mathfrak{X}$, we can guess Z by trying all $O(n^2)$ possible choices.

Assume now that X is a minimal solution containing u and our guess for Z is correct. We enumerate all important (Z, u)-separators of size at most t. We will find by Lemma 6 an important (Z, u)-separator S such that $|S| \leq t$ and $|R(u, S)| + |S| \leq k + t$. If $|R(u, S)| \leq k$, we have found a solution. Otherwise, we delete a set S' of $|R(u, S)| - k$ elements from $R(u, S)$ to obtain a solution X'. To see that this suffices, observe that $N(X') \subseteq S' \cup S$. Therefore, $|N(X')| \leq |S'| + |S| = |R(u, S)| - k + |S| \leq t$. As all important (Z, u)-separators can be listed in time $4^t \cdot n^{O(1)}$ by Lemma 1, the proof of Theorem 2 is complete.

4 Hardness Results

We start this section by complementing Theorem 2, as we show that Cutting at Most k Vertices is NP-complete and W[1]-hard when parameterized by k. We also show that same holds for Cutting at Most k Vertices with Terminal. Note that both of these problems are in XP when parameterized by k, as they can be solved by checking all vertex subsets of size at most k.

Theorem 3. Cutting at Most k Vertices *and* Cutting at Most k Vertices with Terminal *are* NP*-complete and* W[1]*-hard with the parameter* k*.*

Proof. We prove the W[1]-hardness claim for Cutting at Most k Vertices by a reduction from Clique. Recall that this W[1]-complete (see [8]) problem asks for a graph G and a positive integer k where k is a parameter, whether G contains a clique of size k. Let (G, k) be an instance of Clique, $n = |V(G)|$ and $m = |E(G)|$; we construct an instance (G', k', t) of Cutting at Most k Vertices as follows. Let H_V be a clique of size n^3 and identify n vertices of H_V with the vertices of G. Let H_E be a clique of size m and identify the vertices of H_E with the edges of G. Finally, add an edge between vertex v of H_V and vertex e of H_E whenever v is incident to e in G. Set $k' = \binom{k}{2}$ and $t = k + m - \binom{k}{2}$. The construction is shown in Fig. 1 a).

If G has a k-clique K, then for the set X that consists of the vertices e of H_E corresponding to edges of K we have $|X| = \binom{k}{2}$ and $|N_{G'}(X)| = k + m - \binom{k}{2}$. On the other hand, suppose that there is a set of vertices X of G' such that $|X| \leq k'$ and $|N_{G'}(X)| \leq t$. First, we note that X cannot contain any vertices of H_V, as then $N_{G'}(X)$ would be too large. Thus, the set X consists of vertices of H_E. Furthermore, we have that $|X| = \binom{k}{2}$. Indeed, assume that this is not the case. If $|X| \leq \binom{k-1}{2} = \binom{k}{2} - k$, then, since X has at least one neighbor in H_V, we have

$$|N_{G'}(X)| \geq m - |X| + 1 \geq m - \binom{k}{2} + k + 1\,,$$

and if $\binom{k-1}{2} < |X| < \binom{k}{2}$, then X has at least k neighbors in H_V, and thus

$$|N_{G'}(X)| \geq m - |X| + k > m - \binom{k}{2} + k\,.$$

Thus, we have that X only consist of vertices of H_E and $|X| = \binom{k}{2}$. But then the vertices of H_V that are in $N_{G'}(X)$ form a k-clique in G.

The W[1]-hardness proof for Cutting at Most k Vertices with Terminal uses the same arguments. The only difference is that we add the terminal s in the clique H_E and let $k' = \binom{k}{2} + 1$ (see Fig. 1 b).

Because Clique is well known to be NP-complete [12] and our parameterized reductions are polynomial in k, it immediately follows that Cutting at Most k Vertices and Cutting at Most k Vertices with Terminal are NP-complete. □

While we have an FPT-algorithm for Cutting at Most k Vertices when parameterized by k and t or by t only, it is unlikely that the problem has a polynomial kernel (we refer to [8,10,18] for the formal definitions of kernels). Let G be a graph with s connected components $G_1, \ldots, G_s$, and let $k \geq 1$, $t \geq 0$ be integers. Now (G, k, t) is a YES-instance of Cutting at Most k Vertices if and only if (G_i, k, t) is a YES-instance for some $i \in \{1, \ldots, s\}$, because it can always be assumed that a solution is connected. By the results of Bodlaender et al. [3], this together with Theorem 3 implies the following.

Theorem 4. Cutting at Most k Vertices *has no polynomial kernel when parameterized either by k and t or by t only, unless* NP $\subseteq$ coNP/poly.

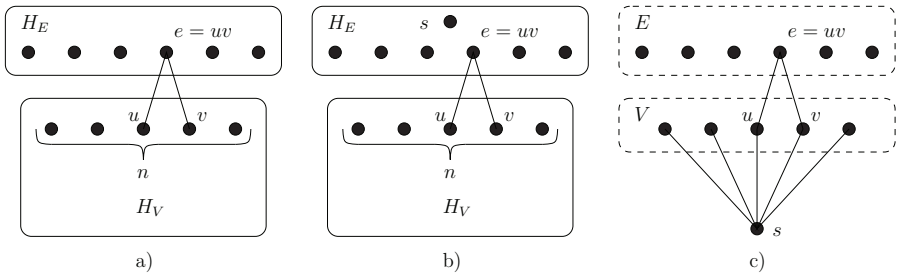


Fig. 1. Constructions of G' in the proofs of Theorems 3 and 5

We will next show that when we consider the size of the separator t as the sole parameter, adding a terminal makes the problem harder. Indeed, while Cutting at Most k Vertices with Terminal with parameter t is trivially in XP, we next show that it is also W[1]-hard, in contrast to Theorem 2.

Theorem 5. Cutting at Most k Vertices with Terminal *is* W[1]*-hard with parameter t.*

Proof. Again, we prove the claim by a reduction from Clique. Let (G, k) be a clique instance, $n = |V(G)|$ and $m = |E(G)|$; we create an instance (G', k', t, s) of Cutting at Most k Vertices with Terminal. The graph G' is constructed as follows. Create a new vertex s as the terminal. For each vertex and edge of G, add a corresponding vertex to G', and add an edge between vertices v and e in G' when e is incident to v in G. Finally, connect all vertices of G' corresponding to vertices of G to the terminal s, and set $k' = n - k + m - \binom{k}{2} + 1$ and $t = k$. The construction is shown in Fig. 1 c).

If G has a k-clique K, then cutting away the k vertices of G' corresponding to K leaves exactly $n - k + m - \binom{k}{2} + 1$ vertices in the connected component of $G' - K$ containing s. Now suppose that $X \subseteq V(G')$ is a set with $s \in X$ such that $|X| \leq k'$ and $|N_{G'}(X)| \leq t$, and let $S = N_{G'}(X)$. Note that the elements of $V(G')$ that do not belong to X are exactly the elements $v \in S$ and the vertices

corresponding to $e = uv$ such that $u, v \in S$ and $e \notin S$; denote this latter set of elements by E_0. Since X is a solution, we have $|S| + |E_0| \geq \binom{k}{2} + k$, and thus $|E_0| \geq \binom{k}{2}$. But this is only possible if S is a k-clique in G. □

Finally, we show that Cutting at most k vertices by edge-cut with terminal is also NP-hard.

Theorem 6. Cutting at most k vertices by edge-cut with terminal *is* NP*-complete.*

Proof. We give a reduction from the Clique problem. It is known that this problem is NP-complete for regular graphs [12]. Let (G, k) be an instance of Clique, with G being a d-regular n-vertex graph. We create an instance (G', k', t, s) of Cutting k Vertices by Edge-Cut with Terminal as follows. The graph G' is constructed by starting from a base clique of size dn. One vertex in this base clique is selected as the terminal s, and we additionally distinguish d special vertices. For each $v \in V(G)$, we add a new vertex to G', and add an edge between this vertex and all of the d distinguished vertices of the base clique. For each edge $e = uv$ in G, we also add a new vertex to G', and add edges between this vertex and vertices corresponding to u and v. The construction is shown in Fig. 2. We set $k' = dn + k + \binom{k}{2}$ and $t = dn - 2\binom{k}{2}$.

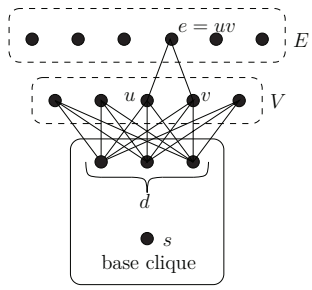


Fig. 2. Construction of G' in the proof of Theorem 6

If G has a k-clique K, then selecting as X the base clique and all vertices of G' corresponding to vertices and edges of K gives a solution to (G', k', t, s), as we have $|X| = dn + k + \binom{k}{2}$ and $|\partial(X)| = (dn - dk) + \left(dk - 2\binom{k}{2}\right) = dn - 2\binom{k}{2}$. For the other direction, consider any solution X to instance (G', k', t, s). The set X must contain the whole base clique, as otherwise there are at least $dn - 1$ edges inside the base clique that belong to $\partial(X)$. Let $V_0 \subseteq V$ and $E_0 \subseteq E$ be the subsets of X corresponding to vertices and edges of G, respectively. If $E_0 = \emptyset$, then $|\partial(X)| = dn$. Assume now that V_0 is fixed, and consider how adding vertices to E_0 changes $|\partial(X)|$. For each edge $e \in E(G)$, if neither of the endpoints of e is in V_0, then adding e to E_0 adds 2 to $|\partial(X)|$. If exactly one of the endpoints of e is in V_0, then adding e to E_0 does not change $|\partial(X)|$. Finally, if both of the endpoints of e are in V_0, then adding e to E_0 reduces $|\partial(X)|$ by 2. Thus, in order

to have $|\partial(X)| \leq dn - 2\binom{k}{2}$, we must have that $|E_0| \geq \binom{k}{2}$ and the endpoints of all edges in E_0 are in V_0. But due to the requirement that $|X| \leq dn + k + \binom{k}{2}$, this is only possible if V_0 induces a clique in G. □

Acknowledgments. We thank the anonymous reviewers for pointing out the related work of Lokshtanov and Marx. This work is supported by the European Research Council (ERC) via grant Rigorous Theory of Preprocessing, reference 267959 (F.F., P.G.) and by the Helsinki Doctoral Programme in Computer Science – Advanced Computing and Intelligent Systems (J.K.).

References

1. Alon, N., Yuster, R., Zwick, U.: Color-coding. Journal of the ACM 42(4), 844–856 (1995)
2. Armon, A., Zwick, U.: Multicriteria global minimum cuts. Algorithmica 46(1), 15–26 (2006)
3. Bodlaender, H.L., Downey, R.G., Fellows, M.R., Hermelin, D.: On problems without polynomial kernels. Journal of Computer and System Sciences 75(8), 423–434 (2009)
4. Cai, L., Chan, S.M., Chan, S.O.: Random separation: A new method for solving fixed-cardinality optimization problems. In: Bodlaender, H.L., Langston, M.A. (eds.) IWPEC 2006. LNCS, vol. 4169, pp. 239–250. Springer, Heidelberg (2006)
5. Cao, Y.: A note on small cuts for a terminal (2013), arXiv:1306.2578 [cs.DS]
6. Chen, J., Liu, Y., Lu, S.: An improved parameterized algorithm for the minimum node multiway cut problem. Algorithmica 55(1), 1–13 (2009)
7. Diestel, R.: Graph theory, 4th edn. Springer (2010)
8. Downey, R.G., Fellows, M.R.: Parameterized complexity. Springer (1999)
9. Feige, U., Krauthgamer, R., Nissim, K.: On cutting a few vertices from a graph. Discrete Applied Mathematics 127(3), 643–649 (2003)
10. Flum, J., Grohe, M.: Parameterized complexity theory. Springer (2006)
11. Galbiati, G.: Approximating minimum cut with bounded size. In: Pahl, J., Reiners, T., Voß, S. (eds.) INOC 2011. LNCS, vol. 6701, pp. 210–215. Springer, Heidelberg (2011)
12. Garey, M.R., Johnson, D.S.: Computers and intractability: a guide to the theory of NP-completeness. W. H. Freeman and Co., San Francisco (1979)
13. Hayrapetyan, A., Kempe, D., Pál, M., Svitkina, Z.: Unbalanced graph cuts. In: Brodal, G.S., Leonardi, S. (eds.) ESA 2005. LNCS, vol. 3669, pp. 191–202. Springer, Heidelberg (2005)
14. Li, A., Zhang, P.: Unbalanced graph partitioning. Theory of Computing Systems 53(3), 454–466 (2013)
15. Lokshtanov, D., Marx, D.: Clustering with local restrictions. Information and Computation 222, 278–292 (2013)
16. Marx, D.: Parameterized graph separation problems. Theoretical Computer Science 351(3), 394–406 (2006)
17. Naor, M., Schulman, L., Srinivasan, A.: Splitters and near-optimal derandomization. In: 36th Annual Symposium on Foundations of Computer Science (FOCS 1995), pp. 182–191. IEEE (1995)
18. Niedermeier, R.: Invitation to fixed-parameter algorithms. Oxford University Press (2006)
19. Watanabe, T., Nakamura, A.: Edge-connectivity augmentation problems. Journal of Computer and System Sciences 35(1), 96–144 (1987)

Paper V

Matti Järvisalo, Petteri Kaski, Mikko Koivisto, and Janne H. Korhonen.

Finding efficient circuits for ensemble computation.

doi:10.1007/978-3-642-31612-8_28

Finding Efficient Circuits for Ensemble Computation*

Matti Järvisalo[1], Petteri Kaski[2], Mikko Koivisto[1], and Janne H. Korhonen[1]

[1] HIIT & Department of Computer Science, University of Helsinki, Finland
[2] HIIT & Department of Information and Computer Science, Aalto University, Finland

Abstract. Given a Boolean function as input, a fundamental problem is to find a Boolean circuit with the least number of elementary gates (AND, OR, NOT) that computes the function. The problem generalises naturally to the setting of *multiple* Boolean functions: find the smallest Boolean circuit that computes *all* the functions simultaneously. We study an NP-complete variant of this problem titled *Ensemble Computation* and, especially, its relationship to the Boolean satisfiability (SAT) problem from both the theoretical and practical perspectives, under the two monotone circuit classes: OR-circuits and SUM-circuits. Our main result relates the existence of nontrivial algorithms for CNF-SAT with the problem of rewriting in subquadratic time a given OR-circuit to a SUM-circuit. Furthermore, by developing a SAT encoding for the ensemble computation problem and by employing state-of-the-art SAT solvers, we search for concrete instances that would witness a substantial separation between the size of optimal OR-circuits and optimal SUM-circuits. Our encoding allows for exhaustively checking all small witness candidates. Searching over larger witness candidates presents an interesting challenge for current SAT solver technology.

1 Introduction

A fundamental problem in computer science both from the theoretical and practical perspectives is program optimisation, i.e., the task of finding the most efficient sequence of elementary operations that carries out a specified computation. As a concrete example, suppose we have eight variables $x_1, x_2, \ldots, x_8$ and our task is to compute each of the eight sums depicted in Fig. 1. What is the minimum number of SUM gates that implement this computation?

This is an instance of a problem that plays a key role in Valiant's study [18] of circuit complexity over a monotone versus a universal basis; Fig. 1 displays Valiant's solution. More generally, the problem is an instantiation of the NP-complete *Ensemble Computation* problem [8]:

> *(SUM-)Ensemble Computation.* Given as input a collection $\mathcal{Q}$ of nonempty subsets of a finite set P and a nonnegative integer b, decide (yes/no) whether there is a sequence
>
> $$Z_1 \leftarrow L_1 \cup R_1, \quad Z_2 \leftarrow L_2 \cup R_2, \quad \ldots \quad , \quad Z_b \leftarrow L_b \cup R_b$$

* This research is supported in part by Academy of Finland (grants 132812 and 251170 (MJ), 252083 and 256287 (PK), and 125637 (MK)), and by Helsinki Doctoral Programme in Computer Science - Advanced Computing and Intelligent Systems (JK).

A. Cimatti and R. Sebastiani (Eds.): SAT 2012, LNCS 7317, pp. 369–382, 2012.

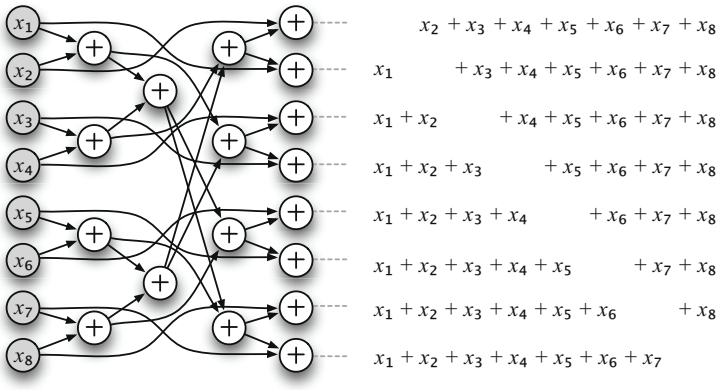


Fig. 1. An instance of ensemble computation (right) and a circuit that solves it (left)

of union operations, where

(a) for all $1 \leq j \leq b$ the sets L_j and R_j belong to $\{\{x\} : x \in P\} \cup \{Z_1, Z_2, \ldots, Z_{j-1}\}$,
(b) for all $1 \leq j \leq b$ the sets L_j and R_j are disjoint, and
(c) the collection $\{Z_1, Z_2, \ldots, Z_b\}$ contains $\mathcal{Q}$.

It is also known that *SUM-Ensemble Computation* remains NP-complete even if the requirement (b) is removed, that is, the unions need not be disjoint [8]; we call this variant *OR-Ensemble Computation*. Stated in different but equivalent terms, each set A in $\mathcal{Q}$ in an instance of *SUM-Ensemble Computation* specifies a subset of the variables in P whose sum must be computed. The question is to decide whether b arithmetic gates suffice to evaluate all the sums in the ensemble. An instance of *OR-Ensemble Computation* asks the same question but with sums replaced by ORs of Boolean variables, and with SUM-gates replaced by OR-gates. We will refer to the corresponding circuits as *SUM-circuits* and *OR-circuits*.

Despite the fundamental nature of these two variants of monotone computation, little seems to be known about their relative power. In particular, here we focus the following open questions:

(Q1) Given an OR-circuit for a collection $\mathcal{Q}$, how efficiently can it be rewritten as a SUM-circuit?
(Q2) Are there collections $\mathcal{Q}$ that require a significantly larger SUM-circuit than an OR-circuit?

Answering these questions would advance our understanding of the computational advantage of, in algebraic terms, idempotent computation (e.g. the maximum of variables) over non-idempotent computation (e.g. the sum of variables); the ability to express the former succinctly in terms of the latter underlies recent advances in algebraic and combinatorial algorithms [2]. Interestingly, it turns out that the questions have strong connections to Boolean satisfiability (SAT) both from the theoretical and practical perspectives, as will be shown in this paper.

As the main theoretical contribution, we establish a connection between (Q1) and the existence of non-trivial algorithms for CNF-SAT. In particular, we show (Theorem 2) that the existence of a subquadratic-time rewriting algorithm implies a nontrivial algorithm for general CNF-SAT (without restrictions on clause length), i.e., an algorithm for CNF-SAT that runs in time $O(2^{cn}m^2n)$ for a constant $0 < c < 1$ that is independent of the number of variables n and the number of clauses m. It should be noted that the existence of such an algorithm for CNF-SAT is a question that has attracted substantial theoretical interest recently [3,14,16,21]. In particular, such an algorithm would contradict the *Strong Exponential Time Hypothesis* [11], and would have significant implications also for the exponential-time complexity of other hard problems beyond SAT. Intuitively, our result suggests that the relationship of the two circuit classes may be complicated and that the difference in the circuit sizes could be large for some collections $\mathcal{Q}$. Furthermore, we show (Proposition 2) that our main result is tight in the sense that (Q1) admits an quadratic-time algorithm.

Complementing our main theoretical result, we address (Q2) from the practical perspective. While it is easy to present concrete instances for which the difference in size between optimal SUM-circuits and OR-circuits is small, finding instances that witness even a factor-2 separation between the number of arithmetic gates is a non-trivial challenge. In fact, our best construction (Theorem 1) achieves this factor only asymptotically, leaving open the question whether there are small witnesses achieving factor 2. As the main practical contribution, we employ state-of-the-art SAT solvers for studying this witness finding task by developing a SAT encoding for finding the optimal circuits for a given ensemble. We show experimentally that our encoding allows for exhaustively checking all small witness candidates. On the other hand, searching over larger witness candidates presents an interesting challenge for current SAT solvers.

As for related earlier work, SAT solvers have been suggested for designing small circuits [4,6,7,12,13], albeit of different types than the ones studied in this work. However, our focus here is especially in circuits implementing an *ensemble* of Boolean functions. A further key motivation that sets this work apart from earlier work is that our interest is not only to find efficient circuits, but also to discover witnesses (ensembles) that separate SUM-circuits and OR-circuits.

2 OR-Circuits, SUM-Circuits, and Rewriting

We begin with some key definitions and basic results related to OR- and SUM-circuits and the task of rewriting an OR-circuit into a SUM-circuit: We show that a SUM-circuit may require asymptotically at least twice as many arithmetic gates as an OR-circuit, and present two rewriting algorithms, one of which rewrites a given OR-circuit with g gates in $O(g^2)$ time into a SUM-circuit. In particular, a SUM-circuit requires at most g times as many arithmetic gates as an OR-circuit.

2.1 Definitions

For basic graph-theoretic terminology we refer to West's introduction [19]. A *circuit* is a directed acyclic graph C whose every node has in-degree either 0 or 2. Each node of

C is a *gate*. The gates of C are partitioned into two sets: each gate with in-degree 0 is an *input gate*, and each gate with in-degree 2 is an *arithmetic gate*. The *size* of C is the number $g = g(C)$ of gates in C. We write $p = p(C)$ for the number of input gates in C. For example, the directed acyclic graph depicted on the left in Fig. 1 is a circuit with 26 gates that partition into 8 input gates and 18 arithmetic gates.

The *support* of a gate z in C is the set of all input gates x such that there is a directed path in C from x to z. The *weight* of a gate z is the size of its support. All gates have weight at least one, with equality if and only if a gate is an input gate. For example, in Fig. 1 the five columns of gates consist of gates that have weight 1, 2, 4, 6, and 7, respectively.

In what follows we study two classes of circuits, where the second class is properly contained within the first class. First, every circuit is an *OR-circuit*. Second, a circuit C is a *SUM-circuit* if for every gate z and for every input gate x it holds that there is at most one directed path in C from x to z.

We adopt the convention of using the operator symbols "$\vee$" and "$+$" on the arithmetic gates to indicate the type of a circuit. Fig. 2 below displays an example of both types of circuits. We observe that the circuit on the left in Fig. 2 is not a SUM-circuit because the bottom right gate can be reached from the input x_1 along two distinct directed paths.

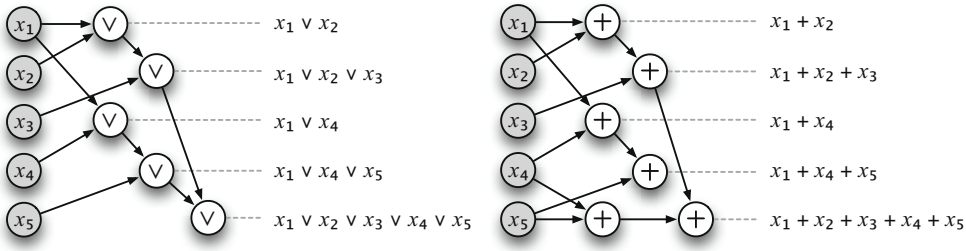


Fig. 2. An OR-circuit (left) and a SUM-circuit (right)

Let $(P, \mathcal{Q})$ be an instance of ensemble computation, that is, let P be a finite set and let $\mathcal{Q}$ be a set of nonempty subsets of P. We adopt the convention that for a SUM-ensemble all circuits considered are SUM-circuits, and for an OR-ensemble all circuits considered are OR-circuits. We say that a circuit C *solves* the instance $(P, \mathcal{Q})$ if (a) the set of input gates of C is P; and (b) for each $A \in \mathcal{Q}$, there exists a gate in C whose support is A. The *size* of the solution is the size of C. A solution to $(P, \mathcal{Q})$ is *optimal* if it has the minimum size over all possible solutions. A circuit C' *implements* a circuit C if for every gate z of C there is a gate z' of C' such that z and z' have the same support. A *circuit rewriting algorithm* takes as input a circuit C and outputs (i) a circuit C' that implements C; and (ii) a mapping $z \mapsto z'$ that identifies each gate z in C with a corresponding gate z' in C'.

2.2 Bounds for Separation

The size of an optimal solution to an instance $(P, \mathcal{Q})$ is dependent on whether we are considering an OR-ensemble or a SUM-ensemble. To see this, let us consider Fig. 2.

Observe that both circuits solve the same instance $(P, \mathcal{Q})$, but only the circuit on the right is a SUM-circuit. We claim that both circuits are optimal. Indeed, observe that the instance has five distinct sets of size at least 2. At least one arithmetic gate is required for each distinct set of size at least 2. Thus, the circuit on the left in Fig. 2 is optimal. Analogously, on the right in Fig. 2 at least four arithmetic gates are required to compute the first four sets in the instance, after which at least two further SUM-gates are required to produce the fifth set because the first four sets intersect pairwise.

The following construction shows that asymptotically (that is, by taking a large enough h and w) at least twice the number of arithmetic gates may be required in an optimal SUM-circuit compared with an optimal OR-circuit.

Theorem 1. *For all $h, w = 1, 2, \ldots$ there exists an ensemble whose optimal OR-circuit has $(h + 1)w - 1$ arithmetic gates and whose optimal SUM-circuit has $(2w - 1)h$ arithmetic gates.*

Proof. Take $P = \{x_0\} \cup \{x_{i,j} : i = 1, 2, \ldots, h; j = 1, 2, \ldots, w\}$ and let $\mathcal{Q}$ consist of the following sets. For each $j = 1, 2, \ldots, w$ and for each $i = 1, 2, \ldots, h$, insert the set $\{x_0, x_{1,j}, x_{2,j}, \ldots, x_{i,j}\}$ to $\mathcal{Q}$. Let us say that this set belongs to *chain* j. Finally, insert the set P into $\mathcal{Q}$. Let us call this set the *top*. In total $\mathcal{Q}$ thus has $hw + 1$ sets, and the largest set (that is, the top) has size $hw + 1$.

Every OR-circuit that solves $(P, \mathcal{Q})$ must use one OR-gate for each element in each chain for a total of hw gates. Excluding the element x_0 which occurs in all sets in $\mathcal{Q}$, the top has size hw, and the largest sets in each chain have size h. Thus, at least $w - 1$ OR-gates are required to construct the top. In particular, an optimum OR-circuit that solves $(P, \mathcal{Q})$ has $hw + w - 1 = (h + 1)w - 1$ arithmetic gates.

Next consider an arbitrary SUM-circuit that solves $(P, \mathcal{Q})$. Observe that each chain requires h distinct SUM-gates, each of which has x_0 in its support. There are hw such SUM-gates in total, at most one of which may be shared in the subcircuit that computes the top. Such a shared SUM-gate has weight at most $h + 1$, whereas the top has weight $hw + 1$. Thus the subcircuit that computes the top can share weight at most $h + 1$ and must use non-shared SUM-gates to accumulate the remaining weight (if any), which requires $h(w - 1)$ gates. Thus, the SUM-circuit requires at least $hw + h(w - 1) = (2w - 1)h$ arithmetic gates. □

Remark 1. Traditional nonconstructive tools for deriving lower bounds to circuit size appear difficult to employ for this type of separation between two monotone circuit classes. Indeed, it is easy to show using standard counting arguments that most ensembles $(P, \mathcal{Q})$ with $|P| = |\mathcal{Q}| = r$ require $\Omega(r^2 / \log r)$ gates for both OR- and SUM-circuits, but showing that there exist ensembles where the required SUM-circuit is significantly larger than a sufficient OR-circuit appears inaccessible to such tools.

2.3 Upper Bounds for Rewriting

Let us now proceed to study the algorithmic task of rewriting a given OR-circuit into a SUM-circuit. In particular, our interest is to quantify the number of extra gates required. We start with the observation that no extra gates are required if all gates in the given OR-circuit have weight at most 4.

Proposition 1. *Every OR-circuit with g gates of weight at most 4 can be rewritten into a SUM-circuit with g gates. Moreover, there is an algorithm with running time $O(g)$ that rewrites the circuit.*

Proof. Let C be an OR-circuit with g gates given as input. First, topologically sort the nodes of C in time $O(g)$. Then, compute the support of each gate by assigning unique singleton sets at the input gates and evaluating the gates in topological order. Finally, proceed in topological order and rewrite the gates of the circuit using the following rules. Input gates do not require rewriting. Furthermore, every OR-gate of weight 2 can be trivially replaced with a SUM-gate. Each OR-gate z with weight 3 either has the property that the in-neighbours z_1, z_2 of z have disjoint supports (in which case we may trivially replace z with a SUM-gate) or z_1, z_2 have weight at least 2. In the latter case, if at least one of z_1, z_2 has weight 3 (say, z_1), we may delete z and replace it with z_1; otherwise rewrite z so that one of its in-neighbours is z_1 and the other in-neighbour is the appropriate input gate. Each OR-gate z with weight 4 either has in-neighbours z_1, z_2 with disjoint supports or z_1, z_2 have weight at least 3 and at least 2, respectively. Again we may either delete z or rewrite z so that one of its in-neighbours is z_1 and the other in-neighbour is the appropriate input gate. It is immediate that this rewriting can be carried out in time $O(g)$. □

Next we observe that an OR-circuit can always be rewritten into a SUM-circuit with at most g times the number of gates in the OR-circuit.

Proposition 2. *There exists an algorithm that in time $O(g^2)$ rewrites a given OR-circuit with g gates into a SUM-circuit.*

Proof. The algorithm operates as follows. Let C be an OR-circuit with g gates and p input gates given as input. Topologically sort the nodes of C in time $O(g)$. Suppose the input gates of C are $x_1, x_2, \ldots, x_p$. Associate with each of the g gates an array of p bits. Then, iterate through the gates of C in topological order. For each input gate x_j, initialise the bit array associated with x_j so that the jth bit is set to 1 and the other bits are set to 0. For each OR-gate z with in-neighbours z_1, z_2, assign the bit array associated with z to be the union of the bit arrays associated with z_1 and z_2. This step takes time $O(gp)$. Finally, iterate through the gates of C. For each arithmetic gate z, output a SUM-circuit that computes the sum of the at most p inputs specified by the bit array associated with z. This requires at most $p - 1$ SUM-gates for each z. The algorithm takes $O(gp)$ time and outputs a circuit with $O(gp)$ gates. The claim follows because $p \leq g$. □

3 Subquadratic Rewriting Implies Faster CNF-SAT

Complementing the quadratic-time algorithm in Proposition 2, this section studies the possibility of developing fast (subquadratic-time) algorithms for rewriting OR-circuits as SUM-circuits. In particular, we show that the existence of such a subquadratic-time rewriting algorithm would, surprisingly, yield a non-trivial algorithm for general CNF-SAT (cf. Refs. [16,21] and [20, Theorem 5]).

Theorem 2. *Let $0 < \epsilon \leq 1$. If there is an algorithm that in time $O(g^{2-\epsilon})$ rewrites a given OR-circuit with g gates into a SUM-circuit, then there is an algorithm that solves CNF-SAT in time $O\big(2^{(1-\epsilon/2)n}m^{2-\epsilon}n\big)$, where n is the number of variables and m is the number of clauses.*

Proof. Let $0 < \epsilon \leq 1$ be fixed and let A be a circuit rewriting algorithm with the stated properties. We present an algorithm for CNF-SAT. Let an instance of CNF-SAT given as input consist of the variables $x_1, x_2, \ldots, x_n$ and the clauses $C_1, C_2, \ldots, C_m$. Without loss of generality (by inserting one variable as necessary), we may assume that n is even. Call the variables $x_1, x_2, \ldots, x_{n/2}$ *low* variables and the variables $x_{n/2+1}, x_{n/2+2}, \ldots, x_n$ *high* variables. The algorithm operates in three steps.

In the first step, the algorithm constructs the following OR-circuit. First let us observe that there are $2^{n/2}$ distinct ways to assign truth values (0 or 1) to the low variables. Each of these assignments indexes an input gate to the circuit. Next, for each clause C_i, we construct a subcircuit that takes the OR of all input gates that *do not satisfy* the clause C_i, that is, the input gate indexed by an assignment a to the low variables is in the OR if and only if no literal in C_i is satisfied by a. For each C_i, this subcircuit requires at most $2^{n/2} - 1$ OR-gates. Let us refer to the output gate of this subcircuit as gate C_i. Finally, for each assignment b to the high variables, construct a subcircuit that takes the OR of all gates C_i such that the clause C_i is *not satisfied* by b. Let us refer to the output gate of this subcircuit as gate b. The constructed circuit has $p = 2^{n/2}$ inputs and $g \leq m(2^{n/2} - 1) + 2^{n/2}(m - 1) = O(2^{n/2}m)$ gates. The construction time for the circuit is $O(2^{n/2}mn)$.

In the second step, the algorithm rewrites the constructed OR-circuit using algorithm A as a subroutine into a SUM-circuit in time $O(g^{2-\epsilon})$, that is, in time $O(2^{(1-\epsilon/2)n}m^{2-\epsilon})$. In particular, the number of gates in the SUM-circuit is $G = O(2^{(1-\epsilon/2)n}m^{2-\epsilon})$. For a gate z in the OR-circuit, let us write z' for the corresponding gate in the SUM-circuit.

In the third step, the algorithm assigns the value 1 to each input a' in the SUM-circuit (any other inputs are assigned to 0), and evaluates the SUM-circuit over the integers using $O(2^{(1-\epsilon/2)n}m^{2-\epsilon})$ additions of $O(n)$-bit integers. If there exists a gate b' that evaluates to a value less than $2^{n/2}$, the algorithm outputs "satisfiable"; otherwise the algorithm outputs "unsatisfiable". The running time of the algorithm is $O(2^{(1-\epsilon/2)n}m^{2-\epsilon}n)$.

To see that the algorithm is correct, observe that in the OR-circuit, the input a occurs in the support of b if and only if there is a clause C_i such that neither a nor b satisfies C_i. Equivalently, the assignment (a, b) into the n variables is not satisfying (because it does not satisfy the clause C_i). The rewrite into a SUM-circuit enables us to infer the presence of an a' that does not occur in the support of b' by counting the number of a' that do occur in the support of b'. SUM-gates ensure that each input in the support of b' is counted exactly once. □

Theorem 2 thus demonstrates that unless the strong exponential time hypothesis [11] fails, there is no subquadratic-time algorithm for rewriting arbitrary OR-circuits into SUM-circuits.

4 Finding Small Circuits Using SAT Solvers

We next develop a SAT encoding for deciding whether a given ensemble has a circuit of a given size.

4.1 SAT Encoding

We start by giving a representation of an OR- or SUM-circuit as a binary matrix. This representation then gives us a straightforward way to encode the circuit existence problem as a propositional formula.

Let $(P, \mathcal{Q})$ be an OR- or SUM-ensemble and let C be a circuit of size g that solves $(P, \mathcal{Q})$. For convenience, let us assume that $|P| = p$, $|\mathcal{Q}| = q$ and $P = \{1, 2, \ldots, p\}$. Furthermore, we note that outputs corresponding to sets of size 1 are directly provided by the input gates, and we may thus assume that $\mathcal{Q}$ does not contain sets of size 1. The circuit C can be represented as a $g \times p$ binary matrix M as follows. Fix a topological ordering $z_1, z_2, \ldots, z_g$ of the gates of C such that $z_i = i$ for all i with $1 \leq i \leq p$ (recall that we identify the input gates with elements of P). Each row i of the matrix M now corresponds to the support of the gate z_i so that for all $1 \leq j \leq p$ we have $M_{i,j} = 1$ if j is in the support of z_i and $M_{i,j} = 0$ otherwise. In particular, for all $1 \leq i \leq p$ we have $M_{i,i} = 1$ and $M_{i,j} = 0$ for all $j \neq i$. Figure 3 displays an example.

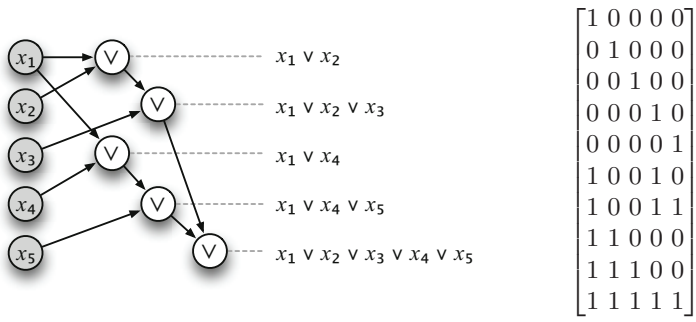


Fig. 3. An OR-circuit (left) and a matrix describing the circuit (right)

Now, C (viewed as an OR-circuit) solves $(P, \mathcal{Q})$ if and only if the matrix M satisfies

(a) for all i with $1 \leq i \leq p$ it holds that $M_{i,i} = 1$ and $M_{i,j} = 0$ for all $j \neq i$,
(b) for all i with $p + 1 \leq i \leq g$ there exist k and ℓ such that $1 \leq k < \ell < i$ and for all j with $1 \leq j \leq p$ it holds that $M_{i,j} = 1$ if and only if $M_{k,j} = 1$ or $M_{\ell,j} = 1$, and
(c) for every set A in $\mathcal{Q}$ there exists an i with $1 \leq i \leq g$ such that for all j with $1 \leq j \leq p$ it holds that $M_{i,j} = 1$ if $j \in A$ and $M_{i,j} = 0$ otherwise.

Similarly, C (viewed as a SUM-circuit) solves $(P, \mathcal{Q})$ if and only if the matrix M satisfies conditions (a), (c), and

(b') for all i with $p + 1 \leq i \leq g$ there exist k and ℓ such that $1 \leq k < \ell < i$ and for all j with $1 \leq j \leq p$ it holds that $M_{i,j} = 1$ if and only if $M_{k,j} = 1$ or $M_{\ell,j} = 1$ and that $M_{k,j} = 0$ or $M_{\ell,j} = 0$.

Based on the above observations, we encode an ensemble computation instance as SAT instance as follows. Given an OR-ensemble $(P, \mathcal{Q})$ and integer g as input, we construct a propositional logic formula φ over variables $M_{i,j}$, where $1 \leq i \leq g$ and $1 \leq j \leq p$, so that any assignment into variables $M_{i,j}$ satisfying φ gives us a matrix that satisfies conditions (a)–(c). We encode condition (a) as

$$\alpha = \bigwedge_{i=1}^{p} \Big(M_{i,i} \wedge \bigwedge_{j \neq i} \neg M_{i,j} \Big) .$$

Similarly, we encode the conditions (b) and (c), respectively, as

$$\beta = \bigwedge_{i=p+1}^{g} \bigvee_{k=1}^{i-2} \bigvee_{\ell=k+1}^{i-1} \bigwedge_{j=1}^{p} \Big((M_{k,j} \vee M_{\ell,j}) \leftrightarrow M_{i,j} \Big) , \quad \text{and}$$

$$\gamma = \bigwedge_{A \in \mathcal{Q}} \bigvee_{i=p+1}^{g} \Big[\Big(\bigwedge_{j \in A} M_{i,j} \Big) \wedge \Big(\bigwedge_{j \notin A} \neg M_{i,j} \Big) \Big] .$$

The desired formula φ is then $\varphi = \alpha \wedge \beta \wedge \gamma$. For a SUM-ensemble, we replace β with

$$\beta' = \bigwedge_{i=p+1}^{g} \bigvee_{k=1}^{i-2} \bigvee_{\ell=k+1}^{i-1} \bigwedge_{j=1}^{p} \Big(((M_{k,j} \vee M_{\ell,j}) \leftrightarrow M_{i,j}) \wedge (\neg M_{k,j} \vee \neg M_{\ell,j}) \Big) .$$

4.2 Practical Considerations

There are several optimisations that can be used to tune this encoding to speed up SAT solving. The resulting SAT instances have a high number of symmetries, as any circuit can be represented as a matrix using any topological ordering of the gates. This makes especially the unsatisfiable instances difficult to tackle with SAT solver. To alleviate this problem, we constrain the rows i for $p + 1 \leq i \leq g$ appear in lexicographic order, so that any circuit that solves $(P, \mathcal{Q})$ has a unique valid matrix representation. Indeed, we note that the lexicographic ordering of the gate supports (viewed as binary strings) is a topological ordering. We insert this constraint to the SAT encoding as the formula

$$\bigwedge_{i=p+2}^{g} \bigwedge_{k=p+1}^{i-1} \Big[(M_{i,1} \vee \neg M_{k,1}) \wedge \bigwedge_{j_1=2}^{p} \Big(\Big(\bigwedge_{j_2=1}^{j_1-1} (M_{i,j_2} \leftrightarrow M_{k,j_2}) \Big) \rightarrow (M_{i,j_1} \vee \neg M_{k,j_1}) \Big) \Big] .$$

We obtain further speedup by constraining the first t arithmetic gates to have small supports. Indeed, the ith arithmetic gate in any topological order has weight at most $i + 1$. Thus, we fix $t = 6$ in the experiments and insert the formula

$$\bigwedge_{i=1}^{t} \bigwedge_{\substack{S \subseteq P \\ |S|=i+2}} \neg \Big(\bigwedge_{j \in S} M_{p+i,j} \Big) .$$

Further tuning is possible if $\mathcal{Q}$ is an *antichain*, that is, if there are no distinct $A, B \in \mathcal{Q}$ with $A \subseteq B$. In this case an optimal circuit C has the property that every gate whose

support is in $\mathcal{Q}$ has out-degree 0. Thus, provided that we do not use the lexicographical ordering of gates as above, we may assume that the gates corresponding to sets in $\mathcal{Q}$ are the last gates in the circuit, and moreover, their respective order is any fixed order. Thus, if $\mathcal{Q} = \{A_1, A_2, \ldots, A_q\}$ is an antichain, we can replace γ with

$$\bigwedge_{i=1}^{q} \Big[\Big(\bigwedge_{j \in A_j} M_{g-q+i,j} \Big) \wedge \Big(\bigwedge_{j \notin A_j} \neg M_{g-q+i,j} \Big) \Big]$$

to obtain a smaller formula. Finally, we note that we can be combine this with the lexicographic ordering by requiring that only rows i for $p + 1 \leq i \leq g - q$ are in lexicographic order.

5 Experiments

We report on two series of experiments with the developed encoding and state-of-the-art SAT solvers: (a) an exhaustive study of small ensembles aimed at understanding the separation between OR-circuits and SUM-circuits, and (b) a study of the scalability of our encoding by benchmarking different solvers on specific structured ensembles.

5.1 Instance Generation and Experimental Setup

For both series of experiments, the problem instances given to SAT solvers were generated by translating the encoding in Sect. 4 into CNF. We used the symmetry breaking constraints and antichain optimisations described in Sect. 4.2; without these, most instances could not solved by any of the solvers.

The formula encoding an input ensemble $(P, \mathcal{Q})$ and a target number of gates g was first translated into a Boolean circuit and then into CNF using the `bc2cnf` encoder (`http://users.ics.tkk.fi/tjunttil/circuits/`), which implements the standard Tseitin encoding [17]. The instance generator and a set of interesting handpicked CNF-level benchmark instances are available at

`http://cs.helsinki.fi/u/jazkorho/sat2012/`.

When working with an ensemble, the size of the optimal OR-circuit or optimal SUM-circuit is not generally known. Thus, we structured the experiments for a given ensemble $(P, \mathcal{Q})$ with $|P| = p$ and $|\mathcal{Q}| = q$ as a sequence of jobs that keeps the ensemble $(P, \mathcal{Q})$ fixed and varies the target number of gates g. We start from a value of g for which a circuit is known to exist ($p(1 + q)$) and then decrease the value in steps of 1 until we hit an unsatisfiable instance at $g = u$; an optimal circuit then has $g = u + 1$ gates.

The experiments were run on Dell PowerEdge M610 blade servers with two quad-core 2.53-GHz Intel Xeon processors and 32 GB of memory. We report the user times recorded via `time` under Linux (kernel version 2.6.38). In the timed benchmarking runs we ran one simultaneous job on a single server, but in the explorative experiments we ran multiple jobs per server in parallel. SAT solvers used were Minisat 2.2.0 [5] and Lingeling 587f [1] (two CDCL solvers among the best for application instances), Clasp 2.0.4 [9] (CDCL solver, one of the best for crafted instances), and March_rw [10] (a DPLL-lookahead solver, one of the best for unsatisfiable random instances).

5.2 Optimal Circuits for All Small Ensembles

We say that two ensembles $(P, \mathcal{Q}_1)$ and $(P, \mathcal{Q}_2)$ are *isomorphic* if there is a permutation of P that takes $\mathcal{Q}_1$ to $\mathcal{Q}_2$. The optimal circuit size is clearly an isomorphism invariant of an ensemble, implying that in an exhaustive study it suffices to consider one ensemble from each isomorphism class.

We carried out an exhaustive study of all nonisomorphic ensembles $(P, \mathcal{Q})$ across the three parameter ranges (i) $p = 5$ and $2 \leq q \leq 7$, (ii) $p = 6$ and $2 \leq q \leq 7$, and (iii) $p = 7$ and $2 \leq q \leq 6$ subject to the following additional constraints: (a) every set in $\mathcal{Q}$ has size at least 2, (b) every set in $\mathcal{Q}$ contains at least two points in P that each occur in at least two sets in $\mathcal{Q}$, and (c) the ensemble is connected (when viewed as a hypergraph with vertex set P and edge set $\mathcal{Q}$). We generated the ensembles using the `genbg` tool that is part of the canonical labelling package `nauty` [15].

For all of the generated 1,434,897 nonisomorphic ensembles, we successfully determined the optimum OR-circuit size and the optimum SUM-circuit size in approximately 4 months of total CPU time using Minisat. Among the instances considered, we found no instance where the gap between the two optima is more than one gate. The smallest ensembles in terms of the parameters p and q where we observed a gap of one gate occurred for $p = 5$ and $q = 5$, for exactly 3 nonisomorphic ensembles; one of the ensembles with accompanying optimal circuits is displayed in Fig. 2. A further analysis of the results led to Theorem 1 and Proposition 1.

After this work the next open parameters for exhaustive study are $p = 7$ and $q = 7$ with 13,180,128 nonisomorphic ensembles.

In general, the large number of isomorphism classes for larger p and q makes an exhaustive search prohibitively time-consuming. A natural idea would be to randomly sample ensembles with given parameters to find an ensemble witnessing a large separation between optimal OR- and SUM-circuits. However, as highlighted in Remark 1, most ensembles require both a large OR-circuit and a large SUM-circuit, suggesting that random sampling would mostly give instances with small difference between optimal OR- and SUM-circuits. This intuition was experimentally supported as follows. We generated random ensembles $(P, \mathcal{Q})$ by setting $P = \{1, 2, \ldots, p\}$ and drawing uniformly at random a $\mathcal{Q}$ consisting of q subsets of P of size at least 2. We generated 1,000 instances for $p = q = 9$ and for $p = q = 10$. Among these instances, we found only one instance (with $p = q = 10$) where the gap between the optimal OR-circuit and and the optimal SUM-circuit was 2, while we know that instances with larger separation do exist for these parameters. However, there were 49 instances with $p = q = 10$ where the optimal circuit sizes were not found within a 6-hour time limit.

5.3 Scaling on Structured Ensembles

To test the scalability of our encoding and to benchmark different solvers, we also studied two parameterised families of structured ensembles for varying family parameters and target number of gates g. The first family is illustrated by the Valiant's construction in Fig. 1 for $p = 8$. This family is parameterised by the number of inputs p, with $P = \{1, 2, \ldots, p\}$ and $\mathcal{Q} = \{P \setminus \{i\} \colon i \in P\}$. As benchmarks we generated CNF instances for $p = 8, 9, 10, 11$ and $g = 2p, 2p + 1, \ldots, 2p + 20$ using the SUM-encoding

and the antichain optimisation. The second family is given in Theorem 1 and is parameterised by two parameters h and w. As benchmarks we generated CNF instances for $h = 3$ and $w = 5$ and $g = 32, 33, \ldots, 52$ using both the OR-encoding and the SUM-encoding.

The results for the two benchmark families are reported in Figs. 4 and 5. The solver March_rw was omitted from the second benchmark due to its poor performance on the first benchmark family. In an attempt to facilitate finding upper bounds for even larger instances, we also tested the local search solver SATTIME2011, which performed notably well on satisfiable crafted instances in the 2011 SAT Competition. However, in our experiments on instances from the satisfiable regime, SATTIME2011 was unable to find the solution within the 3600-second time limit already for the ensembles in Fig. 4 with $p = 8$ and $g = 26, 27, 28$.

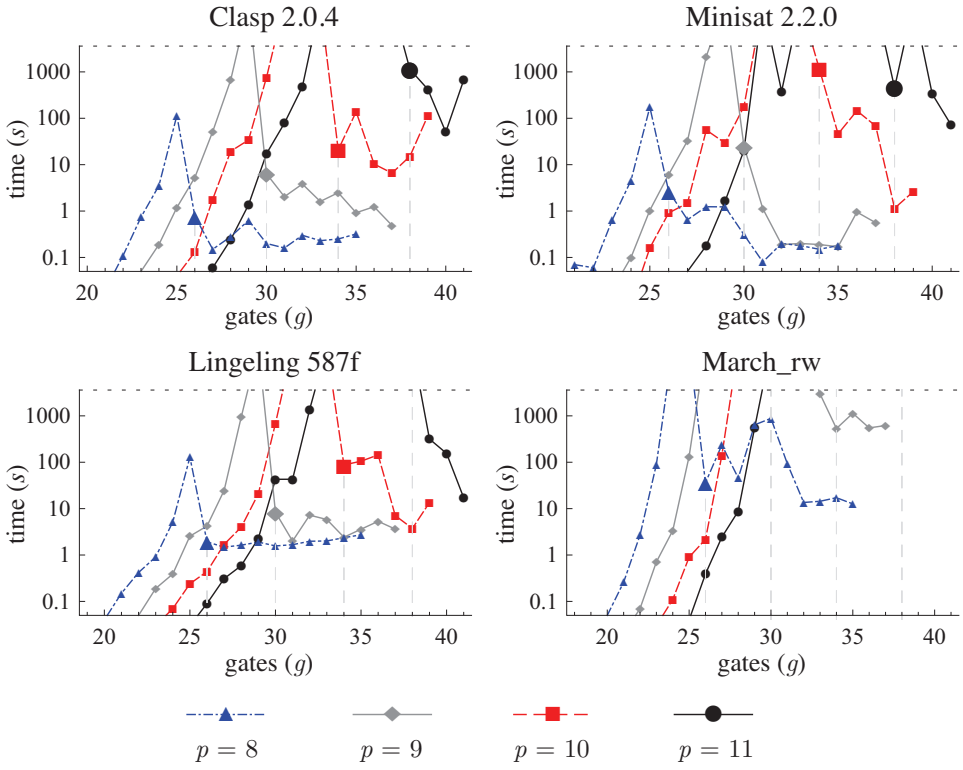


Fig. 4. Solution times for different SAT solvers as a function of the number of gates on SUM-ensembles corresponding to Valiant's construction (Fig. 1). The data points highlighted with larger markers and a vertical dashed line indicate the smallest circuits found. The horizontal dashed line at 3600 seconds is the timeout limit for each run. As the instance size p grows, the unsatisfiable instances with g just below the size of the optimal circuit rapidly become very difficult to solve.

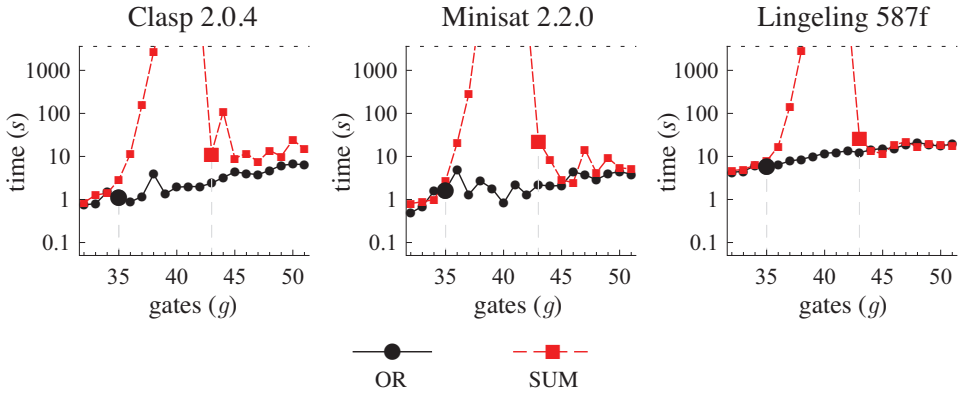


Fig. 5. Solution times for different SAT solvers as a function of the number of gates on OR- and SUM-ensembles from Theorem 1 with parameters $w = 5$ and $h = 3$. The data points highlighted with larger markers and a vertical dashed line indicate the smallest circuits found. The horizontal dashed line at 3600 seconds is the timeout limit for each run. The optimal OR circuit is small, and SAT solvers have no difficulty in finding it.

6 Conclusions

We studied the relative power of OR-circuits and SUM-circuits for ensemble computation, and developed tight connections to Boolean satisfiability from both the theoretical and practical perspectives. As the main theoretical contribution, we showed that, while OR-circuits can be rewritten in quadratic-time into SUM-circuits, a subquadratic-time rewriting algorithm would imply that general CNF-SAT has non-trivial algorithms, which would contradict the strong exponential time hypothesis. From the practical perspective, we developed a SAT encoding for finding smallest SUM- and OR-circuits for a given ensemble. State-of-the-art SAT solvers proved to be a highly useful tool for studying the separation of these two circuit classes. Using the developed encoding, we were able to exhaustively establish the optimum OR-circuit and SUM-circuit sizes for all small instances, which contributed to our analytical understanding of the problem and led to the theoretical results presented in this paper. Our publicly available instance generator may also be of independent interest as a means of generating interesting benchmarks.

Larger, structured instances provide interesting challenges for current state-of-the-art SAT solver technology. Further developments either on the encoding or the solver level—including tuning SAT solvers especially for this problem—would allow for providing further understanding to the problem of separating different circuit classes.

Acknowledgment. We thank Teppo Niinimäki for insight concerning the construction in Theorem 1.

References

1. Biere, A.: Lingeling, Plingeling, PicoSAT and PrecoSAT at SAT Race 2010. FMV Technical Report 10/1, Johannes Kepler University, Linz, Austria (2010)
2. Björklund, A., Husfeldt, T., Kaski, P., Koivisto, M., Nederlof, J., Parviainen, P.: Fast zeta transforms for lattices with few irreducibles. In: Proc. SODA, pp. 1436–1444. SIAM (2012)
3. Dantsin, E., Wolpert, A.: On Moderately Exponential Time for SAT. In: Strichman, O., Szeider, S. (eds.) SAT 2010. LNCS, vol. 6175, pp. 313–325. Springer, Heidelberg (2010)
4. Demenkov, E., Kojevnikov, A., Kulikov, A.S., Yaroslavtsev, G.: New upper bounds on the Boolean circuit complexity of symmetric functions. Inf. Process. Lett. 110(7), 264–267 (2010)
5. Eén, N., Sörensson, N.: An Extensible SAT-solver. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, pp. 502–518. Springer, Heidelberg (2004)
6. Estrada, G.G.: A Note on Designing Logical Circuits Using SAT. In: Tyrrell, A.M., Haddow, P.C., Torresen, J. (eds.) ICES 2003. LNCS, vol. 2606, pp. 410–421. Springer, Heidelberg (2003)
7. Fuhs, C., Schneider-Kamp, P.: Synthesizing Shortest Linear Straight-Line Programs over GF(2) Using SAT. In: Strichman, O., Szeider, S. (eds.) SAT 2010. LNCS, vol. 6175, pp. 71–84. Springer, Heidelberg (2010)
8. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman and Company (1979)
9. Gebser, M., Kaufmann, B., Schaub, T.: The Conflict-Driven Answer Set Solver *clasp*: Progress Report. In: Erdem, E., Lin, F., Schaub, T. (eds.) LPNMR 2009. LNCS, vol. 5753, pp. 509–514. Springer, Heidelberg (2009)
10. Heule, M., Dufour, M., van Zwieten, J.E., van Maaren, H.: March_eq: Implementing Additional Reasoning into an Efficient Look-Ahead SAT Solver. In: Hoos, H.H., Mitchell, D.G. (eds.) SAT 2004. LNCS, vol. 3542, pp. 345–359. Springer, Heidelberg (2005)
11. Impagliazzo, R., Paturi, R.: On the complexity of k-SAT. J. Comput. System Sci. 62(2), 367–375 (2001)
12. Kamath, A.P., Karmarkar, N.K., Ramakrishnan, K.G., Resende, M.G.C.: An interior point approach to Boolean vector function synthesis. In: Proc. MWSCAS, pp. 185–189. IEEE (1993)
13. Kojevnikov, A., Kulikov, A.S., Yaroslavtsev, G.: Finding Efficient Circuits Using SAT-Solvers. In: Kullmann, O. (ed.) SAT 2009. LNCS, vol. 5584, pp. 32–44. Springer, Heidelberg (2009)
14. Lokshtanov, D., Marx, D., Saurabh, S.: Known algorithms on graphs of bounded treewidth are probably optimal. In: Proc. SODA, pp. 777–789. SIAM (2010)
15. McKay, B.: nauty user's guide. Tech. Rep. TR-CS-90-02, Australian National University, Department of Computer Science (1990)
16. Pătraşcu, M., Williams, R.: On the possibility of faster SAT algorithms. In: Proc. SODA, pp. 1065–1075. SIAM (2010)
17. Tseitin, G.S.: On the complexity of derivation in propositional calculus. In: Automation of Reasoning 2: Classical Papers on Computational Logic 1967-1970, pp. 466–483. Springer, Heidelberg (1983)
18. Valiant, L.G.: Negation is powerless for Boolean slice functions. SIAM J. Comput. 15(2), 531–535 (1986)
19. West, D.B.: Introduction to graph theory. Prentice Hall Inc., Upper Saddle River (1996)
20. Williams, R.: A new algorithm for optimal 2-constraint satisfaction and its implications. Theoret. Comput. Sci. 348(2-3), 357–365 (2005)
21. Williams, R.: Improving exhaustive search implies superpolynomial lower bounds. In: Proc. STOC, pp. 231–240. ACM (2010)

TIETOJENKÄSITTELYTIETEEN LAITOS
PL 68 (Gustaf Hällströmin katu 2 b)
00014 Helsingin yliopisto

DEPARTMENT OF COMPUTER SCIENCE
P.O. Box 68 (Gustaf Hällströmin katu 2 b)
FI-00014 University of Helsinki, FINLAND

JULKAISUSARJA **A**

SERIES OF PUBLICATIONS **A**

Reports are available on the e-thesis site of the University of Helsinki.

A-2008-1 I. Autio: Modeling Efficient Classification as a Process of Confidence Assessment and Delegation. 212 pp. (Ph.D. Thesis)

A-2008-2 J. Kangasharju: XML Messaging for Mobile Devices. 24+255 pp. (Ph.D. Thesis).

A-2008-3 N. Haiminen: Mining Sequential Data – in Search of Segmental Structures. 60+78 pp. (Ph.D. Thesis)

A-2008-4 J. Korhonen: IP Mobility in Wireless Operator Networks. 186 pp. (Ph.D. Thesis)

A-2008-5 J.T. Lindgren: Learning nonlinear visual processing from natural images. 100+64 pp. (Ph.D. Thesis)

A-2009-1 K. Hätönen: Data mining for telecommunications network log analysis. 153 pp. (Ph.D. Thesis)

A-2009-2 T. Silander: The Most Probable Bayesian Network and Beyond. 50+59 pp. (Ph.D. Thesis)

A-2009-3 K. Laasonen: Mining Cell Transition Data. 148 pp. (Ph.D. Thesis)

A-2009-4 P. Miettinen: Matrix Decomposition Methods for Data Mining: Computational Complexity and Algorithms. 164+6 pp. (Ph.D. Thesis)

A-2009-5 J. Suomela: Optimisation Problems in Wireless Sensor Networks: Local Algorithms and Local Graphs. 106+96 pp. (Ph.D. Thesis)

A-2009-6 U. Köster: A Probabilistic Approach to the Primary Visual Cortex. 168 pp. (Ph.D. Thesis)

A-2009-7 P. Nurmi: Identifying Meaningful Places. 83 pp. (Ph.D. Thesis)

A-2009-8 J. Makkonen: Semantic Classes in Topic Detection and Tracking. 155 pp. (Ph.D. Thesis)

A-2009-9 P. Rastas: Computational Techniques for Haplotype Inference and for Local Alignment Significance. 64+50 pp. (Ph.D. Thesis)

A-2009-10 T. Mononen: Computing the Stochastic Complexity of Simple Probabilistic Graphical Models. 60+46 pp. (Ph.D. Thesis)

A-2009-11 P. Kontkanen: Computationally Effcient Methods for MDL-Optimal Density Estimation and Data Clustering. 75+64 pp. (Ph.D. Thesis)

A-2010-1 M. Lukk: Construction of a global map of human gene expression - the process, tools and analysis. 120 pp. (Ph.D. Thesis)

A-2010-2 W. Hämäläinen: Efficient search for statistically significant dependency rules in binary data. 163 pp. (Ph.D. Thesis)

A-2010-3 J. Kollin: Computational Methods for Detecting Large-Scale Chromosome Rearrangements in SNP Data. 197 pp. (Ph.D. Thesis)

A-2010-4 E. Pitkänen: Computational Methods for Reconstruction and Analysis of Genome-Scale Metabolic Networks. 115+88 pp. (Ph.D. Thesis)

A-2010-5 A. Lukyanenko: Multi-User Resource-Sharing Problem for the Internet. 168 pp. (Ph.D. Thesis)

A-2010-6 L. Daniel: Cross-layer Assisted TCP Algorithms for Vertical Handoff. 84+72 pp. (Ph.D. Thesis)

A-2011-1 A. Tripathi: Data Fusion and Matching by Maximizing Statistical Dependencies. 89+109 pp. (Ph.D. Thesis)

A-2011-2 E. Junttila: Patterns in Permuted Binary Matrices. 155 pp. (Ph.D. Thesis)

A-2011-3 P. Hintsanen: Simulation and Graph Mining Tools for Improving Gene Mapping Efficiency. 136 pp. (Ph.D. Thesis)

A-2011-4 M. Ikonen: Lean Thinking in Software Development: Impacts of Kanban on Projects. 104+90 pp. (Ph.D. Thesis)

A-2012-1 P. Parviainen: Algorithms for Exact Structure Discovery in Bayesian Networks. 132 pp. (Ph.D. Thesis)

A-2012-2 J. Wessman: Mixture Model Clustering in the Analysis of Complex Diseases. 118 pp. (Ph.D. Thesis)

A-2012-3 P. Pöyhönen: Access Selection Methods in Cooperative Multi-operator Environments to Improve End-user and Operator Satisfaction. 211 pp. (Ph.D. Thesis)

A-2012-4 S. Ruohomaa: The Effect of Reputation on Trust Decisions in Inter-enterprise Collaborations. 214+44 pp. (Ph.D. Thesis)

A-2012-5 J. Sirén: Compressed Full-Text Indexes for Highly Repetitive Collections. 97+63 pp. (Ph.D. Thesis)

A-2012-6 F. Zhou: Methods for Network Abstraction. 48+71 pp. (Ph.D. Thesis)

A-2012-7 N. Välimäki: Applications of Compressed Data Structures on Sequences and Structured Data. 73+94 pp. (Ph.D. Thesis)

A-2012-8 S. Varjonen: Secure Connectivity With Persistent Identities. 139 pp. (Ph.D. Thesis)

A-2012-9 M. Heinonen: Computational Methods for Small Molecules. 110+68 pp. (Ph.D. Thesis)

A-2013-1 M. Timonen: Term Weighting in Short Documents for Document Categorization, Keyword Extraction and Query Expansion. 53+62 pp. (Ph.D. Thesis)

A-2013-2 H. Wettig: Probabilistic, Information-Theoretic Models for Etymological Alignment. 130+62 pp. (Ph.D. Thesis)

A-2013-3 T. Ruokolainen: A Model-Driven Approach to Service Ecosystem Engineering. 232 pp. (Ph.D. Thesis)

A-2013-4 A. Hyttinen: Discovering Causal Relations in the Presence of Latent Confounders. 107+138 pp. (Ph.D. Thesis)

A-2013-5 S. Eloranta: Dynamic Aspects of Knowledge Bases. 123 pp. (Ph.D. Thesis)

A-2013-6 M. Apiola: Creativity-Supporting Learning Environments: Two Case Studies on Teaching Programming. 62+83 pp. (Ph.D. Thesis)

A-2013-7 T. Polishchuk: Enabling Multipath and Multicast Data Transmission in Legacy and Future Interenet. 72+51 pp. (Ph.D. Thesis)

A-2013-8 P. Luosto: Normalized Maximum Likelihood Methods for Clustering and Density Estimation. 67+67 pp. (Ph.D. Thesis)

A-2013-9 L. Eronen: Computational Methods for Augmenting Association-based Gene Mapping. 84+93 pp. (Ph.D. Thesis)

A-2013-10 D. Entner: Causal Structure Learning and Effect Identification in Linear Non-Gaussian Models and Beyond. 79+113 pp. (Ph.D. Thesis)

A-2013-11 E. Galbrun: Methods for Redescription Mining. 72+77 pp. (Ph.D. Thesis)

A-2013-12 M. Pervilä: Data Center Energy Retrofits. 52+46 pp. (Ph.D. Thesis)

A-2013-13 P. Pohjalainen: Self-Organizing Software Architectures. 114+71 pp. (Ph.D. Thesis)