LUND UNIVERSITY

Studies in Efficient Discrete Algorithms

SLEDNEU, DZMITRY

2016

Document Version:
Publisher's PDF, also known as Version of record

Link to publication

Citation for published version (APA):
SLEDNEU, DZMITRY. (2016). *Studies in Efficient Discrete Algorithms.* [Doctoral Thesis (compilation), Mathematics (Faculty of Sciences)]. Lund University, Faculty of Science, Centre for Mathematical Sciences, Mathematics.

Total number of authors:
1

General rights
Unless other specific re-use rights are stated the following general rights apply:
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.
• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal

Read more about Creative commons licenses: https://creativecommons.org/licenses/

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

LUND UNIVERSITY

PO Box 117
221 00 Lund
+46 46-222 00 00

Download date: 08. Sep. 2026

Studies in Efficient Discrete Algorithms

Dzmitry Sledneu

Lund University

Faculty of Science
Centre for Mathematical Sciences
Mathematics

Mathematics
Centre for Mathematical Sciences
Lund University
Box 118
SE-221 00 Lund
Sweden

www.maths.lu.se

Doctoral Theses in Mathematical Sciences 2016:4
ISSN 1404-0034

LUNFMA-1037-2016

ISBN 978-91-7623-866-0 (paperback)
ISBN 978-91-7623-867-7 (PDF)

© 2016 Dzmitry Sledneu

Contents

Preface **v**
List of Papers v
Acknowledgements vi

Introduction **1**
Algorithms and Complexity 1
Graphs 4
Paper 1: All-Pairs Shortest Paths 5
Paper 2: Optimal Partitions 5
Paper 3: Geometric Matrix Multiplication 6
Paper 4: Conuting Trinangulations 6
Paper 5: Detecting Monomials with k Distinct Variables 7

1 All-Pairs Shortest Paths **9**
1.1 Introduction 9
1.2 A Reduction of APSP to Mixed Matrix Products 12
1.3 Fast Computation of the Mixed Products for Clustered Data 14
1.4 Main Results 17
1.5 APSP in Vertex-Weighted Uniform Disk Graphs 18
1.6 Final Remarks 20

2 Optimal Partitions **23**
2.1 Introduction 23
2.2 Preliminaries 24
2.3 The Algorithm 25
2.4 Extensions to Graphs of Bounded Treewidth 29
2.5 Final Remarks 30

3 Geometric Matrix Multiplication **31**
3.1 Introduction 31
3.2 3D Histograms and Their Rectangular Partitions 33
3.3 Geometric Algorithms for the Arithmetic Matrix Product 39
3.4 Final Remarks 49

4 Counting Triangulations **51**
4.1 Introduction 51
4.2 Preliminaries 54
4.3 An Abstract Crossing-Free Structure 56
4.4 Dynamic Programming 57
4.5 Approximation Factor 60
4.6 Main Results 65
4.7 Final Remarks 66

5 Detecting Monomials with k Distinct Variables **67**
5.1 Introduction 67
5.2 Preliminaries 69
5.3 FPT Algorithms 69
5.4 Hardness and Inapproximability Results 72
5.5 Extensions to the Boolean Case 75

Bibliography **84**

Preface

List of Papers

This thesis is based on the following five papers:

1. Andrzej Lingas and Dzmitry Sledneu. A combinatorial algorithm for all-pairs shortest paths in directed vertex-weighted graphs with applications to disc graphs. In Mária Bieliková, Gerhard Friedrich, Georg Gottlob, Stefan Katzenbeisser, and György Turán, editors, *SOFSEM 2012: Theory and Practice of Computer Science — 38th Conference on Current Trends in Theory and Practice of Computer Science, Špindlerův Mlýn, Czech Republic, January 21–27, 2012. Proceedings*, volume 7147 of *Lecture Notes in Computer Science*, pages 373–384. Springer, 2012.

2. Marek Karpinski, Andrzej Lingas, and Dzmitry Sledneu. Optimal cuts and partitions in tree metrics in polynomial time. *Information Processing Letters*, 113(12):447–451, 2013.

3. Peter Floderus, Jesper Jansson, Christos Levcopoulos, Andrzej Lingas, and Dzmitry Sledneu. 3D rectangulations and geometric matrix multiplication. In Hee-Kap Ahn and Chan-Su Shin, editors, *Algorithms and Computation — 25th International Symposium, ISAAC 2014, Jeonju, Korea, December 15–17, 2014, Proceedings*, volume 8889 of *Lecture Notes in Computer Science*, pages 65–78. Springer, 2014.

   The version included is an extended journal version.

4. Marek Karpinski, Andrzej Lingas, and Dzmitry Sledneu. A QPTAS for the base of the number of crossing-free structures on a planar point set. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors,

Automata, Languages, and Programming — 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6–10, 2015, Proceedings, Part I, volume 9134 of *Lecture Notes in Computer Science*, pages 785–796. Springer, 2015.

The version included is an extended journal version.

5. Peter Floderus, Andrzej Lingas, Mia Persson, and Dzmitry Sledneu. Detecting monomials with k distinct variables. *Information Processing Letters*, 115(2):82–86, 2015.

Acknowledgements

I would like to express my deepest grattitude to my supervisor Andrzej Lingas. I am greatly indebted for his guidance, assistance, help, encouragement, patience, and so much more.

I am also very thankful to my coauthors Marek Karpinski, Peter Floderus, Jesper Jansson, Christos Levcopoulos, Mia Persson, Nadia Brauner, Gerd Finke, and Yakov Shafransky.

Introduction

Algorithms and Complexity

This thesis consists of five papers within the design and analysis of efficient algorithms. Algorithms lie at the heart of computing. They are vitally important for the modern society. Before describing the contents of the papers we introduce (somewhat informally) some basic notions and concepts.

According to [20] an *algorithm* is any well-defined computational procedure that takes some value, or set of values, as *input* and produces some value, or set of values, as *output*, using a sequence of computational steps.

An algorithm is applied to a well-specified *computational problem*. An example of such a computational problem is to find the largest number in a given list of numbers. Here the input for this problem is a list of numbers and the desired output is a number from this list, that is not less than any other number on the list.

Given such a computational problem, any input that satisfies the problem's description is called an *instance* of the problem. For example, the list $(5, 3, 7, 1, 10, 2)$ is an instance of the aforementioned problem, and the required output for this instance is 10.

An algorithm is said to be *correct* [20] if, for every input instance of a problem, it halts with the correct output. In this case the algorithm *solves* the given computational problem. An incorrect algorithm either never halts for some problem instances, or returns an incorrect answer for some of them.

If the operations performed by an algorithm are always determined solely by the input instance of a problem, such an algorithm is called *deterministic*. If the operations performed by an algorithm are determined by the input instance of a problem with the help of a random bits, such an algorithm is called *randomized*.

An algorithm can be specified in many different ways with different levels of detail. For example, it can be an English text, or code written in some computer programming

language such as C++, or hardware chip design. Using such a specification the algorithm can be *executed* by whatever means possible. For example, by a person following the text description, or by a CPU running the software built from the code, or by a hardware chip created from the chip design document. In this thesis algorithms are expressed using a *pseudocode* which is somewhat less strict than programming language code, but still more formal than a plain English text.

Since a computational problem can have infinitely many instances, in order to show that a given algorithm solves it, one needs to prove (usually formally) that the algorithm halts for all possible input instances of the problem and returns the correct output. Not all problems admit an algorithm that halts for all possible input instances of the problem (e. g., halting problem [20]).

Having several algorithms that solve the same computational problem, the natural question arises: which algorithm is the *best* one? Also, what amount of *resources* this algorithm requires? In order to answer these questions the *analysis of algorithms* has been developed. Often the *running time* is used to compare algorithms (*space complexity* and *input/output complexity* or *I/O complexity* are also studied, but we won't use these concepts in the thesis).

In order to analyze the running time, the *computational model* should be chosen. In this thesis we use the basic *random-access machine (RAM)* model of computation.

Given a particular computational model, the *running time* of an algorithm on a particular input is the number of primitive operations (steps) executed [20]. It should be noted, that for different inputs (instances) the running time of the same algorithm can be quite different. It is quite easy to find the largest number in the list of ten numbers, but a bit more difficult if the list has billion numbers.

The standard approach here is to analyze the dependency of the running time on the *input size* parameter, which is some measure of the input instance of a problem and depends on a computational problem being studied. For example, for the problem of finding the largest number in a list of 32-bit integer numbers, the input size might be the length of the list. Running an algorithm on different instances of the problem with the same input size can result in different running times: *best-case*, *average-case*, and *worst-case* running time. In this thesis, we are interested in the worst-case running time, i. e., the longest running time among all possible input instances of the same size.

The running time $T(n)$ of an algorithm is a function of an input size n. Instead of the closed-form expression one uses the following *asymptotic notation*.

$T(n) = O(f(n))$ means that the function $f(n)$ is *an asymptotic upper bound* for the function $T(n)$, or more formally: $T(n) \in \{g(n) \mid \exists c > 0, n_0 > 0, \text{ s. t. } 0 \leq g(n) \leq cf(n), \forall n \geq n_0\}$. For example, $T(n) = \frac{3}{4}n^2 - 2n + 17 = O(n^3)$.

$T(n) = \Theta(f(n))$ means that the function $f(n)$ is *an asymptotically tight bound*

for the function $T(n)$, or more formally: $T(n) \in \{g(n) \mid \exists c_1 > 0, c_2 > 0, n_0 > 0, \text{ s. t. } 0 \leq c_1 f(n) \leq g(n) < c_2 f(n), \forall n \geq n_0\}$. For example, $T(n) = \frac{3}{4}n^2 - 2n + 17 = \Theta(n^2)$.

$T(n) = \Omega(f(n))$ means that the function $f(n)$ is *an asymptotic lower bound* for the function $T(n)$, or more formally: $T(n) \in \{g(n) \mid \forall c > 0 \exists n_0 > 0, \text{ s. t. } 0 \leq cf(n) \leq g(n), \forall n \geq n_0\}$. For example, $T(n) = \frac{3}{4}n^2 - 2n + 17 = \Omega(n)$.

If the running time of an algorithm is $T(n) = \Theta(1)$, such an algorithm is called a *constant-time* algorithm.

If the running time of an algorithm is $T(n) = \Theta(n)$, such an algorithm is called a *linear-time* algorithm.

If the running time of an algorithm is $T(n) = \Theta\left(n^2\right)$, such an algorithm is called a *quadratic-time* algorithm.

In general, if the running time of an algorithm is $T(n) = \Theta(n^k)$ for some constant $k > 0$, such an algorithm is called a *polynomial-time* algorithm, and if the running time of an algorithm is $T(n) = \Theta(2^{\varepsilon n})$ for some constant $\varepsilon > 0$, such an algorithm is called an *exponential-time* algorithm.

Some computational problems can be solved by polynomial-time algorithms, while for others there are only exponential-time algorithms known. Finally, some computational problems cannot be solved by any algorithm at all (e. g., halting problem [20]).

Computational problems that require yes or no as an answer are called *decision problems*, and computational problems that require the maximum or the minimum of some *objective function* of the input as an answer are called *optimization problems*. For example, the problem of determining if a given number is the largest number in a given list, is a decision problem, and the problem of finding the largest number in a given list is an optimization version of the former decision problem.

Complexity classes group computational problems according to the complexity of algorithms known to solve them. Decision problems that admit a polynomial-time algorithm form the class P (polynomial-time). All decision problems for which the proposed solution can be verified in polynomial time in the size of the input form the class NP (nondeterministic polynomial-time).[1]

In order to better understand the possibility of having an efficient algorithm for the problems from NP, a notion of *NP-complete problem* has been introduced. Informally speaking, an NP-complete problem is a problem in NP that is as hard as any other problem in NP [20]. There is no polynomial algorithm known for any NP-complete problem, and it is conjectured that $P \neq NP$.

When a decision version of an optimization problem is NP-complete and the

[1] For problems in NP one can nondeterministically guess the solution and then verify that it is indeed a solution in polynomial time, hence the name "nondeterministic polynomial-time".

optimization problem cannot be solved *exactly* in an efficient way, it is natural to look for an efficient *approximation algorithm*, which finds a *near-optimal* solution to the optimization problem in polynomial time.

In order to measure the quality of an approximate solution, an *approximation ratio* is used. An algorithm has an approximation ratio $\rho(n)$ [20] if, for any input size n, the cost C of the solution produced by the algorithm is within a factor of $\rho(n)$ of the cost C^* of an optimal solution, i. e., $\max\left(\frac{C}{C^*}, \frac{C^*}{C}\right) \leq \rho(n)$, where $C, C^* > 0$.

An *approximation scheme* is an approximation algorithm that takes an instance of the problem, a parameter $\varepsilon > 0$ and returns an approximate solution with the approximation ratio $1 + \varepsilon$. The running time of such an algorithm depends both on the size of the problem instance and on the parameter ε. If for any fixed $\varepsilon > 0$ the running time is polynomial in the size of the input, such an approximation scheme is called a *polynomial-time approximation scheme (PTAS)*. If the running time of a polynomial-time approximation scheme is also polynomial in $1/\varepsilon$, such a scheme is called a *fully polynomial-time approximation scheme (FPTAS)*. It should be noted that not all optimization problems whose decision version is in *NP* admit PTAS, unless $P = NP$ (for example, the minimum vertex cover problem [20]).

Graphs

Most of the problems discussed in this thesis deal with *undirected or directed graphs*.

An undirected graph is a mathematical object that is formally defined as an ordered pair (V, E) where V is a finite set of *vertices* and $E \subseteq \{\{v, u\} \mid v \in V, u \in V\}$ is a set of unordered pairs of vertices called *edges* [20]. A directed graph is defined similarly as an ordered pair (V, E) where V is a finite set of vertices and $E \subseteq V \times V$ is a set of ordered pairs of vertices called edges. In both cases edges or vertices or both can have weights. An undirected graph is *complete*, if every pair of distinct vertices forms an edge. A graph $G' = (V', E')$ is a *subgraph* of a graph $G = (V, E)$, if $V' \subseteq V$ and $E' \subseteq E$.

A *path* of *length* k from a vertex v to a vertex u in an undirected (directed) graph $G = (V, E)$ [20] is a sequence of vertices $(v_0, v_1, v_2, \ldots, v_k)$, where $v_0 = v, v_k = u$ and $\{v_{i-1}, v_i\} \in E$ ((v_{i-1}, v_i), respectively) for $i = 1, 2, \ldots, k$. The path contains the vertices $v_0, v_1, \ldots, v_k$ and the edges $\{v_0, v_1\}, \{v_1, v_2\}, \ldots, \{v_{k-1}, v_k\}$ ((v_0, v_1), $(v_1, v_2), \ldots, (v_{k-1}, v_k)$, respectively). The length of the path is the number of edges in the path [20]. The path $(v_0, v_1, v_2, \ldots, v_k)$ in an undirected (directed) graph of length $k > 2$ ($k > 1$, respectively) where $v_0 = v_k$ is called a *cycle*. An undirected graph where for every pair of distinct vertices v and u there is a path connecting them is called

connected.

An undirected connected graph without cycles is called a *tree*.

A subgraph of a graph $G = (V, E)$ that is a tree which includes (spans) all the vertices from V is called a *spanning tree* of G. The spanning tree of an edge-weighted graph that has the minimum total weight is called a *minimum weight spanning tree*.

Paper 1: All-Pairs Shortest Paths

We consider the problem of computing all-pairs shortest paths in a directed graph with real weights assigned to vertices. The general all-pairs shortest paths problem in an edge-weighted directed graph can be solved in time $O(n^3/2^{\Omega(\log n)^{1/2}})$ [69]. For a special case of vertex-weighted graphs the all-pairs shortest paths problem can be solved in time $O(n^{2.842})$ [71].

For an $n \times n$ $(0, 1)$-matrix C, let K_C be the complete weighted graph on the rows of C where the weight of an edge between two rows is equal to their Hamming distance (i. e., the number of positions at which the corresponding row elements are different).

Let $MWT(C)$ be the weight of a minimum weight spanning tree of K_C.

We show that the all-pairs shortest path problem for a directed graph G on n vertices with nonnegative real vertex weights and adjacency matrix A_G can be solved by an (combinatorial) randomized algorithm in time

$$\widetilde{O}(n^2\sqrt{n + \min\{MWT(A_G), MWT(A_G^t)\}}).$$

(The notation $\widetilde{O}(\,)$ suppresses polylogarithmic factors and B^t stands for the transposed matrix B.)

As a corollary, we conclude that the transitive closure of a directed graph G can be computed by an (combinatorial) randomized algorithm in the aforementioned time.

We also conclude that the all-pairs shortest path problem for so-called uniform disk graphs, with nonnegative real vertex weights, induced by point sets of bounded density within a unit square can be solved in $\widetilde{O}(n^{2.75})$ time.

Paper 2: Optimal Partitions

A pair of sets $V_1 \subset V, V_2 = V \setminus V_1$ in an undirected graph $G = (V, E)$ is called a *cut*. The weight of the cut (V_1, V_2) is the total weight of the edges crossing the cut (i. e., $\{u, v\} \in E, v \in V_1, u \in V_2$). The Maximum (Minimum) Cut problem is to find a cut in a given edge-weighted graph achieving the maximum (minimum) weight. If

the required sizes of V_1 and V_2 are given as an input, the problem is called Maximum (Minimum) Partition problem. In particular, if $|V| = 2k$ and $|V_1| = |V_2| = k$, the problem is called a Maximum (Minimum) Bisection problem.

We present a polynomial-time dynamic programming algorithm for optimal partitions of a complete edge-weighted graph, where the edges are weighted by the length of the unique shortest path connecting those vertices in the a priori given tree (shortest path metric induced by a tree). This resolves, in particular, the complexity status of the optimal partition problems in one dimensional geometric (Euclidean) setting. We discuss also an extension of our method to the class of shortest path metrics induced by the so-called bounded treewidth graphs.

Paper 3: Geometric Matrix Multiplication

The problem of partitioning an orthogonal polyhedron P into a minimum number of 3D rectangles is known to be NP-hard. In this paper, we first develop an approximation algorithm with the approximation ratio 4 for the special case of the problem in which P is a so-called 3D histogram. It runs in $O(m \log m)$ time, where m is the number of corners in P. We then apply it to compute the exact arithmetic matrix product of two $n \times n$ matrices A and B with nonnegative integer entries. The computation takes time $\tilde{O}(n^2 + \min\{r_A r_B,\ n \min\{r_A,\ r_B\}\})$ where $\tilde{O}$ suppresses polylogarithmic (in n) factors and r_A and r_B denote the minimum number of 3D rectangles into which the 3D histograms induced by A and B can be partitioned, respectively.

Paper 4: Counting Triangulations

The number of triangulations of a planar n point set S is known to be c^n, where the base c lies between 2.43 and 30. Similarly, the number of crossing-free spanning trees on S is known to be d^n, where the base d lies between 6.75 and 141.07. The fastest known algorithm for counting triangulations of S runs in $O^*(2^n)$ time while that for counting crossing-free spanning trees runs in $O^*(7.125^n)$ time. The fastest known arbitrarily close approximation algorithms for the base of the number of triangulations of S and the base of the number of crossing-free spanning trees of S, respectively, run in time subexponential in n. We present the first quasi-polynomial approximation schemes for the base of the number of triangulations of S and the base of the number of crossing-free spanning trees on S, respectively. (A *quasi-polynomial approximation scheme (QPTAS)* is an approximation algorithm that takes an instance of the problem, a parameter $\varepsilon > 0$ and returns an approximate solution with the approximation ratio

$1+\varepsilon$ in time quasi-polynomial in n (i. e., $n^{(\log n)^c}$ for some constant $c > 0$) for any fixed $\varepsilon > 0$.)

Paper 5: Detecting Monomials with k Distinct Variables

We study the complexity of detecting monomials with special properties in the sum-product expansion of a polynomial represented by an arithmetic circuit of size polynomial in the number of input variables and using only multiplication and addition. We focus on monomial properties expressed in terms of the number of distinct variables occurring in a monomial. Our first result is a randomized fixed-parameter tractable (FPT) algorithm for detection of a monomial having at least k distinct variables, parametrized with respect to k. (A *fixed-parameter tractable (FPT)* algorithm, parametrized with respect to some k, is an algorithm that runs in time $f(k)O(n^c)$ for some constant $c > 0$, where n is the size of an input, and f is an arbitrary function of k.) For a more restricted class of circuits, we can also provide a deterministic FPT algorithm for detection of a monomial having at most k distinct variables parametrized by the degree of the polynomial represented by the input circuit. Furthermore, we derive several hardness results on detection of monomials with such properties within exact, parametrized and approximation complexity.

Paper 1

A Combinatorial Algorithm for All-Pairs Shortest Paths in Directed Vertex-Weighted Graphs with Applications to Disc Graphs

1.1 Introduction

The problems of finding shortest paths and determining their lengths are fundamental in algorithms. They have been extensively studied in algorithmic graph theory. A central open question in this area is if there is a substantially subcubic in the number of vertices algorithm for the all-pairs shortest path problem for directed graphs with real edge weights (APSP) in the addition-comparison model [70, 72]. For several special cases of weights and/or graphs substantially subcubic algorithms for the APSP problem are known [5, 15, 61, 71, 72, 73]. However, in the general case the fastest known algorithm due to Chan [15] (see also [14]) runs in time $O(n^3 \log^3 \log n / \log^2 n)$, achieving solely a moderate polylogarithmic improvement over the $O(n^3)$ bound yielded by Floyd-Warshall and Johnson's algorithms [3, 72].

The situation is different for directed graphs with real vertex weights. Recently,

Chan has shown that the APSP problem for the aforementioned graphs can be solved in time $O(n^{2.844})$ [15] and Yuster has slightly improved the latter bound to $O(n^{2.842})$ by using an improved bound on rectangular multiplication [71].

The basic tool in achieving substantially subcubic upper bounds on the running time for the APSP for directed graphs with constrained edge weights or real vertex weights are the fast algorithms for arithmetic square and rectangular matrix multiplication [19, 38]. One typically exploits here the close relationship between the APSP problem and the so called distance or $(\min, +)$ product [5, 61, 70, 71, 72, 73].

Unfortunately, these fast algorithms for matrix multiplication, yielding equally fast algorithms for Boolean matrix product, are based on recursive algebraic approaches over a ring difficult to implement. Thus, another central question in this area is whether or not there is a substantially subcubic *combinatorial* (i.e., not relaying on ring algebra) algorithm for the Boolean product of two $n \times n$ Boolean matrices [10, 58, 70]. Again, the fastest known combinatorial algorithm for Boolean matrix product due to Bansal and Williams [10] running in time $O(n^3 \log^2 \log n / \log^{9/4} n)$ achieves solely a moderate polylogarithmic improvement over the trivial $O(n^3)$ bound. On the other hand, several special cases of Boolean matrix product admit substantially subcubic combinatorial algorithms [11, 34, 51].

In particular, Björklund et al. [11] provided a combinatorial randomized algorithm for Boolean matrix product which is substantially subcubic in case the rows of the first $n \times n$ matrix or the columns of the second one are highly clustered, i.e., their minimum spanning tree in the Hamming metric has low cost. More exactly, their algorithm runs in time $\tilde{O}(n(n+c))$, where c is the minimum of the costs of the minimum spanning trees for the rows and the columns, respectively, in the Hamming metric. It relies on the fast Monte Carlo methods for computing an approximate minimum spanning tree in the L_1 and L_2 metrics given in [40, 41].

The assumption that the input directed graph is highly clustered in the sense that the minimum spanning tree of the rows or columns of its adjacency matrix in the Hamming metric has a subquadratic cost does not yield any direct applications of the algorithm of Björklund et al. [11] to shortest path problems, not even to the transitive closure. The reason is that the cost of the analogous minimum spanning tree can grow dramatically in the power graphs[1] of the input graph. In particular, we cannot obtain directly an upper time-bound on the transitive closure of Boolean matrix corresponding to that for the Boolean matrix product from [11] by applying the asymptotic equality between the time complexity of matrix product over a closed semi-ring and that of its transitive closure over the semi-ring due to Munro [54]. The

[1] In the i-th power graph there is an edge from v to u if there is a path composed of at most i edges from v to u in the input graph.

reason is the dependence of the upper bound from [11] on the cost of the minimum spanning tree.

In this paper, we extend the idea of the method from [11] to include a mixed product of a real matrix with a Boolean one. We combine the aforementioned extension with the ideas used in the design of subcubic algorithms for important variants of the APSP problem [5, 73], in particular those for directed graphs with vertex weights [15, 71], to obtain not only a substantially subcubic combinatorial algorithm for the transitive closure but also for the APSP problem in highly clustered directed graphs with real vertex weights.

For an $n \times n$ $0-1$ matrix C, let K_C be the complete weighted graph on the rows of C where the weight of an edge between two rows is equal to their Hamming distance. Let $MWT(C)$ be the weight of a minimum weight spanning tree of K_C. We show that the all-pairs shortest path problem for a directed graph G on n vertices with non-negative real weights and an adjacency matrix A_G can be solved by a combinatorial randomized algorithm in $\widetilde{O}(n^2\sqrt{n + \min\{MWT(A_G), MWT(A_G^t)\}})$ time. It follows in particular that the transitive closure of a directed graph G can be computed by a combinatorial randomized algorithm in the aforementioned time.

Our algorithms are of Monte Carlo type and by increasing the polylogarithmic factor at the time bounds, the probability that they return a correct output within the bounds can be amplified to $1 - \frac{1}{n^\alpha}$, where $\alpha \geq 1$.

Since there are no practical or combinatorial substantially subcubic-time algorithms not only for the APSP problem but even for the transitive closure problem for arbitrary directed graphs at present, our simple adaptive method might be a potentially interesting alternative for a number of graph classes.

As an example of an application of our method, we consider the APSP problem for vertex-weighted uniform disk graphs induced by point sets of bounded density within a unit square. We obtain a combinatorial algorithm for this problem running in time $O(\sqrt{r}n^{2.75})$, where r is the radius of the disks around the vertices in a unit square.

The recent interest in disk graphs, in particular uniform disk graphs, stems from their applications in wireless networks. In this context, the restriction to point sets of bounded density is quite natural. In [32], Fürer and Kasiviswanathan provided a roughly $O(n^{2.5})$-time preprocessing for *approximate* $O(\sqrt{n})$-time distance queries in arbitrary disk graphs.

Our paper is structured as follows. In the next section, we show a reduction of the APSP problem for directed graphs with real vertex-weights to a mixed matrix product of a distance matrix over reals with the $0-1$ adjacency matrix. In Section 1.3, we present an algorithm for such a mixed product which generalizes that for the Boolean matrix product from [11] and runs in subcubic time if the input $0-1$ matrix is

highly clustered. By combining the results of Sections 1.2, 1.3, we can derive our main results in Section 1.4. In the next section, we present the application of our method to uniform disk graphs induced by point sets of bounded density. We conclude with final remarks.

1.2 A Reduction of APSP to Mixed Matrix Products

1.2.1 The APSP problem

Formally, the All-Pairs Shortest Paths problem (APSP) in a directed graph $G = (V, E)$ with real weights $w(v)$ associated to vertices $v \in V$ is to compute the $|V| \times |V|$ distance matrix D_G such that $D_G(v, u)$ is the distance $\delta_G(v, u)$ from v to u in G, i.e., the minimum total weight of vertices on a path from v to u in G. An additional goal of the APSP problem is to compute a concise data structure representing the shortest paths.

Note that $\delta_G(v, u)$ is equal to the minimum total weight of inner vertices on a path from v to u in G increased by the weights of v and u.

We shall assume $|V| = n$ throughout the paper.

For $i = 0, 1, \ldots, n-1$, let $\delta^i_G(v, u)$ be the distance from v to u on paths consisting of at most i edges, i.e., the minimum total weight of vertices on a path from v to u having at most i edges in G. Next, let D^i_G be the $|V| \times |V|$ matrix such that $D^i_G[v, u]$ is equal to $\delta^i_G(v, u)$.

For convention, we assume $\delta^0_G(v, v) = 0$ and $\delta^0_G(v, u) = +\infty$ for $v \neq u$. Hence, D^0_G has zeros on the diagonal and $+\infty$ otherwise. In D^1_G, all the entries $D^1_G[v, u]$ where $(v, u) \in E$ are set to $w(v) + w(u)$ instead of $+\infty$. Thus, both D^0_G and D^1_G can be easily computed in time $O(n^2)$.

1.2.2 Mixed matrix products

Let A be an $n \times n$ matrix over $R \cup \{+\infty\}$, and let B be an $n \times n$ matrix with entries in $\{0, 1\}$. The *mixed right product* C of A and B is defined by

$$C[i, j] = \min\{A[i, k] | 1 \leq k \leq n \,\&\, B[k, j] = 1\} \cup \{+\infty\}.$$

If $C[i, j] \neq +\infty$ then the index k such that $C[i, j] = A[i, k]$ (and thus $B[k, j] = 1$) is called a witness for $C[i, j]$. Analogously, the *mixed left product* C' of B and A is defined by

$$C'[i, j] = \min\{A[k, j] | 1 \leq k \leq n \,\&\, B[i, k] = 1\} \cup \{+\infty\},$$

and if $C'[i,j] \neq +\infty$ then the index k such that $C'[i,j] = A[k,j]$ is called a witness for $C'[i,j]$.

An $n \times n$ matrix W such that whenever $C[i,j] \neq +\infty$ then $W[i,j]$ is a witnesses for $C[i,j]$ is called a witness matrix for the right mixed product of A and B. Analogously, we define a witness matrix for the left mixed product of B and A.

1.2.3 The reduction

Let A_G denote the $n \times n$ adjacency matrix of $G = (V,E)$, i.e., $A_G[v,u] = 1$ iff $(v,u) \in E$.

Lemma 1.1. *For an arbitrary $i \in \{0,1,\ldots,n-2\}$, D_G^{i+1} can be computed on the basis of D_G^i and the right mixed product of D_G^i with A_G or D_G^i and the left mixed product of A_G with D_G^i in time $O(n^2)$.*

Proof. It is sufficient to observe that for any pair v, u of vertices in G, $D_G^{i+1}[v,u]$ is equal to

$$\min\{D_G^i[v,u], \min\{D_G^i[v,x] + w(u) | 1 \leq x \leq n \;\&\; A_G[x,u] = 1\} \cup \{+\infty\}\}$$

Symmetrically, $D_G^{i+1}[v,u]$ is equal to

$$\min\{D_G^i[v,u], \min\{D_G^i[x,u] + w(v) | 1 \leq x \leq n \;\&\; A_G[v,x] = 1\} \cup \{+\infty\}\}$$

□

The following lemma follows the general strategy used to prove Theorem 3.4 in [15].

Lemma 1.2. *Let G be a directed graph G on n vertices with non-negative real vertex weights. Suppose that the right (or left) mixed product of an $n \times n$ matrix over $R \cup \{+\infty\}$ with the adjacency matrix A_G of G along with the witness matrix can be computed in time $T_{mix}(n) = \Omega(n^2)$. The APSP problem for G can be solved in time $\widetilde{O}(n^{1.5}\sqrt{T_{mix}(n)})$.*

Proof. We begin by computing D_G^{t-1} for some $t \in [2,\ldots,n]$ which will be specified later. By Lemma 1.1 this computation takes time $O(tT_{mix}(n))$.

It remains to determine distances between pairs of vertices where any shortest path consists of at least t edges. For this purpose, we determine a subset B of V, the so called bridging set [73], hitting all the aforementioned long paths. We apply the following fact to $l = t$ and sets of t vertices on shortest consisting of exactly $t-1$ edges, similarly as in [5, 15, 71, 73].

Fact 1.1. *Given a collection of N subsets of $\{1,\ldots,n\}$, where each subset has size exactly l, we can find a subset B of size $O((n/l)\log n)$ that hits all subsets in the collection in time $O(Nl)$.*

Since our application of Fact 1.1 is analogous to those in [5, 15, 71, 73], we solely sketch it referring the reader for details to the aforementioned papers.

Note that for each pair $v,\ u$, of vertices for which any shortest path has at least t edges there is a pair v', u' of vertices on a shortest path from v to u such that any shortest path from v' to u' has exactly $t-1$ edges. For all such pairs v', u', we can find a shortest path on $t-1$ edges, and thus on t vertices, by backtracking on the computation of D_G^{t-1} and using witnesses for the mixed products. In total, we generate $O(n^2)$ such paths on t vertices in time $O(tn^2)$. The application of Fact 1.1 also takes time $O(tn^2)$.

Next, we run Dijkstra's single-source shortest path algorithm [3] for all vertices in the bridging set B in the input graph G and in the graph resulting from reversing the direction of edges in G. In this way, we determine $D_G[v,u]$ for all pairs $(v,u) \in (B\times V)\cup(V\times B)$.

Now, it is sufficient for all remaining pairs (v,u) in $V\times V$ to set

$$D_G[v,u] = \min\{D_G^{t-1}(v,u), \min_{b\in B}\{D_G[v,b]+D_G[b,u]-w(b)\}$$

in order to determine the whole D_G.

The computation of D_G^{t-1} takes $O(tT_{mix}(n))$ time which asymptotically is not less than the $O(tn^2)$ time taken by the construction of the bridging set. The runs of Dijkstra's algorithm and the final computation of D_G require $\widetilde{O}(\frac{n}{t}n^2)$ time. By setting $t=\sqrt{\frac{n^3}{T_{mix}(n)}}$, we obtain the lemma. □

1.3 Fast Computation of the Mixed Products for Clustered Data

Our algorithm for the right (or, left) mixed product relies on computation of an approximate minimum spanning tree of the columns (or rows, respectively) of the Boolean input matrix in the Hamming metric.

1.3.1 Approximate minimum spanning tree in high dimensional space

For $c \geq 1$ and a finite set S of points in a metric space, a *c-approximate minimum spanning tree for S* is a spanning tree in the complete weighted graph on S, with edge weights equal to the distances between the endpoints, whose total weight is at most c times the minimum.

In [40] (section 4.3) and [39] (section 3), Indyk and Motwani in particular considered the bichromatic ε-approximate closest pair problem for n points in R^d with integer coordinates in $O(1)$ under the L_p metric, $p \in \{1, 2\}$. They showed that there is a dynamic data structure for this problem which supports insertions, deletions and queries in time $O(dn^{1/(1+\varepsilon)})$ and requires $O(dn + n^{1+1/(1+\varepsilon)})$-time preprocessing. In consequence, by a simulation of Kruskal's algorithm they deduced the following fact.

Fact 1.2. *For $\varepsilon > 0$, a $1+\varepsilon$-approximate minimum spanning tree for a set of n points in R^d with integer coordinates in $O(1)$ under the L_1 or L_2 metric can be computed by a Monte Carlo algorithm in time $O(dn^{1+1/(1+\varepsilon)})$.*

In [41] Indyk, Schmidt and Thorup reported even slightly more efficient (by a poly-log factor) reduction of the problem of finding a $1+\varepsilon$-approximate minimum spanning tree to the bichromatic ε-approximate closest pair problem via an easy simulation of Prim's algorithm.

Note that the L_1 metric for points in R^n with $0,\ 1$-coordinates coincides with the n-dimensional Hamming metric. Hence, Fact 1.2 immediately yields the following corollary.

Corollary 1.1. *For $\varepsilon > 0$, a $1+\varepsilon$-approximate minimum spanning tree for a set of n $0-1$ strings of length n under the Hamming metric can be computed by a Monte Carlo algorithm in time $O(n^{2+1/(1+\varepsilon)})$.*

1.3.2 The algorithm for mixed matrix product

The idea of our combinatorial algorithm for the right mixed product C of A with B and its witness matrix is a generalization of that from [11]. Let $P(r, v)$ denote a priority queue (implemented as a heap) on the entries $A[r, k]$ such that $B[k, v] = 1$ ordered by their values in nondecreasing order.

First, we compute an approximate minimum spanning tree of the columns of B in the Hamming metric. Then, we fix a traversal of the tree. Next, for each row r of A, we traverse the tree, construct $P(r, start)$ where *start* is the first column of B in the tree traversal and then maintain $P(r, v)$ for the currently traversed v by updating

$P(r,u)$ where u is the predecessor of v in the traversal. A minimum element in $P(r,v)$ yields a witness for $C[r,v]$. The cost of the updates in a single traversal of the tree is proportional to the cost of the tree modulo a logarithmic factor.

Algorithm 1.1

Input: $n \times n$ matrix A over $R \cup \{+\infty\}$ and an $n \times n$ Boolean matrix B; $P(r,v)$ stands for a priority queue on the entries $A[r,k]$ s.t. $B[k,v] = 1$ ordered by their values in nondecreasing order.

Output: A witness matrix W for the right mixed product C of A and B.

1: Compute an $O(\log n)$-approximate minimum spanning tree T_B of the columns of B in the Hamming metric;
2: Fix a traversal of the tree T_B linear in its size;
3: Set *start* to the first node of the traversal;
4: **for each** pair of consecutive neighboring columns v, u in the traversal **do**
5: precompute the set $D_{v,u}$ of positions where 1s occur in v but not in u and the set $D_{u,v}$ of positions where 1s occur in u but not in v;
6: **end for**
7: **for each** row r of A **do**
8: Construct the priority queue $P(r,start)$ and if $P(r,start) \neq \emptyset$ set $W[r,start]$ to the index k where $A[r,k]$ is the minimum element in $P(r,start)$;
9: Traverse the tree T_B and for each node v different from *start* compute the priority queue $P(r,v)$ from the priority queue $P(r,u)$, where u is the predecessor of v in the traversal, by utilizing $D_{v,u}$ and $D_{u,v}$. If $P(r,v) \neq \emptyset$ set $W[r,v]$ to the index k where $A[r,k]$ is the minimum element in $P(r,v)$.
10: **end for**

Lemma 1.3. *Algorithm 1.1 is correct, i.e., it outputs the witnesses matrix for the right mixed product of matrices A and B.*

For an $n \times n$ Boolean matrix C, let K_C be the complete weighted graph on the rows of C where the weight of an edge between two rows is equal to their Hamming distance. Next, let $MWT(C)$ be the weight of a minimum weight spanning tree of K_C.

Lemma 1.4. *Algorithm 1.1 can be implemented in time* $\widetilde{O}(n(n + MWT(B^t))) + t(n)$, *where* $t(n)$ *is the time taken by the construction of the* $O(\log n)$*-approximate minimum weight spanning tree in step 1.*

Proof. Step 1 can be implemented in time $t(n)$ while steps 2, 3 take time $O(n)$. Step 4 takes $O(n^2)$ time. The block in Step 7 isciterated n times.

The first step in the block, i.e., the construction of $P(r, start)$ takes $O(n \log n)$ time. The update of $P(r,u)$ to $P(r,v)$ takes $O(\log n(|D_{v,u}| + |D_{u,v}|))$ time. Note that $|D_{v,u}| + |D_{u,v}|$ is precisely the Hamming distance between the columns v and u. It follows by the $O(\log n)$ approximation factor of T_B that the total time taken by these updates is $O(MWT(B^t) \log^2 n)$.

We conclude that Step 7 can be implemented in time $\widetilde{O}(nMWT(B^t))$. □

Theorem 1.1. *The right mixed product of two $n \times n$ matrices A over $R \cup \{+\infty\}$ and B over $\{0,1\}$ can be computed by a combinatorial randomized algorithm in time $\widetilde{O}(n(n + MWT(B^t)))$. Analogously, the left mixed product of B and A can be computed by a combinatorial randomized algorithm in time $\widetilde{O}(n(n + MWT(B)))$.*

Proof. By Corollary 1.1, an $\Theta(\log n)$-approximate minimum spanning tree can be constructed by a Monte Carlo algorithm in time $\widetilde{O}(n^2)$ (observe that $n^{1/f} = O(1)$ if $f = \Omega(\log n)$). Hence, by Lemmata 1.3, 1.4, we obtain the theorem for the right mixed product. The upper bound on the time required to compute the left mixed product follows symmetrically. □

1.4 Main Results

Lemma 1.2 combined with Theorem 1.1 yield our main result.

Theorem 1.2. *Let G a directed graph G on n vertices with non-negative real vertex weights. The all-pairs shortest path problem for G can be solved by a combinatorial randomized algorithm in time*

$$\widetilde{O}(n^2 \sqrt{n + \min\{MWT(A_G), MWT(A_G^t)\}}).$$

By setting vertex weights, say, to zero, we obtain immediately the following corollary.

Corollary 1.2. *The transitive closure of a directed graph G on n vertices can be computed by a combinatorial randomized algorithm in time*

$$\widetilde{O}(n^2 \sqrt{n + \min\{MWT(A_G), MWT(A_G^t)\}}).$$

Equivalently, we can formulate Corollary 1.2 as follows.

Corollary 1.3. *The transitive closure of an $n \times n$ Boolean matrix B (over the Boolean semi-ring) can be computed by a combinatorial randomized algorithm in time*

$$\widetilde{O}(n^2 \sqrt{n + \min\{MWT(B), MWT(B^t)\}}).$$

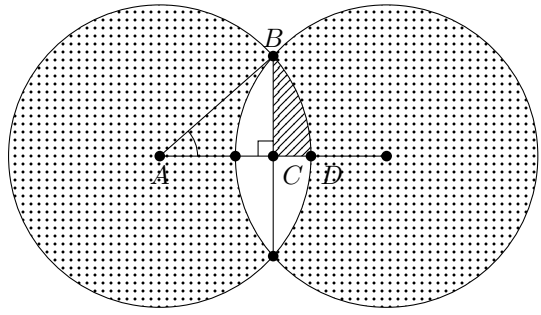


Figure 1.1

1.5 APSP in Vertex-Weighted Uniform Disk Graphs of Bounded Density

In this section, we consider uniform disk graphs that are induced by a set P of n points in a unit square in the plane that are $b(n)$-dense, where $b : N \to N$. Formally, we say that P is $b(n)$-dense iff each cell of the regular $\sqrt{n} \times \sqrt{n}$ grid within the unit square contains at most $b(n)$ points. The vertices of such an induced disk graph are the points in P, and two vertices are adjacent in the graph iff their Euclidean distance is at most r, where r is a positive constant not exceeding 1. We shall term the aforementioned graphs as *uniform disk graphs induced by $b(n)$-dense point sets*.

Lemma 1.5. *Given two intersecting disks on the plane of the same radius r with the distance d between centers, the area of the symmetric difference is $O(rd)$.*

Proof. $AC = \frac{d}{2}, AB = r$.

The area of the triangle ABC is

$$Area_{ABC} = \frac{1}{2}ACBC = \frac{1}{2}\frac{d}{2}\sqrt{r^2 - \frac{d^2}{4}} = \frac{1}{8}d\sqrt{4r^2 - d^2}.$$

The area of the circular sector ABD is

$$Area_{ABD} = \frac{1}{2}r^2 \angle BAC = \frac{1}{2}r^2 \arccos\left(\frac{d}{2r}\right).$$

The area of BCD is $Area_{BCD} = Area_{ABD} - Area_{ABC}$.

The area of the symmetric difference

$$Area = 2(\pi r^2 - 4Area_{BCD}) = 2\pi r^2 - 4r^2 \arccos\left(\frac{d}{2r}\right) + d\sqrt{4r^2 - d^2}.$$

Finally, by using Taylor series expansion

$$4r^2 \arccos\left(\frac{d}{2r}\right) = 4r^2\left(\frac{\pi}{2} - \frac{d}{2r} + O\left(\left(\frac{d}{2r}\right)^2\right)\right) =$$
$$= 2\pi r^2 - 2dr + O(d^2) = 2\pi r^2 - 2dr + O(rd)$$

and $\sqrt{4r^2 - d^2} \leq 2r$ we get $Area = O(rd)$. □

Lemma 1.6. *Let G be a uniform disk graph induced by a $b(n)$-dense point set. For each edge (v,u) of G, the number of vertices in G that are a neighbor of exactly one of the vertices v, u, i.e., the Hamming distance between the two rows in the adjacency matrix of G corresponding to v and u, respectively, is $O(r \times b(n)(dist(v,u) \times n + \sqrt{n}))$.*

Proof. The number of vertices of G that are a neighbor of exactly one of the vertices v and u is at most the minimum number of cells of the regular $\sqrt{n} \times \sqrt{n}$ grid within the unit square that cover the symmetric difference $S(v,u)$ between the disks centered at v and u, respectively, multiplied by $b(n)$. The aforementioned number of cells is easily seen to be at most the area $A(v,u)$ of $S(v,u)$ divided by the area of the grid cell, i.e., $A(v,u) \times n$, plus the number of cells of the grid intersected by the perimeter of $S(v,u)$, i.e., $O(r\sqrt{n})$. By Lemma 1.5, we have $A(v,u) = O(dist(v,u) \times r)$. Hence, the aforementioned number of cells is $O(r(dist(v,u) \times n + \sqrt{n}))$. □

The following lemma is a folklore (e.g., it follows directly from the upper bound on the length of closed path through a set of points in a d-dimensional cube given in Lemma 2 in [44]).

Lemma 1.7. *The minimum Euclidean spanning tree of any set of n points in a unit square in the plane has total length $O(\sqrt{n})$.*

Combining Lemmata 1.6, 1.7, we obtain the following one.

Lemma 1.8. *For a uniform disk graph G induced by a $b(n)$-dense n-point set, a spanning tree of the rows (or, columns) of the adjacency matrix of G in the Hamming metric having cost $O(rn^{3/2})$ can be found in time $O(n^2)$.*

Proof. Construct a minimum Euclidean spanning tree of the n points forming the vertex set of G. It takes time $O(n \log n)$ and the resulting tree T has total length $O(\sqrt{n})$ by Lemma 1.7. Form a spanning tree U of the rows (or, columns) of the adjacency matrix of G by connecting by edge the rows corresponding to v and u iff $(v,u) \in T$. By Lemma 1.6 and the $O(\sqrt{n})$ length of T, the total cost of U is $O(rn^{3/2}b(n))$. □

By plugging Lemma 1.8 into Theorem 1.2, we obtain our main result in this section.

Theorem 1.3. *Let G be a uniform disk graph, with non-negative real vertex weights, induced by a $b(n)$-dense n-point set. The all-pairs shortest path problem for G can be solved by a combinatorial algorithm in time $\widetilde{O}(\sqrt{r}n^{2.75}\sqrt{b(n)})$.*

In the application of the method of Theorem 1.2 yielding Theorem 1.3, we can use the deterministic algorithm of Lemma 1.8 to find a spanning tree of the rows or columns of the adjacency matrix of G instead of the randomized approximation algorithm from Fact 1.2.

By straightforward calculations, our upper time-bound for APSP in vertex-weighted uniform disk graphs induced by $O(1)$-dense point sets subsumes that for APSP in sparse graphs based on Dijkstra's single-source shortest-path algorithm, running in time $\widetilde{O}(nm)$, where m is the number of edges, for $r >> n^{-1/6}$.

Finally, we can also easily extend Theorem 1.3 to include uniform ball graphs in a d-dimensional Euclidean space. In the extension, the term $\sqrt{r}$ in the upper time-bound generalizes to $\sqrt{r^{d-1}}$.

1.6 Final Remarks

We can easily extend our main result to include solving the APSP problem for vertex and edge weighted directed graphs in which the number of different edge weights is bounded, say by q. This can be simply achieved by decomposing the adjacency matrix A_G into the union of up to q matrices $A_1, A_2, \ldots, A_l$ in one-to-one correspondence with the distinct edge weights and consequently replacing each mixed product with l such products in Lemmata 1.1, 1.2. In the final upper bound, $MWT(A_G)$ and $MWT(A_G^t)$ are replaced by $\sum_{i=1}^{l} MWT(A_i)$ and $\sum_{i=1}^{l} MWT(A_i^t)$, respectively.

It is an interesting problem to determine if there are other natural graph classes where $MWT(A_G)$ or $MWT(A_G^t)$ are substantially subquadratic in the number of vertices.

It follows from the existence of the so called Hadamard matrices [13] that there is an infinite sequence of graphs with $n_i \times n_i$ adjacency matrices A_i such that

$$\min\{MWT(A_i), MWT(A_i^t)\} = \Omega\left((n_i)^2\right)$$

holds.

Paper 2

Optimal Cuts and Partitions in Tree Metrics in Polynomial Time

2.1 Introduction

The optimal partition problems for unweighted and weighted graphs are classical NP-hard combinatorial optimization problems.

The typical partition problems, like MAX-CUT and MAX-BISECTION are well known to be APX-hard [36]. Also the existence of a PTAS for MIN-BISECTION has been likely ruled out in [48]. The maximization problems are typically APX-complete [36] whereas for MIN-BISECTION only logarithmic approximations are know [9, 59]. The metric counterparts of these problems are to find the corresponding optimal partitions of the complete graph on the input point set, where edges are weighted by metric distances between their endpoints. The metric MAX-CUT, MAX-BISECTION, MIN-BISECTION and other partitioning problems were all proved to have *polynomial time approximation schemes* (PTAS) [4, 22, 23, 24, 25, 43, 45]. These problems are known to be NP-hard in general metric setting (e.g., for $1-2$ metrics [25]). Their exact computational status for geometric (i.e., L_p) metrics and this even for dimension one was widely open.

In this paper, in particular, we resolve the complexity status of these problems for just dimension one by giving a polynomial time algorithm. Our solution, somewhat

(a) (b)

Figure 2.1: The cut values of the first partition (a) and the second partition (b) are respectively 8 and 6. It follows that single geometric cuts are not always sufficient to generate minimum bisections on the real line.

surprisingly, involves certain new ideas for applying dynamic programming which could be also of independent interest (see [46] for a preliminary version). In fact, our dynamic programming method works for the more general case of tree metric spaces, where the underlying trees have nonnegative real edge weights (see also the embeddability properties of arbitrary metrics in tree metrics [29]). Observe that the one-dimensional case can be modeled by line graphs with nonnegative real edge weights. We also give an evidence that our polynomial time method can be extended to include analogous partition problems in metric spaces induced by shortest paths in graphs of constant treewidth.

2.2 Preliminaries

We define our dynamic programming method in terms of generalized subproblems on finite multisets of vertices in an undirected graph with nonnegative real edge weights.

For a partition of a finite multiset P of graph vertices into two multisets P_1 and P_2, the *value of the cut* is the sum over all pairs $(v, w) \in P_1 \times P_2$ of length (i.e., total weight) of a shortest path connecting v with w. For example, see Fig. 2.1a.

The MAX-CUT problem for P will be now to find a partition of P into two multisets that maximizes the value of the cut. If $|P| = n$ and the two multisets are additionally required to be of cardinality k and $n - k$, respectively, then we obtain the $(k, n - k)$ MAX-PARTITION problem for P. In particular, if n is even and $k = n/2$ then we have the MAX-BISECTION problem. Next, if we replace the requirement of maximization with that of minimization then we obtain the $(k, n - k)$ MIN-PARTITION and MIN-BISECTION problems for P, respectively.

The aforementioned optimal partition problems can be generalized to include a multiset of points in an arbitrary metric (in particular, geometric) space by just replacing the weight of a shortest path connecting v with w with the distance between v and w.

Given a graph $G = (V, E)$, a *tree decomposition* is a pair (X, T), where $X = \{X_1, \ldots, X_l\}$ is a family of subsets (called bags) of V, and T is a tree whose nodes are

the bags X_i, satisfying the following properties [27, section 12.3]:

1. The union of all bags X_i equals V.

2. For each $(v, w) \in E$, there is a bag X_i that contains both v and w.

3. If X_i and X_j both contain a vertex v, then all nodes X_k of the tree in the (unique) path between X_i and X_j contain v as well.

The *width* of a tree decomposition is the size of its largest set X_i minus one. The treewidth $tw(G)$ of a graph G is the minimum width among all possible tree decompositions of G.

2.3 The Algorithm

Let T be a tree with at most n vertices and nonnegative real edge weights. Next, let P be a multiset of vertices of T whose cardinality does not exceed n.

Let us root T at some vertex r. Next, for each vertex v of T, let T_v stands for the subtree of T induced by all vertices of T from which the path to r passes through v. We shall assume that T_v is rooted at v. Finally, let P_v be the set of all elements in P that are copies of vertices in T_v, and let $P(v)$ stand for the set of all elements in P that are copies of the vertex v.

Consider a subtree T_v of T and a partition of P into two multisets A and B. The crucial observation is as follows.

In order to compute the total weight of shortest paths or their fragments contained within T_v, it is sufficient to know the partition of P_v into $P_v \cap A$ and $P_v \cap B$, and just the number of elements in $P \setminus P_v$ that belong to A or B.

Indeed, the specific placement of the elements of $(P \setminus P_v) \cap A$ or $(P \setminus P_v) \cap B$ in the tree T is not relevant when we consider solely the maximal fragments of shortest paths connecting to these elements that are contained within T_v.

The subproblem $MAXCUT(v, p, q, s, t)$ is to find a partition of P_v into two multisets A and B of cardinality p and q, respectively, that maximizes the sum of:

1. the total weight of shortest paths between pairs of elements in $A \times B$;

2. t times the total weight of the paths between pairs of elements in $A \times \{v\}$;

3. s times the total weight of the paths between pairs of elements in $B \times \{v\}$.

The parameters s and t are interpreted as the number of elements in $P \setminus P_v$ that belong to the same set as those in A or B, respectively, in the sought two partition of P.

We shall denote the maximum possible value of the sum by $mc(v, p, q, s, t)$. Note that the total number of subproblems is $O(n^3)$.

Assume first that T is binary.

If T_v is the singleton $(\{v\}, \emptyset)$ then $MAXCUT(v, p, q, s, t)$ can be trivially solved in $O(1)$ time. Otherwise, we can reduce the subproblem to smaller ones by the recurrences given in the following lemmata.

Lemma 2.1. *Suppose that v has two children v_1 and v_2. The value $mc(v, p, q, s, t)$ is equal to the maximum over partitions of $|P(v)|$ into the sum of natural numbers $p(v)$, $q(v)$, partitions of $|P_{v_1}|$ into the sum of natural numbers p_1 and q_1, and partitions of $|P_{v_2}|$ into the sum of natural numbers q_1, q_2, where $p = p(v) + p_1 + p_2$ and $q = q(v) + q_1 + q_2$, of the total of:*

1. $mc(v_1, p_1, q_1, s + p_2 + p(v), t + q_2 + q(v))$;
2. $mc(v_2, p_2, q_2, s + p_1 + p(v), t + q_1 + q(v))$;
3. $(p_1 q_2 + p_2 q_1)(weight(v_1, v) + weight(v_2, v))$;
4. $p_1 t \times weight(v_1, v) + p_2 t \times weight(v_2, v)$;
5. $q_1 s \times weight(v_1, v) + q_2 s \times weight(v_2, v)$.

Proof. The first and second component values correspond to the total weights of shortest paths or their fragments within T_{v_i} connecting pairs of elements in $P_{v_i} \times P_{v_i}$ or in $P_{v_i} \times (P \setminus P_{v_i})$, $i \in \{1, 2\}$, that belong to different sets in the sought two partition of P (see Fig. 2.2a).

The three remaining ones correspond to the total weight of the edges $\{v, v_1\}$ and $\{v, v_2\}$ on shortest paths between copies of vertices in P belonging to different sets in the sought two partition. In particular, the third component value corresponds to the total weight of the $\{v, v_1\}$ and $\{v, v_2\}$ fragments of shortest paths connecting pairs $\{a, b\}$ of elements in P, where a is a copy of a vertex in T_{v_1} while b is a copy of a vertex in T_{v_2} and a, b belong to different sets of the two partition (see Fig. 2.2b).

Next, the fourth component value corresponds to the total weight of the $\{v, v_1\}$ and $\{v, v_2\}$ fragments of shortest paths connecting pairs $\{a, b\}$ of elements in P, where a is a copy of a vertex in T_{v_1} or T_{v_2} belonging to the first set in the sought partition, while b is a copy of a vertex in $P \setminus P_v$ belonging to the second set in the partition. The fifth component value can be specified symmetrically by switching the first set with the second set in the two partition. For $p_1 t \times weight(v_1, v)$ term see Fig. 2.2c, the other terms are symmetrical. □

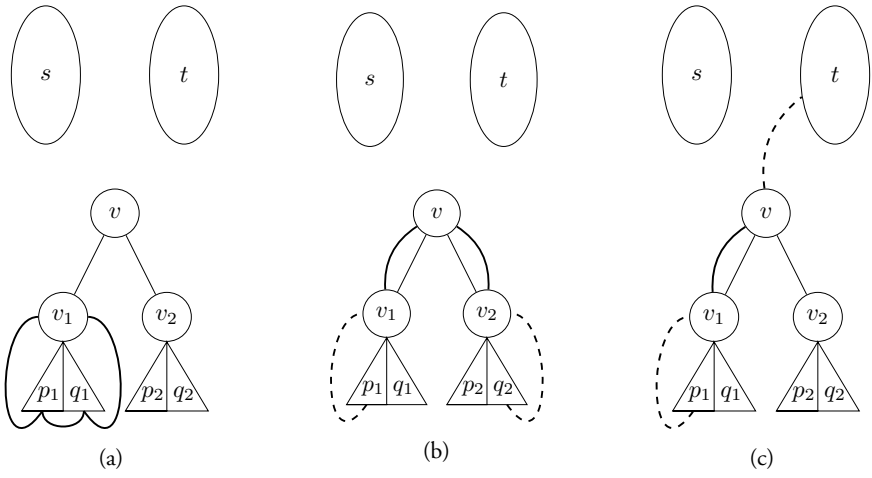


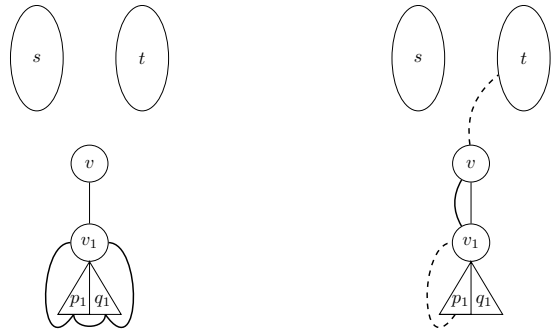
Figure 2.2: Distinct cases in the proof of Lemma 2.1

Lemma 2.2. *Suppose that v has a single child v_1. The value $mc(v,p,q,s,t)$ is equal to the maximum over partitions of $|P(v)|$ into the sum of natural numbers $p(v)$, $q(v)$, of the total of:*

1. $mc(v_1, p - p(v), q - q(v), s + p(v), t + q(v))$;
2. $(p - p(v))t \times weight(v_1, v)$;
3. $(q - q(v))s \times weight(v_1, v)$.

Proof. The first component value corresponds to the total weight of shortest paths or their fragments within T_{v_1} connecting pairs of elements in $P_{v_1} \times P_{v_1}$ or in $P_{v_1} \times (P \setminus P_{v_1})$ that belong to different sets in the sought two partition of P (see Fig. 2.3a). The second component value corresponds to the total weight of the $\{v, v_1\}$ fragments of shortest paths connecting pairs $\{a, b\}$ of elements in P, where a is a copy of a vertex in T_{v_1} belonging to the first set in the sought partition, while b is a copy of a vertex in $P \setminus P_v$ belonging to the second set in the partition (see Fig. 2.3b). Finally, the third component value can be specified symmetrically by switching the first set with the second set in the two partition. □

In Lemma 2.1, after picking a partition of $|P(v)|$ and $|P_{v_1}|$, the partition of $|P_{v_2}|$ is determined by the constraints $p = p(v) + p_1 + p_2$ and $q = q(v) + q_1 + q_2$. It



(a) (b)

Figure 2.3: Two cases in the proof of Lemma 2.2, where $p_1 = p - p(v)$, and $q_1 = q - q(v)$.

follows that the number of candidates for partition sequences in Lemma 2.1 does not exceed $O(n^2)$ while that number in Lemma 2.2 is only $O(n)$. Furthermore, if the input multiset P is a set, i.e., $|P(v)| \leq 1$ for all vertices v, then the number of candidate partition sequences in Lemmata 2.1, 2.2 is $O(n)$ and 2, respectively. We conclude that if the tree T is binary then we can compute $mc(v, p, q, s, t)$ for all the corresponding subproblems $MAXCUT(v, p, q, s, t)$ in a bottom up fashion in order of nondecreasing $|T_v|$ in $O(n^5)$ time in the general case and in $O(n^4)$ time if P is a set.

Now, to obtain the maximum value of a cut for P it is sufficient to find the maximum among the values of the form $mc(r, p, q, 0, 0)$ (recall that r is the root of T). If we additionally require $p = q$ then we obtain a maximum value of bisection etc. By backtracking, we can construct maximum cut, maximum bisection etc. The minimum variants of the partition problems can be solved analogously.

If the rooted T is not binary, we can easily transform it to a binary tree T' by adding totally at most $n - 2$ dummy vertices between parents with more than two children and their respective children. The dummy vertices between a parent v and its children form a binary subtree rooted at v whose leaves are the children. We set the weight of the edges incident to the children to the corresponding weights of the edges between v and their respective children in T. All other edge weights are set to zero in the subtree. The considered optimal cut and partitions problems for P in T are equivalent to those for P in T'.

Theorem 2.1. *Let T be a tree with at most n vertices and nonnegative real edge weights, and let P be a multiset of vertices in T with cardinality at most n. The MAX-CUT, MAX-BISECTION, $(k, n-k)$ MAX-PARTITION, MIN-BISECTION and $(k, n-k)$ MIN-PARTITION problems for P can be solved by dynamic programming in $O(n^5)$ time. If P is a set then these problems can be solved in $O(n^4)$ time.*

The geometric optimal partition problems for a finite mulitset of points on the real line can be equivalently formulated as the corresponding partition problems in the shortest-path metric induced by the line graph whose vertices correspond to the points in the multiset and whose edges correspond to the minimal intervals between different points. The edges are weighted with the length of intervals. In the case of a line graph, the reductions of subproblems to smaller ones rely solely on Lemma 2.2, where the number of candidate partitions is asymptotically smaller than that in Lemma 2.1 by a factor of n. Hence, the overall asymptotic time complexity decreases correspondingly.

Corollary 2.1. *The geometric MAX-CUT problem on the real line as well as the geometric MAX-BISECTION, $(k, n-k)$ MAX-PARTITION, MIN-BISECTION and $(k, n-k)$ MIN-PARTITION problems on the real line are solvable in $O(n^4)$ time. If the input multiset of points is a set then these problems are solvable in in $O(n^3)$ time.*

2.4 Extensions to Graphs of Bounded Treewidth for Minimum Partition Problems

A natural way of generalizing our method for tree metrics to those induced by graphs of bounded treewidth is as follows. Construct a tree decomposition T of the input graph G. For a bag b in the decomposition, let T_b be the subtree of T rooted at b, and let G_b stand for the subgraph of G induced by vertices contained in the bags of T_b. One could consider analogously subproblems with G_b, the number of vertices in the first and the second set of sought partition within G_b and $G \setminus G_b$, respectively, as the subproblem parameters.

Unfortunately, the aforementioned parameters do not seem sufficient to specify such a generalized subproblem. Simply put, if b is not a singleton (as this is in the case of trees) then it is not clear which vertex in b on a shortest path connecting a vertex in G_b with a vertex in $G \setminus G_b$ belonging to a different set in the sought partition, would be the last one on the path within G_b.

One could try to tackle this problem by introducing $2|b|$ new parameters specifying for each vertex in b the number of shortest paths connecting vertices in G_b with those in $G \setminus G_b$ belonging to a different set in the sought partition, where v is the last vertex

in G_b on the shortest paths before entering $G \setminus G_b$. (A single shortest path for each such pair would be counted.) Then, one could generalize the recurrences given in Lemmata 2.1, 2.2 but just in case of the minimum partition problems.

The reason is that taking minimum would naturally force shortest path connections in the solutions to the subproblems. On the contrary, taking maximum would rather force longest path connections in these solutions. Thus, in the latter case, the generalized method would not yield a correct solution to the posed maximum partition problem.

Let us take a closer look at the generalized recurrences for minimum partition problems. We may assume w.l.o.g. that T is binary [12]. Let b_1 and b_2 be the child bags of the bag b in the tree decomposition T of G. For each of the aforementioned $2|b|$ parameters of a subproblem associated with G_b, we need to consider possible partitions into at most $|b_1| + |b_2|$ components corresponding to the analogous parameters for the subproblems associated with G_{b_1} and G_{b_2}, respectively. Thus, if the width of the tree decomposition T is d, such a generalized recurrence would involve $(n^{O(d)})^{O(d)} = n^{O(d^2)}$ optimal solutions to smaller subproblems. It follows that the total time taken by the generalization to include minimum partitions in metric spaces induced by shortest paths in G would be $n^{O(d^2)}$. The cost of the preprocessing, i.e., the construction of the decomposition tree of G of minimal width is marginal here [12].

Claim 2.1. *Let G be a graph of treewidth d with at most n vertices and nonnegative real edge weights, and let P be a multiset of vertices in G with cardinality at most n. The MIN-BISECTION and $(k, n-k)$ MIN-PARTITION problems for P can be solved by dynamic programming in $n^{O(d^2)}$ time.*

2.5 Final Remarks

Our dynamic programming method for optimal partitions in tree metrics yields in particular polynomial time solutions to these problems on the real line. It remains an open problem whether optimal partitions in geometric metrics of dimension greater than 1 admit polynomial time methods or they turn out to be inherently hard. At stake is the exact computational status of other geometric problems for which our knowledge is very limited at the moment. The exact computation complexity status of the maximum partition problems for shortest-path metrics in bounded treewidth graphs is also a challenging open problem.

Paper 3

3D Rectangulations and Geometric Matrix Multiplication

3.1 Introduction

This paper considers two intriguing and at a first glance unrelated problems.

The first problem lies at the heart of three-dimensional computational geometry. It belongs to the class of *polyhedron decomposition* problems, whose applications range from data compression and database systems to pattern recognition, image processing, and computer graphics [47, 60]. The problem is to partition an input orthogonal polyhedron into a minimum number of 3D rectangles. Dielissen and Kaldewai [26] have shown this problem to be NP-hard. (Formally, the NP-hardness proof by [26] is for polyhedra with holes, but the authors remark that the proof should also work for simple polyhedra.) To the best of our knowledge, no non-trivial approximation factors for minimum rectangular partitions of simple orthogonal polyhedra are known, even in restricted non-trivial cases such as that of a 3D histogram, a straightforward generalization of a planar histogram; see Section 3.2 below for the definition. In contrast, the problem of partitioning an orthogonal (planar) polygonal region into a minimum number of 2D rectangles admits a polynomial-time solution [42, 47].

The second problem we consider is that of multiplying two $n \times n$ matrices. There exist fast algorithms that do so in substantially subcubic time, e.g., a recent one due

to Le Gall runs in $O(n^{2.3728639})$ time [33], but they suffer from very large overheads. On the positive side, input matrices in real world applications often belong to quite restricted matrix classes, so a natural approach is to design faster algorithms for such special cases. Indeed, efficient algorithms for *sparse* matrix multiplication have been known for long time. In the Boolean case, despite considerable efforts by the algorithms community, the fastest known combinatorial algorithms for Boolean $n \times n$ matrix multiplication barely run in subcubic time (in $O(n^3(\log\log n)^2/(\log n)^{9/4})$ time [10], to be precise), but much faster algorithms for Boolean matrix product for restricted classes of Boolean matrices have been developed [11, 34, 51]. For example, when at least one of the input Boolean matrices admits an exact covering of its ones by a relatively small number of rectangular submatrices, the Boolean matrix product can be computed efficiently [51]. Similarly, if the rows of the first input Boolean matrix or the columns of the second input Boolean matrix can be represented by a relatively cheap minimum cost spanning tree in the Hamming metric (or its generalization to include blocks of zeros or ones) then the Boolean matrix product can be computed efficiently by a randomized combinatorial algorithm [11, 34].

3.1.1 New results

Our first contribution is an $O(m \log m)$-time, 4-approximation algorithm for computing a minimum 3D rectangular partition of an input 3D histogram with m corners. It works by projecting the input histogram onto the base plane, partitioning the resulting planar straight-line graph into a number of 2D rectangles not exceeding its number of vertices, and transforming the resulting 2D rectangles into 3D rectangles of appropriate height. Importantly, the known algorithms for minimum partition of an orthogonal polygon with holes into 2D rectangles [42, 47] do not yield the aforementioned upper bound on the number of rectangles in the more general case of planar straight-line graphs.

Our second contribution is a new technique for multiplying two matrices with nonnegative integer entries. We interpret the matrices as 3D histograms and decompose them into blocks that can be efficiently manipulated in a pairwise manner using the interval tree data structure. Let A and B be two $n \times n$ matrices with nonnegative integer entries, and let r_A and r_B denote the minimum number of 3D rectangles into which the 3D histograms induced by A and B can be partitioned. By applying our 4-approximation algorithm above, we can compute $A \times B$ in $\tilde{O}(n^2 + r_A r_B)$ time, where $\tilde{O}$ suppresses polylogarithmic (in n) factors. Next, by using another idea of slicing the histogram of A (or B) into parts corresponding to rows of A (or columns of B) and measuring the cost of transforming a slice into a consecutive one, we obtain an upper bound of $\tilde{O}(n^2 + n\min\{r_A,\ r_B\})$. We also give a generalization of the latter

upper bound in terms of the minimum cost of a spanning tree for the slices, where the distance between a pair of slices corresponds to the cost of transforming one slice into the other.

We remark that $r_A = O(n^2)$ and $r_B = O(n^2)$ always hold. For inputs where $r_A r_B = \tilde{O}(n^2)$, the worst-case running time of our first algorithm for matrix multiplication is $\tilde{O}(n^2)$, which is almost optimal and much better than that of the currently fastest one for the general case [33]. Furthermore, when at least one of r_A and r_B is $\tilde{O}(n)$, the worst-case running times of our second and third algorithms are almost optimal.

3.1.2 Organization of the paper

Section 3.2 presents our 4-approximation algorithm for a partition of a 3D histogram into a minimum number of 3D rectangles. Section 3.3 presents our algorithms for the arithmetic matrix product. Section 3.4 concludes with some final remarks.

3.2 3D Histograms and Their Rectangular Partitions

A 2D histogram is a polygon with an edge e, which we call the *base* of the histogram, having the following property: for every point p in the interior of histogram, there is a (unique) line segment perpendicular to e, connecting p to e and lying totally in the interior of the histogram. In this paper, we consider orthogonal histograms only. For simplicity, we consider the base of a histogram as being horizontal, and all other edges of the histogram lying above the base. In this way, a 2D histogram can also be thought of as the union of rectangles standing on the base of the histogram.

A *3D histogram* is a natural generalization of a 2D histogram. To define a 3D histogram, we need the concept of the "base plane", which for simplicity we define as the horizontal plane containing two of the axes in the Euclidean space. A 3D histogram can then be thought of as the union of orthogonal 3D rectangles, standing on the base plane. The *base of the histogram* is the union of the lower faces (also called *bases*) of all these rectangles.

Definition 3.1. A 3D histogram is a union of a finite set C of orthogonal 3D rectangles such that: (i) each element in C has a face on the horizontal base plane; and (ii) all elements in C are located above the base plane.

(In the literature, what we call a 3D histogram is sometimes termed a 2D histogram or a 1D histogram when used to summarize 2D or 1D data, respectively [55].)

By a *rectangular partition* (or *rectangulation*) of a 3D histogram P, we mean an orthogonal partition of P into 3D rectangles. In Section 3.2.2 below, we consider

the problem of finding a rectangular partition of a given 3D histogram P into as few 3D rectangles as possible. We present a 4-approximation algorithm for this problem with time complexity $O(m \log m)$, where m denotes the number of vertices in P. The algorithm partitions P into less than m' 3D rectangles, where m' is the number of vertices in the vertical projection of P (i.e., $m' < m$), by applying a subroutine described in Section 3.2.1 that partitions any orthogonal planar straight-line graph (PSLG) with m' vertices into less than m' 2D rectangles.

3.2.1 Partitioning an orthogonal PSLG into 2D rectangles

The problem of partitioning an orthogonal polygon in two dimensions into as few rectangles as possible has been well studied in the literature [42, 47], and polynomial-time algorithms for this problem exist (see [42, 47] and the references therein). However, for our purposes, it is not necessary to compute an *optimal* solution for the 2-dimensional problem. Instead, we just need a partition of a planar straight-line graph (PSLG) into less than m' rectangles, where m' denotes the number of vertices in the input PSLG. We will show that a simple algorithm, which is faster than the optimal algorithms in [42, 47], suffices to obtain this (not always optimal) upper bound.

Since this subsection considers 2D only, we use the term "horizontal" for line segments parallel to the X-axis. By "vertical" lines, we mean lines or line segments parallel to the Y-axis. Each vertex in the planar graphs in our application has degree 2, 3, or 4.

Definition 3.2. A planar straight-line graph (PSLG) $PG = (V, E)$, as used in this paper, is a planar graph where every vertex has an x- and a y-coordinate. Each edge is drawn as a straight line segment, all edges meet at right angles, and each vertex has degree 2, 3, or 4. A *rectangular partition* of PG is a partition $R = (V \cup V_R, E \cup E_R)$ that adds edges and vertices to PG so that R is still a PSLG while every face in R is a rectangle.

Given a PSLG PG, we denote $m' = |V|$. We say that a vertex v of PG is *concave* if it has degree 2, its two adjacent edges are perpendicular to each other, and the corner at v which is of 270 degrees does not lie in the outer, infinite face of PG. Any vertex which is not concave is called *convex*.

We use a sweep line approach to generate a partition into less than m' rectangles. We perform a horizontal sweep with a vertical sweep line [21], using the vertices of PG as event points. Whenever the sweep line reaches a concave vertex v, we insert into the graph PG a vertical line segment s connecting v to the closest edge of PSLG upwards or downwards, thus canceling the concavity at v and transforming v into a

convex vertex of degree 3. Hence, if there was already an edge of PG below v, then the new segment s is inserted above v, otherwise it is inserted below v. To preserve the property that the resulting graph is still a PSLG, the other endpoint of s may have to become a new vertex of the PSLG. (This is a standard procedure for trapezoidation; see, e.g., [21] for more details.) After the sweep is complete, all concave vertices have been eliminated. (It may happen that two concave vertices with the same x-coordinate are connected by a single vertical segment that is disjoint from the rest of the input PSLG. In this case, the plane sweep algorithm will produce this segment. Thus, no two segments produced by the algorithm overlap or touch each other.)

The correctness of the algorithm is easy to see: it eliminates all concave corners of PG by adding vertical line segments. Hence, in the resulting PSLG, each face, except for the outer face, is a rectangle. The running time of this algorithm is dominated by the cost of the plane sweep, which is $O(m' \log m')$ according to well-known methods in computational geometry; see, e.g., [21].

The next lemma relates the resulting number of 2D rectangles to the number of vertices in the input PSLG.

Lemma 3.1. *Any PSLG $PG = (V, E)$ with $|V| = m'$ and minimum vertex degree 2 can be partitioned into b rectangles with $b < m'$ in $O(m' \log m')$ time.*

Proof. Let R denote the set of rectangles in the rectangular partition produced by the plane sweep algorithm described above. We use a "charging scheme" to prove the stated inequality. The charging scheme starts by giving each vertex $v \in V$ four tokens; thus, a total of $4m'$ tokens are used. Each vertex v then distributes its tokens in a certain way to the rectangles in R that are adjacent to v. We will show that every rectangle in R receives at least four tokens. Since we started by giving a total of $4m'$ tokens to the vertices, this will prove that there exist at most m' rectangles, and thus $b \leq m'$. Moreover, vertices adjacent to the outer face do not give away more than three tokens. We will thus obtain the strict inequality $b < m'$.

Now we describe the details of the charging scheme. Let v be any vertex of V. The vertex v gives one token to each rectangle r in R which in any way is adjacent to it, with one exception. The exception occurs when v is a concave vertex; then, v is partitioned by a vertical segment e_r added by the algorithm. This segment partitions the three quadrants at the concave corner around the vertex so that one rectangle occupies one quadrant and one occupies the two others. Then v distributes two tokens to the new rectangle occupying only one quadrant, which therefore has a corner at v, and only one token to each one of the other rectangles of R adjacent to v. See Fig. 3.1; in this example, v distributes only three of its four tokens, because v happens to be adjacent to the outer face.

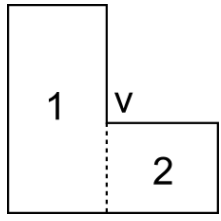


Figure 3.1: A concave corner v and a vertical segment (dashed) added by the algorithm. One of the rectangles receives two tokens, and the other one receives one token. If there had existed an additional rectangle with v as a corner, it would also have received one token.

We now show that each rectangle receives at least four tokens. Let r be any rectangle in R. First note that each vertical segment added by the algorithm has at least one endpoint at a vertex in V. Moreover, for any rectangle r in R, each of the vertical sides of r includes at least one vertex of V. Therefore, each rectangle is adjacent to at least two vertices of V. We distinguish three cases, depending on the number of vertices of V adjacent to r. Observe that the adjacencies are not necessarily at the corners of r.

- Case 1: r is adjacent to at least four vertices of V. Since r will receive at least one token from each of them we are done.
- Case 2: r is adjacent to precisely three vertices of V. Then at one of the vertical sides of r there is only one vertex of V. Moreover, this vertex v must be at a corner of r and fulfills the criteria for giving two tokens to r. The remaining two adjacent vertices of V give at least one token each, so we are done. See Fig. 3.2.
- Case 3: r is adjacent to precisely two vertices of V. This must mean that both vertical sides of r are segments added by the algorithm, and that one of the endpoints of each of these sides is a vertex of V at a corner of r. This corresponds to the condition for receiving two tokens mentioned earlier. So in total, r receives four tokens from the two corners, and we are done. See Fig. 3.2.

□

3.2.2 Partitioning a 3D histogram into 3D rectangles

We now explain how to construct a projected PSLG PP from any input 3D histogram P and how to apply the fast 2D rectangular partition algorithm from Section 3.2.1 to PP

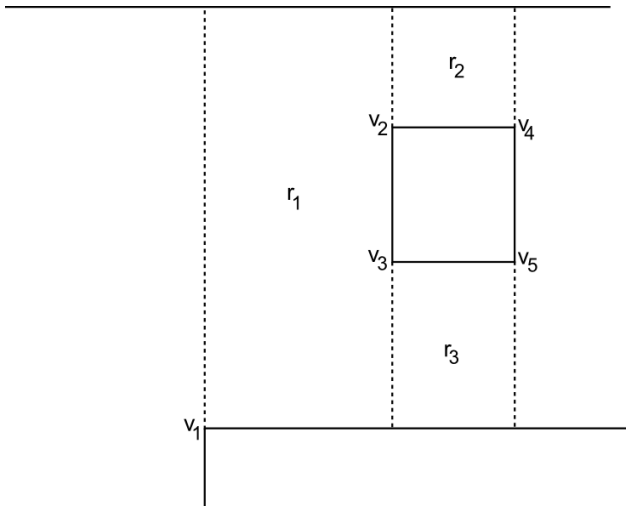


Figure 3.2: An example of two rectangles r_2 and r_3 that are only adjacent to two vertices (v_2, v_4 and v_3, v_5, respectively). The dashed line segments indicate vertical line segments added by the algorithm, while the full line segments indicate edges of PG. The rectangle r_1 is adjacent to three vertices from PG, and will get two tokens from v_1 and one from v_2 and v_3 each. r_2 and r_3 will receive two tokens from each adjacent vertex.

to obtain a good partition of P into 3D rectangles.

Definition 3.3. The *planar projection* PP is an orthogonal projection of the input 3D histogram P along the "down" direction onto the base plane in Definition 3.1.

See Fig. 3.3 for an illustration.

We can interpret PP as a PSLG where each corner and each subdividing point on a line segment corresponds to a vertex. The edges naturally correlate to the connecting line segments between vertices. Each vertex in PP is the vertical projection of at least two vertices of P. Two edges of the 3D histogram may partially overlap in the 2D projection, but the edges in the 2D projection are considered as non-overlapping. Thus, an edge of the 3D histogram may split into several edges in the 2D projection, since vertices should only appear as endpoints of edges.

Every vertex in PP must have at least two neighbors. This follows from the fact that each vertex of P (and of any orthogonal polyhedron) has at least two incident

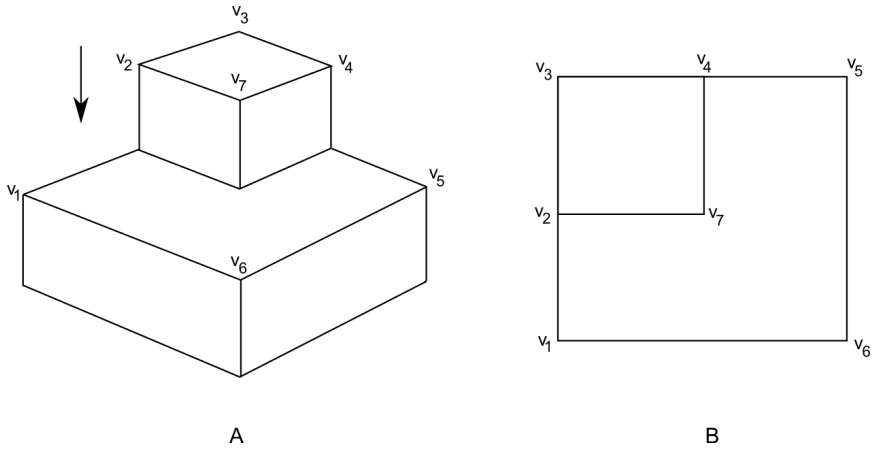


Figure 3.3: A displays a 3D histogram and the direction in which we do the projection. B displays the projected figure on the plane with corresponding vertices labeled.

horizontal edges. It may happen that some vertex of PP is the vertical projection of up to four vertices of P, so those four vertices of P may have a total of eight neighbors in P. But since PP is an orthogonal PSLG, no vertex of PP has more than four neighbors.

Now we are ready to show the main theorem of this section.

Theorem 3.1. *For any 3D histogram P with m corners, a 4-approximation R of a partition of P into as few 3D rectangles as possible can be computed in $O(m \log m)$ time.*

Proof. We let PP be the planar projection of P as in Definition 3.3, assign $PG := PP$, and apply the algorithm from Lemma 3.1 to compute a planar partition R' of PG. The final 3D partition R is obtained from R' by reversing the projection so that each 2D rectangle corresponds to the top of a 3D rectangle in R.

To analyze the approximation factor, denote the number of 3D rectangles in an optimal solution R_{opt} by OPT, the number of 3D rectangles produced by the algorithm described above by b, and the number of vertices in PP by m'. Every corner of P is adjacent to at least one vertical edge of a 3D rectangle in R_{opt}, which means that every vertex in PP has to be the vertical projection of at least one such vertical edge. Next, every 3D rectangle in R_{opt} has 4 vertical edges, so the total number of vertical edges in R_{opt} (some of which may be projected onto the same vertex in PP) is $4OPT$. Thus, $m' \leq 4OPT$. By Lemma 3.1, we have $b < m'$ and it follows that $b < 4OPT$.

Since the projection can be obtained by contracting each corner in P and all of its vertical neighbors into one vertex, the projection can be implemented in $O(m)$ time. Thus, the $O(m \log m)$-term from Lemma 3.1 will dominate the time complexity. □

3.3 Geometric Algorithms for the Arithmetic Matrix Product

In this section, we present our three geometric or in part geometric algorithms for arithmetic matrix product.

3.3.1 Geometric data structures and notation

Our algorithms for arithmetic matrix multiplication use some data structures for interval and rectangle intersection. An *interval tree* is a leaf-oriented binary search tree that supports intersection queries for a set Q of closed intervals on the real line as follows:

Fact 3.1 ([52]). *Suppose that the left endpoints of the intervals in a set Q belong to a subset $\mathcal{U}$ of real numbers of size l and $|Q| = q$. An interval tree T of depth $O(\log l)$ for Q can be constructed in $O(l + q \log lq)$ time using $O(l + q)$ space. The insertion or deletion of an interval with left endpoint in $\mathcal{U}$ into T takes $O(\log l + \log q)$ time. The intersection query is supported by T in $O(\log l + r)$ time, where r is the number of reported intervals.*

Remark 3.1. The interval tree of Fact 3.1 ([52]) can be generalized to the weighted case, where an integer weight is associated with each interval to insert or delete, by maintaining in each node of the interval tree the sum of weights of intervals whose fragments it represents. In effect, the generalized interval insertions or deletions as well the intersection query have the same time complexity as those in Fact 3.1. Moreover, the generalized interval tree supports a weight intersection query asking for the total weight of the intervals containing the query point in $O(\log l + \log q)$ time.

We use the following data structure, easily obtained by computing all prefix sums:

Fact 3.2. *For a sequence of integers $a_1, a_2, \ldots, a_n$ one can construct a data structure that supports a query asking for reporting the sum $\sum_{k=i}^{j} a_k$ for $1 \leq i \leq j \leq n$ in $O(1)$ time. The construction takes $O(n)$ time.*

In the rest of the paper, A and B denote two $n \times n$ matrices with nonnegative integer entries, and C stands for their matrix product. We also need the following concepts:

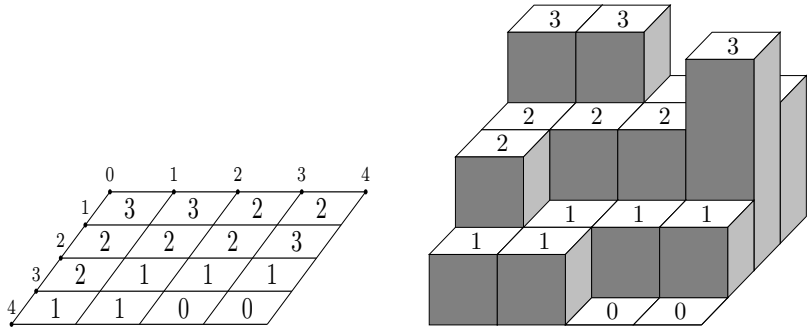


Figure 3.4: (a) A matrix D on a grid, and (b) its corresponding histogram $his(D)$.

- For any $n \times n$ matrix D with nonnegative integer entries, consider the $[0, n] \times [0, n]$ integer grid whose unit cells are in one-to-one correspondence with the entries of D. More precisely, the grid cell between the horizontal lines $i - 1$ and i (counting from the top) and vertical lines $j - 1$ and j (counting from the left) corresponds to $D_{i,j}$ (see Fig. 3.4a). Then, $his(D)$ stands for the 3D histogram whose base consists of all unit cells of the $[0, n] \times [0, n]$ integer grid corresponding to positive entries of D and whose height over the cell corresponding to $D_{i,j}$ is the value of $D_{i,j}$ (see Fig. 3.4b).

- For any $n \times n$ matrix D, nonnegative integers $1 \leq i_1 \leq i_2 \leq n$, $1 \leq k_1 \leq k_2 \leq n$, and h_1, h_2, where $h_1 < h_2 \leq D_{i,j}$ for $i_1 \leq i \leq i_2$ and $j_1 \leq j \leq j_2$, $rec_D(i_1, i_2, k_1, k_2, h_1, h_2)$ is the 3D rectangle with the corners $(i_1 - 1, k_1 - 1, h_l)$, $(i_1 - 1, k_2, h_l)$, $(i_2, k_1 - 1, h_l)$, (i_2, k_2, h_l), where $l = 1, 2$, lying within $his(D)$.

- For any $n \times n$ matrix D, r_D denotes the minimum number of 3D rectangles $rec_D(i_1, i_2, k_1, k_2, h_1, h_2)$ which form a partition of $his(D)$.

Note that $r_D \leq n^2$ for any $n \times n$ matrix D, as $his(D)$ can be trivially partitioned into n^2 3D rectangles, each covering one grid cell.

3.3.2 Algorithm 3.1

Our first geometric algorithm for nonnegative integer matrix multiplication relies on the following key lemma:

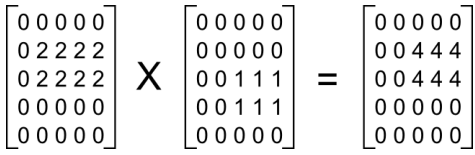


Figure 3.5: An example of how Lemma 3.2 works. The two input matrices correspond to $rec_A(2,3,2,5,2,4)$ and $rec_B(3,4,3,5,4,5)$, respectively.

Lemma 3.2. *Let P_A be a partition of the matrix A into 3D rectangles $rec_A(i_1,i_2,k_1,k_2,h_1,h_2)$, and let P_B be a partition of the matrix B into 3D rectangles $rec_B(k'_1,k'_2,j_1,j_2,h'_1,h'_2)$. For any $1 \leq i \leq n$, $1 \leq j \leq n$, the entry $C_{i,j}$ of the matrix product C of A and B is equal to the sum of $(h_2-h_1)(h'_2-h'_1) \times \#[k_1,k_2] \cap [k'_1,k'_2]$ over rectangle pairs $rec_A(i_1,i_2,k_1,k_2,h_1,h_2) \in P_A$, $rec_B(k'_1,k'_2,j_1,j_2,h'_1,h'_2) \in P_B$ satisfying $i \in [i_1,i_2]$ and $j \in [j_1,j_2]$.*

Proof. For $1 \leq l_1 < l_2 \leq n$ and $1 \leq m_1 < m_2 \leq n$, let $I(l_1,l_2,m_1,m_2)$ be the $n \times n$ $0-1$ matrix where $I(l_1,l_2,m_1,m_2)_{i,k} = 1$ if and only if $l_1 \leq i \leq l_2$ and $m_1 \leq k \leq m_2$.

Clearly, we have $A = \sum_{rec_A(i_1,i_2,k_1,k_2,h_1,h_2) \in P_A} (h_2-h_1) I(i_1,i_2,k_1,k_2)$. Similarly, we have $B = \sum_{rec_B(k'_1,k'_2,j_1,j_2,h'_1,h'_2) \in P_B} (h'_2-h'_1) I(k'_1,k'_2,j_1,j_2)$.

It follows that $C = A \times B$ is the sum over pairs $rec_A(i_1,i_2,k_1,k_2,h_1,h_2) \in P_A$, $rec_B(k'_1,k'_2,j_1,j_2,h'_1,h'_2) \in P_B$ of $(h_2-h_1)(h'_2-h'_1) \times I(i_1,i_2,k_1,k_2) \times I(k'_1,k'_2,j_1,j_2)$. It remains to observe that $(I(i_1,i_2,k_1+1,k_2) \times I(k'_1,k'_2,j_1+1,j_2))_{i,j} = \#[k_1,k_2] \cap [k'_1,k'_2]$ if $i_1 < i \leq i_2$ and $j_1 < j \leq j_2$ and it is equal to zero otherwise. □

See Fig. 3.5 for a visualization of Lemma 3.2. Lemma 3.2 suggests the following algorithm.

Lemma 3.3. *Let $int(P_A,P_B)$ be the number of pairs $rec_A(i_1,i_2,k_1,k_2,h_1,h_2) \in P_A$, $rec_B(k'_1,k'_2,j_1,j_2,h'_1,h'_2) \in P_B$, for which $[k_1,k_2] \cap [k'_1,k'_2] \neq \emptyset$. Algorithm 3.1 runs in $\tilde{O}(n^2 + int(P_A,P_B)) = \tilde{O}(n^2 + r_A r_B)$ time.*

Proof. To implement steps 1 and 2 in $\tilde{O}(n^2)$ time, use the algorithm from Theorem 3.1 in Section 3.2.2. Step 3 can be implemented in $\tilde{O}(n + r_A + r_B) = O(n^2)$ time by Fact 3.1. In Step 4, the queries to S take $\tilde{O}(int(P_A,P_B))$ time by Fact 3.1. In Step 5, the initialization of the data structure U takes $\tilde{O}(n)$ time by Lemma 3.2. Next, the updates of the data structure U take $\tilde{O}(int(P_A,P_B))$ time by Lemma 3.2, while computing all columns of C takes $\tilde{O}(n^2)$ time by Remark 3.1. □

Algorithm 3.1

Input: Two $n \times n$ matrices A, B with nonnegative integer entries.
Output: The arithmetic matrix product C of A and B.

1: Compute a partition P_A of $his(A)$ into 3D rectangles $rec_A(i_1, i_2, k_1, k_2, h_1, h_2)$ whose number is within $O(1)$ of the minimum.
2: Compute a partition P_B of $his(B)$ into 3D rectangles $rec_B(k'_1, k'_2, j_1, j_2, h'_1, h'_2)$ whose number is within $O(1)$ of the minimum.
3: Initialize an interval tree S on the k- and k'-coordinates of the rectangles in P_A and P_B. For each 3D rectangle $rec_A(i_1, i_2, k_1, k_2, h_1, h_2) \in P_A$, insert $[k_1, k_2]$ with a pointer to $rec_A(i_1, i_2, k_1, k_2, h_1, h_2)$ into S.
4: Initialize interval lists $Start_j$, End_j, for $j = 1, \ldots, n$. For each rectangle $rec_B(k'_1, k'_2, j_1, j_2, h'_1, h'_2) \in P_B$ report all intervals $[k_1, k_2]$ in S that intersect $[k'_1, k'_2]$. For each such $[k_1, k_2]$ with a pointer to $rec_A(i_1, i_2, k_1, k_2, h_1, h_2)$, insert the interval $[i_1, i_2]$ with the weight $(h_2 - h_1) \times (h'_2 - h'_1) \times \#[k_1, k_2] \cap [k'_1, k'_2]$ into the lists $Start_{j_1}$ and End_{j_2}.
5: Initialize a weighted interval tree U on endpoints $1, \ldots, n$. For $j = 1, \ldots, n$, iterate the following steps. For $j > 1$, remove all weighted intervals $[i_1, i_2]$ on the list End_{j-1} from U. Insert all weighted intervals $[i_1, i_2]$ on the list $Start_j$ into U. For $i = 1, \ldots, n$, set $C_{i,j}$ to the value returned by U in response to the weight query at i (see also Fig. 3.6).

Theorem 3.2. *The matrix product of two $n \times n$ matrices A, B with nonnegative integer entries can be computed in $O(n^2 + r_A r_B)$ time.*

Proof. Algorithm 3.1 yields the theorem. Its correctness follows from Lemma 3.2 that basically says that for each pair of 3D rectangles, $rec_A(i_1, i_2, k_1, k_2, h_1, h_2) \in P_A$ and $rec_B(k'_1, k'_2, j_1, j_2, h'_1, h'_2) \in P_B$, $C_{i,j}$ should be increased by $(h_2 - h_1) \times (h'_2 - h'_1) \times \#[k_1, k_2] \cap [k'_1, k'_2]$ for $i \in [i_1, i_2]$ and $j \in [j_1, j_2]$. In Step 4, two identical intervals $[i_1, i_2]$ corresponding to the left and right edge of the submatrix of C whose entries should be increased by the aforementioned value are inserted in the lists $Start_{j_1}$ and End_{j_2}, respectively. In both cases, they are weighted by the aforementioned value. In Step 5, in iteration j_1, the weighted interval $[i_1, i_2]$ from $Start_{j_1}$ is inserted into the weighted interval tree U, and in iteration $(j_2 + 1)$, it is removed from U as its copy is in End_{j_2}. In the iterations $j = j_1, \ldots, j_2$ in Step 5, when the interval $[i_1, i_2]$ is kept in the weighted interval tree, U and the entries of the submatrix $C_{i,j}$, $i_1 \le i \le i_2$, $j_1 \le j \le j_2$, are evaluated, the weight of the interval contributes to their value. The upper time bound follows from Lemma 3.3. □

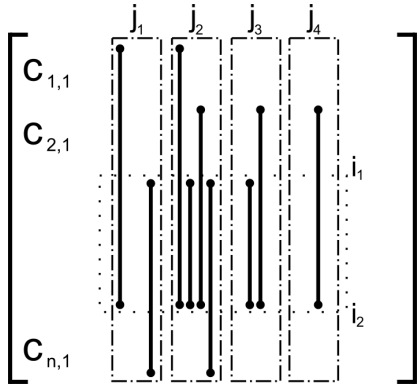


Figure 3.6: Illustrating how to fill in the entries of C column-wise in Step 5 of Algorithm 3.1 by sweeping and updating the interval tree U from the left to the right. In particular, $[i_1, i_2] \in \mathit{Start}_{j_1} \cap \mathit{End}_{j_2}$ holds.

3.3.3 Algorithm 3.2

When only one of the matrices A and B admits a partition of its 3D histogram into relatively few 3D rectangles and we have to assume the trivial partition of the other one into n^2 3D rectangles, the upper bound of Theorem 3.2 in terms of r_A, r_B and n seems too weak. In this case, an upper bound in terms of $\mathit{int}(P_A, P_B)$ and n in Lemma 3.3 may be much better. To derive a better upper bound in terms of just $\min\{r_A, r_B\}$ and n, we shall design another algorithm based on the slicing of the 3D histogram admitting a partition into relatively few 3D rectangles.

For an $n \times n$ matrix D with nonnegative integer entries and $i = 1, \ldots, n$, let $\mathit{slice}_i(D)$ stand for the part of $\mathit{his}(D)$ between the two planes perpendicular to the Y axis whose intersection with the XY plane are the horizontal lines $i-1$ and i on the $[0, n] \times [0, n]$ grid. In other words, $\mathit{slice}_i(D)$ is a 3D histogram for the i-th row. Also note that a $\mathit{slice}_i(D)$ can be identified with an orthogonal 2D histogram. See Fig. 3.7 for an example. We define a *geometric distance* between two orthogonal 2D histograms H_1 and H_2 with a common base as the number of maximal vertical strips s such that:

1. for $i = 1, 2$, s contains exactly one maximal subsegment e_i of an edge of H_i different from and parallel to the base of the histograms, and
2. the subsegments e_1 and e_2 do not overlap.

We shall call such strips *differentiating strips*. For $\mathit{slice}_i(D)$ and $\mathit{slice}_k(D)$, we define

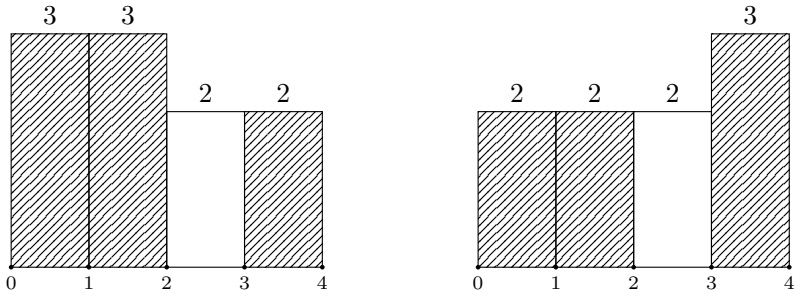


Figure 3.7: Let $slice_1(D)$ be the 2D histogram on the left and $slice_2(D)$ the 2D histogram on the right. Differentiating strips are shaded. Here, $gd(slice_1(D), slice_2(D)) = 2$.

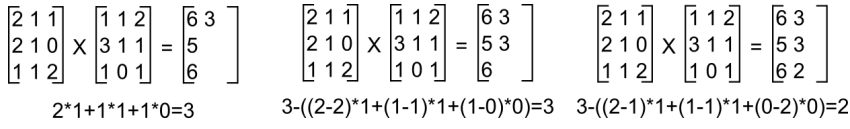


Figure 3.8: An example of how Algorithm 3.2 fills in the entries in a column of the output matrix.

the geometric distance $gd(slice_i(D), slice_k(D))$ as that for the corresponding orthogonal 2D histograms.

Lemma 3.4. *For an $n \times n$ matrix D with nonnegative integer entries, $\sum_{i=1}^{n-1} gd(slice_i(D), slice_{i+1}(D)) = O(r_D)$ holds.*

Proof. Each differentiating strip contributes, possibly jointly with one or two neighboring differentiating strips, to two vertices in the projected planar graph considered in the proof of Theorem 3.1. Thus, it contributes to the parameter m' in the aforementioned proof with at least 1. It follows $\sum_{i=1}^{n-1} gd(slice_i(D), slice_{i+1}(D)) \leq m'$. Hence, the inequality $m' \leq 4OPT$ established in the proof of Theorem 3.1 yields the lemma. □

Lemma 3.5. *Algorithm 3.2 runs in $\tilde{O}(n(n + r_A))$ time.*

Proof. Step 1 can be easily implemented in $O(n^2)$ time. Step 2 (a) takes $\tilde{O}(n)$ time according to Fact 3.2 while Step 2 (b) can be trivially implemented in $O(n)$ time.

Algorithm 3.2

Input: Two $n \times n$ matrices A and B with nonnegative integer entries.
Output: The matrix product C of A and B.

1: For $i = 1, \ldots, n-1$, find the differentiating strips for $slice_i(A)$ and $slice_{i+1}(A)$ and for each such strip s the indices $k_1(s)$ and $k_2(s)$ of the interval of entries $A_{i,k_1(s)}, \ldots, A_{i,k_2(s)}$ in the i-th row of A corresponding to it, as well as the difference $h(s)$ between the common value of each entry in $A_{i,k_1(s)}, \ldots, A_{i,k_2(s)}$ and the common value of each entry in $A_{i+1,k_1(s)}, \ldots, A_{i+1,k_2(s)}$.
2: **for** $j = 1, \ldots, n$ **do**
3: Initialize a data structure T_j for counting partial sums of continuous fragments of the j-th column of the matrix B.
4: Compute $C_{1,j}$.
5: **for** $i = 1, \ldots, n-1$ **do**
6: Set $C_{i+1,j}$ to $C_{i,j}$.
7: For each differentiating strip s for $slice_i(A)$ and $slice_{i+1}(A)$, compute $\sum_{k=k_1(s)}^{k_2(s)} B_{k,j}$ using T_j and set $C_{i+1,j}$ to $C_{i+1,j} + h(s) \sum_{k=k_1(s)}^{k_2(s)} B_{k,j}$ (see Fig. 3.8 for an illustration).
8: **end for**
9: **end for**

Finally, based on Step 1, Step 2 (c)-ii takes $\tilde{O}(gd(slice_i(D), slice_{i+1}(D)))$ time. It follows that Step 2 (c) can be implemented in $\tilde{O}(\sum_{i=1}^{n-1} gd(slice_i(A), slice_{i+1}(A)))$ time, i.e., in $\tilde{O}(r_A)$ time by Lemma 3.4. Consequently, Step 2 takes $\tilde{O}(n(n + r_A))$ time. □

Theorem 3.3. *The arithmetic matrix product of two $n \times n$ matrices A, B with nonnegative integer entries can be computed in $\tilde{O}(n(n + \min\{r_A, r_B\}))$ time.*

Proof. The correctness of Algorithm 3.2 follows from the observation that a differentiating strip s for $slice_i(A)$ and $slice_{i+1}(A)$ yields the difference $h(s) \sum_{k=k_1(s)}^{k_2(s)} B_{k,j}$ between $C_{i+1,j}$ and $C_{i,j}$ just on the fragment corresponding to $A_{i,k_1(s)}, \ldots, A_{i,k_2(s)}$ and $A_{i+1,k_1(s)}, \ldots, A_{i+1,k_2(s)}$, respectively.

Lemma 3.5 yields the upper bound $\tilde{O}(n(n + r_A))$ on the running time. The analogous bound $\tilde{O}(n(n + r_B))$ follows from the equalities $AB = (B^T A^T)^T$, $his(B) \equiv his(B^T)$, and consequently $r_B = r_{B^T}$. □

3.3.4 Algorithm 3.3

In Algorithm 3.2, the linear order in which the $C_{i,j}$ are updated to $C_{i+1,j}$ for $i = 1,\ldots,n-1$, along the row order of the matrix A is not necessarily optimal. Following the Boolean case [11, 34], it may be more efficient to update $C_{i,j}$ while traversing a minimum spanning tree for the slices of $his(A)$ under the geometric distance (i.e., a minimum spanning tree in a complete undirected graph whose vertices are in one-to-one correspondence with the slices of $his(A)$ and where each edge $\{i,j\}$ in is assigned the weight $gd(slice_i(A), slice_j(A))$). Here, however, we encounter the difficulty of constructing such an optimal spanning tree or a close approximation in substantially subcubic time. The next lemma will be useful.

Lemma 3.6. *Consider the family of orthogonal planar histograms with the base* $[0,n]$ *for any* $n \geq 2$ *and integer coordinates of its vertices in* $[0, 2^M - 2]$*, where* $M = O(\log n)$*. There is an* $O(n)$*-time transformation of any histogram* H *in the family into a* $0-1$ *string* $t(H)$ *such that, for any* H_1 *and* H_2 *in the family,* $gd(H_1,H_2) \leq hd(t(H_1),t(H_2)) \leq M \cdot gd(H_1,H_2)$*, where* $hd(\ ,\ )$ *stands for the Hamming distance.*

Proof. Any histogram H in the family is uniquely represented by the vector $(H[1],\ldots,H[n]) \in \{1,\ldots,2^M-1\}^n$, where $H[1],\ldots,H[n]$ are the values of Y coordinates of the points on the "roof" of H increased by one with X coordinates $0.5, 1.5, \ldots, n-0.5$ respectively.

For any $y \in \{0,\ldots,2^M-1\}$ denote its binary representation of length exactly M (padded with leading zeros if necessary) as $\mathrm{bin}(y)$.

$$\text{Let } f(H,i) = \begin{cases} \mathrm{bin}(H[i]), & i = 1 \vee i > 1 \wedge H[i] \neq H[i-1] \\ \mathrm{bin}(0), & \text{otherwise.} \end{cases}$$

The transformation t is then defined as $t(H) = f(H,1)\ldots f(H,n)$. We have $hd(t(H_1),t(H_2)) = \sum_{i=1}^{n} hd(f(H_1,i),f(H_2,i))$ and

$$gd(H_1,H_2) = \begin{cases} 1, & H_1[1] \neq H_2[1] \\ 0, & \text{otherwise} \end{cases} + \\ + \sum_{i=2}^{n} \begin{cases} 1, & (H_1[i] \neq H_1[i-1] \vee H_2[i] \neq H_2[i-1]) \wedge (H_1[i] \neq H_2[i]) \\ 0, & \text{otherwise.} \end{cases}$$

Consider all possibilities that contribute exactly one to $gd(H_1,H_2)$:

1. $H_1[1] \neq H_2[1]$. In this case $f(H_1,1) = \mathrm{bin}(H_1[1])$, $f(H_2,1) = \mathrm{bin}(H_2[1])$ and $0 \leq hd(\mathrm{bin}(H_1[1]),\mathrm{bin}(H_2[1])) \leq M$.

2. $2 \leq i \leq n \wedge H_1[i] \neq H_1[i-1] \wedge H_2[i] = H_2[i-1] \wedge H_1[i] \neq H_2[i]$. In this case $f(H_1, i) = \text{bin}(H_1[i])$, $f(H_2, i) = \text{bin}(0)$ and $1 \leq hd(\text{bin}(H_1[i]), \text{bin}(0)) \leq M$.

3. $2 \leq i \leq n \wedge H_1[i] = H_1[i-1] \wedge H_2[i] \neq H_2[i-1] \wedge H_1[i] \neq H_2[i]$. See case 2.

4. $2 \leq i \leq n \wedge H_1[i] \neq H_1[i-1] \wedge H_2[i] \neq H_2[i-1] \wedge H_1[i] \neq H_2[i]$. See case 1.

To complete the proof, observe that in all other cases $hd(f(H_1, i), f(H_2, i)) = 0$. □

Fact 3.3 ([40]). *For $\varepsilon > 0$, a $(1+\varepsilon)$-approximation minimum spanning tree for a set of n points in R^d with integer coordinates in $O(1)$ under the L_1 or L_2 metric can be computed by a Monte Carlo algorithm in $O(dn^{1+1/(1+\varepsilon)})$ time.*

By combining the transformation of Lemma 3.6 with Fact 3.3 applied to the L_1 metric in $\{0,1\}^n$ and selecting $\varepsilon = \log n$, we obtain the following lemma.

Lemma 3.7. *Let A be two $n \times n$ matrix with nonnegative integer entries in $[0, n^{O(1)}]$. An $O(\log^2 n)$-approximation minimum spanning tree for the set of slices of $his(A)$ under the geometric distance can be constructed by a Monte Carlo algorithm in $\tilde{O}(n^2)$ time.*

By using Lemma 3.7, we obtain the following generalization of Algorithm 3.2.

Definition 3.4. For an $n \times n$ matrix D with non-negative integer entries in $[0, n^{O(1)}]$, let M_D stand for the minimum cost of a spanning tree of $slice_i(D)$, $i \in [1, n]$.

Lemma 3.8. *Algorithm 3.3 runs in $\tilde{O}(n(n + M_A))$ with high probability.*

Proof. The approximate minimum spanning tree S in Step 1 can be constructed by a Monte Carlo algorithm in $\tilde{O}(n^2)$ time by Lemma 3.7. Its traversal can be easily found in $O(n)$ time. Since the length of the traversal is linear in n, Step 2 can be easily implemented in $O(n^2)$ time analogously as the corresponding Step 1 in Algorithm 3.2. Finally, based on Step 2, Step 3 (c)-ii takes $\tilde{O}(gd(slice_i(D), slice_l(D)))$ time analogously as Step 2 (c)-ii in Algorithm 3.2. Let U stand for the set of directed edges forming the traversal of the spanning tree S. It follows that Step 3 (c) can be implemented in $\tilde{O}(\sum_{(i,l)\in U} gd(slice_i(A), slice_l(A)))$ time, i.e., in $\tilde{O}(M_A)$ time by Lemma 3.7. Consequently, Step 3 takes $\tilde{O}(n(n + M_A))$ time. □

By Lemma 3.8 and a proof analogous to that of Theorem 3.3, we obtain a randomized generalization of Theorem 3.3:

Algorithm 3.3

Input: Two $n \times n$ matrices A and B with nonnegative integer entries in $[0, n^{O(1)}]$.
Output: The matrix product C of A and B.

1: Find an $O(\log^2 n)$-approximate spanning tree S for $slice_i(A)$, $i = 1, \ldots, n$, under the geometric distance and a traversal (i.e., a non-necessarily simple path visiting all vertices) of S.
2: For any pair $(slice_i(A), slice_l(A))$, where the latter slice follows the former in the traversal, find the differentiating strips for $slice_i(A)$ and $slice_l(A)$ and for each such strip s the indices $k_1(s)$ and $k_2(s)$ of the interval of entries $A_{i,k_1(s)}, \ldots, A_{i,k_2(s)}$ in the i-th row of A corresponding to it, as well as the difference $h(s)$ between the common value of each entry in $A_{i,k_1(s)}, \ldots, A_{i,k_2(s)}$ and the common value of each entry in $A_{l,k_1(s)}, \ldots, A_{l,k_2(s)}$.
3: **for** $j = 1, \ldots, n$ **do**
4: Initialize a data structure T_j for counting partial sums of continuous fragments of the j-th row of the matrix B.
5: Compute $C_{q,j}$ where q is the index of the slice from which the traversal of S starts.
6: **while** following the traversal of S **do**
7: Set i, l to the indices of the previously traversed slice and the currently traversed slice, respectively.
8: Set $C_{l,j}$ to $C_{i,j}$.
9: For each differentiating strip s for $slice_i(A)$ and $slice_l(A)$, compute $\sum_{k=k_1(s)}^{k_2(s)} B_{k,j}$ using T_j and set $C_{l,j}$ to $C_{l,j} + h(s) \sum_{k=k_1(s)}^{k_2(s)} B_{k,j}$.
10: **end while**
11: **end for**

Theorem 3.4. *Let A, B be two $n \times n$ matrices A, B with nonnegative integer entries in $[0, n^{O(1)}]$. The arithmetic matrix product of A and B can be computed by a randomized algorithm in $\tilde{O}(n(n + \min\{M_A, M_{B^T}\}))$ time with high probability.*

Proof. The correctness of Algorithm 3.3 analogously as that of Algorithm 3.2 follows from the observation that a differentiating strip s for $slice_i(A)$ and $slice_{i+1}(A)$ yields the difference $h(s) \sum_{k=k_1(s)}^{k_2(s)} B_{k,j}$ between $C_{i+1,j}$ and $C_{i,j}$ just on the fragment corresponding to $A_{i,k_1(s)}, \ldots, A_{i,k_2(s)}$ and $A_{i+1,k_1(s)}, \ldots, A_{i+1,k_2(s)}$, respectively. Lemma 3.8 yields the upper bound $\tilde{O}(n(n + M_A))$. The symmetric one $\tilde{O}(n(n + M_{B^T}))$ follows from the equality $AB = (B^T A^T)^T$. □

3.4 Final Remarks

A natural question is: In Theorem 3.1 in Section 3.2.2, would it help to replace the fast but suboptimal algorithm from Lemma 3.1 by a (slower) algorithm that optimally rectangulates the 2D projection? The answer is that it may yield improved results in certain cases, but it would not give a better approximation factor than 4 in general. An example of this is when the optimal 3D partition consists of k cubes of decreasing sizes lying on top of each other. Then the 2D projection is k concentric squares of different sizes and an optimal rectangulation of the corresponding 2D projection consists of $4k - 3$ rectangles. Here, the approximation factor tends to 4 as k increases, and we conclude that the fast algorithm from Lemma 3.1 is good enough.

As mentioned in Section 3.1, the general problem of computing a minimum 3D rectangular partition of an *unrestricted* orthogonal polyhedron is NP-hard [26]. However, it is unknown whether the problem is NP-hard or not in the special case where the input is a *3D histogram*. Although the existence of a polynomial-time algorithm for this particular problem variant would not affect the time complexity of our matrix multiplication algorithms (because the constant-factor approximations of r_A and r_B incurred by Theorem 3.1 are absorbed into the asymptotic running times anyway), it might be an interesting theoretical issue to resolve.

The 4-approximation algorithm for minimum rectangular partition of a 3D histogram in case the histogram is $his(D)$ for an input $n \times n$ matrix D with nonnegative integer entries can be implemented in $O(n^2)$ time. Also note that the resulting partition of $his(D)$ can be used to form a compressed representation of D requiring solely $\tilde{O}(r_D)$ bits if the values of the entries in D are $n^{O(1)}$-bounded.

Our geometric algorithms for integer matrix multiplication can also be applied to derive faster $(1+\varepsilon)$-approximation algorithms for integer matrix multiplication; if the range of an input matrix D is $[0, n^{O(1)}]$, then round each entry to the smallest integer power of $(1+\varepsilon)$ that is not less than the entry. The resulting matrix D' has only a logarithmic number of different entry values and hence $r_{D'}$ may be much less than r_D.

We also note that our algorithms and upper time bounds for integer $n \times n$-matrix multiplication can be extended to integer *rectangular* matrix multiplication in a straightforward way.

Finally, our geometric algorithms for matrix multiplication can be adapted to compute other matrix product of two integer $n \times n$ matrices, e.g., their distance product, i.e., the matrix product of the matrices over the semi-ring $(\mathbb{Z}, \min, +)$, within the same asymptotic complexity. In the case of Algorithm 3.1, we need to assume that the rectangular partitions of the matrices consist solely of 3D rectangles placed on the base plane. Then in step 4 of the adapted Algorithm 3.1, $h'_1 = h_1 = 0$, and we just

assign the weight $h_1 + h_2$ to the interval $[i_1, i_2]$. We need also to adapt the weighted interval tree U in step 5, by assigning to each node of the data structure the minimum (instead of the sum) of the weights of the intervals it represents. Thus, the answer to an intersection query becomes the minimum of the weights of intervals covering the query point. Since our 4-approximation algorithm for minimum rectangular partition of a 3D histogram produces solely partitions satisfying the aforementioned additional assumption, the asymptotic complexity of the adapted algorithm remains the same. For Algorithms 3.2 and 3.3, we need just to replace sums with minima and multiplications with additions (in step 2(c)-ii of Algorithm 3.2 and step 3(c)-iii of Algorithm 3.3, respectively). Also, the data structure T_j needs to be modified to return the values of partial minima instead of partial sums in both cases.

Paper 4

A QPTAS for the Base of the Number of Crossing-Free Structures on a Planar Point Set

4.1 Introduction

By a crossing-free structure in the Euclidean plane, we mean a *planar straight-line graph* (PSLG), i.e., a plane graph whose edges $\{v, u\}$ are represented by properly non-intersecting straight-line segments with endpoints v, u, respectively. Triangulations and spanning trees on finite planar point sets are the two most basic examples of crossing-free structures in the plane, i.e., PSLGs. The problems of counting the number of such structures for a given planar n-point set belong to the most intriguing in Computational Geometry [7, 8, 31, 37, 62, 63].

4.1.1 Counting triangulations

A *triangulation* of a set S of n points in the Euclidean plane is a PSLG on S with a maximum number of edges. Let $F_t(S)$ stand for the set of all triangulations of S.

The problem of computing the number of triangulations of S, i.e., $|F_t(S)|$, is easy when S is in convex position. Simply, by a straightforward recurrence, $|F_t(S)| = C_{n-2}$, where C_k is the k-th Catalan number, in this special case. However, in the general case, the problem of computing the number of triangulations of S is neither known to be

#P-hard nor known to admit a polynomial-time counting algorithm.

It is known that $|F_t(S)|$ lies between $\Omega(2.43^n)$ [63] and $O(30^n)$ [62]. See also Table 4.1. Since the so called flip graph whose nodes are triangulations of S is connected [66], all triangulations of S can be listed in exponential time by a standard traversal of this graph. When the number of the so called onion layers of the input point set is constant, the number of triangulations and other crossing-free structures can be determined in polynomial time [6]. Only recently, Alvarez and Seidel have presented an elegant algorithm for the number of triangulations of S running in $O^*(2^n)$ time [8] which is substantially below the aforementioned lower bound on $|F_t(S)|$ (the O^* notation suppresses polynomial in n factors).

Also recently, Alvarez, Bringmann, Ray, and Seidel [7] have presented an approximation algorithm for the number of triangulations of S based on a recursive application of the planar simple cycle separator [53]. Their algorithm runs in subexponential $2^{O(\sqrt{n}\log n)}$ time and over-counts the number of triangulations by at most a subexponential $2^{O(n^{\frac{3}{4}}\sqrt{\log n})}$ factor. It also yields a subexponential-time approximation scheme for the base of the number of triangulations of S, i.e., for $|F_t(S)|^{\frac{1}{n}}$. The authors of [7] observe also that just the inequalities $\Omega(8.65^n) \leq |F_t(S)| \leq O(30^n)$ yield the large exponential approximation factor $O(\sqrt{30/8.65}^n)$ for $|F_t(S)|$ trivially computable in polynomial time.

4.1.2 Counting spanning trees

A *spanning tree* U on a set S of n points in the Euclidean plane is a connected PSLG on S that is cycle-free, equivalently, that has $n-1$ edges. Let $F_s(S)$ stand for the set of all spanning trees on S.

It is known that $|F_s(S)|$ lies between $\Omega(6.75^n)$ [31] and $O(141.07^n)$ [37]. See also Table 4.1. The fastest known algorithms for computing $|F_s(S)|$ runs in $O^*(7.125^n)$ time [67].

The aforementioned approximation algorithm for $|F_t(S)|$ due to Alvarez, Bringmann, Ray, and Seidel can be adapted to compute $|F_s(S)|$ in the same asymptotic subexponential $2^{O(\sqrt{n}\log n)}$ time within the same asymptotic subexponential $2^{O(n^{\frac{3}{4}}\sqrt{\log n})}$ approximation factor [7]. The adaption also yields a subexponential-time approximation scheme for the base of the number of spanning trees on S, i.e., for $|F_s(S)|^{\frac{1}{n}}$.

4.1.3 Our contributions

We take a similar approximation approach to the problems of counting triangulations of S and counting spanning trees on S as Alvarez, Bringmann, Ray, and Seidel

Table 4.1: Bounds on the number of different types of plane graphs according to [7].

Graph type	Lower bound	Reference	Upper bound	Reference
Triangulations	$\Omega(2.43^n)$	[63]	$O(30^n)$	[62]
Spanning cycles	1		$O(54.55^n)$	[64]
Perfect matchings	$\Omega^*(2^n)$	[57]	$O(10.05^n)$	[65]
Spanning trees	$\Omega^*(6.75^n)$	[31]	$O(141.07^n)$	[37]

in [7]. However, importantly, instead of using recursively the planar simple cycle separator [53], we shall apply recursively the so called balanced α-cheap l-cuts of maximum independent sets of triangles within a dynamic programming framework developed by Adamaszek and Wiese in [2]. By using the aforementioned techniques, the authors of [2] designed the first quasi-polynomial time approximation scheme for the maximum weight independent set of polygons belonging to the input set of polygons with poly-logarithmically many edges.

Observe that a triangulation of S can be viewed as a maximum independent set of triangles drawn from the set of all triangles with vertices in S that are free from other points in S (triangles, or in general polygons, are identified with their open interiors). Also, a spanning tree on S can be easily complemented to a full triangulation on S. These simple observations enable us to use the aforementioned balanced α-cheap l-cuts recursively in order to bound an approximation factor of our approximation algorithm. The parameter α specifies the maximum fraction of an independent set of triangles that can be destroyed by the l-cut, which is a polygon with at most l vertices in a specially constructed set of points of polynomial size.

Similarly as the approximation algorithm from [7], our algorithm may over-count the true number of triangulations or spanning trees because the same triangulation or spanning tree, respectively, can be partitioned recursively in many different ways. In contrast with the approximation algorithm in [7], our algorithm may also under-count the number of triangulations of S or spanning trees on S, since our partitions generally destroy a fraction of edges in a triangulation or a spanning tree on S.

Our approximation algorithm for the number of triangulations of (or, the number of spanning trees on, respectively) a set S of n points with integer coordinates in the plane runs in $n^{(\log(n)/\varepsilon)^{O(1)}}$ time. For $\varepsilon > 0$, it returns a number at most $2^{\varepsilon n}$ times smaller and at most $2^{\varepsilon n}$ times larger than the number of triangulations of S (or, the number of spanning trees on S, respectively). Note that even for $\varepsilon = (\log n)^{-O(1)}$, the running time is still quasi-polynomial.

As a corollary, we obtain quasi-polynomial approximation schemes for the base

of the number of triangulations of S, i.e., for $|F_t(S)|^{\frac{1}{n}}$, and the base of the number of spanning trees on S, i.e., for $|F_s(S)|^{\frac{1}{n}}$, respectively. This implies that the problems of approximating $|F_t(S)|^{\frac{1}{n}}$ and $|F_s(S)|^{\frac{1}{n}}$ cannot be APX-hard (under standard complexity theoretical assumptions).

4.1.4 Organization of the paper

In Preliminaries, we introduce basic concepts of the dynamic programming framework from [2]. In the following section, we present five properties of an abstract family of (crossing-free) structures on which the analysis of our approximation algorithm relies. Section 4.4 presents our approximation counting algorithm for the number of such structures on S and its time-complexity analysis. In Sections 4.5.1, 4.5.2, upper bounds on the under-counting and the over-counting of the algorithm are derived, respectively. In Section 4.6, we obtain our main results by showing that planar triangulations and spanning trees satisfy these five properties.

4.2 Preliminaries

An algorithm is called *quasi-polynomial-time* if its worst case running time is $n^{(\log n)^c}$ for some fixed c.

A *quasi-polynomial-time approximation scheme* (QPTAS) for an optimization or counting problem P is a family of algorithms $\{A_\varepsilon\}$ satisfying the following condition. For every $\varepsilon > 0$, there is a natural number N such that for each instance I of P with size $n \geq N$, if $Opt(I)$ is the measure of an optimal solution to I when P is an optimization problem or just the exact (positive) number in case of counting problem then the measure $A_\varepsilon(I)$ of the approximation solution or just the approximate solution returned by A_ε, respectively, satisfies $\max\{\frac{A_\varepsilon(I)}{Opt(I)}, \frac{Opt(I)}{A_\varepsilon(I)}\} \leq 1 + \varepsilon$ and A_ε runs in quasi-polynomial-time for the fixed ε.

The Maximum Weight Independent Set of Polygons Problem (MWISP) is defined as follows [2]. We are given a set Q of n polygons in the Euclidean plane. Each polygon has at most k vertices, each of the vertices has integer coordinates. Next, each polygon P in Q is considered as an open set, i.e., it is identified with the set of points forming its interior. Also, each polygon $P \in Q$ has weight $w(P) > 0$ associated with it. The task is to find a maximum weight independent set of polygons in Q, i.e., a maximum weight set $Q' \subseteq Q$ such that for all pairs $P_i,\ P_j$ of polygons in Q', if $P_i \neq P_j$ then it holds $P_i \cap P_j = \emptyset$.

The *bounding box* of Q is the smallest axis aligned rectangle containing all polygons in Q.

Note that in particular if Q consists of all triangles with vertices in a finite planar point set S such that no other point in S lies inside them or on their perimeter, each having weight 1, then the set of all maximum independent sets of polygons in Q is just the set of all triangulations of S. Recall that the latter set is denoted by $F_t(S)$ while the set of all spanning trees on S is denoted by $F_s(S)$.

Adamaszek and Wiese have shown that if $k = poly(\log n)$ then MWISP admits a QPTAS [2].

Fact 4.1 ([2]). *Let k be a positive integer. There exists a $(1+\varepsilon)$-approximation algorithm with a running time of $(nk)^{(\frac{k}{\varepsilon}\log n)^{O(1)}}$ for the Maximum Weight Independent Set of Polygons Problem provided that each polygon has at most k vertices.*

Recently, Har-Peled generalized Fact 4.1 to include arbitrary polygons [35].

We need the following tool from [2].

Definition 4.1. Let $l \in \mathbb{N}$ and $\alpha \in \mathbb{R}$ where $0 < \alpha < 1$. Let T be a set of pairwise non-touching triangles. A polygon Γ is a balanced α-cheap l-cut of T if

- Γ has at most l edges,
- the total weight of all triangles in T that intersect Γ does not exceed an α fraction of the total weight of triangles in T,
- the total weight of the triangles in T contained in Γ does not exceed two thirds of the total weight of triangles in T,
- the total weight of the triangles in T outside Γ does not exceed two thirds of the total weight of triangles in T.

For a set of triangles T in the plane, a *DP-point* is a *basic DP-point* or an *additional DP-point*. The set of basic DP-points contains the four vertices of the bounding box of T and each intersection of a vertical line passing through a corner of a triangle in T with any edge of a triangle in T or a horizontal edge of the bounding box. The set of additional DP-points consists of all intersections of pairs of straight-line segments whose endpoints are basic DP-points. The authors of [2] observe that the total number of DP-points is $O(n^4)$.

Fact 4.2 (Lemma 3.6 in [2]). *Let $\delta > 0$ and let T be a set of pairwise non-touching triangles in the plane such that the weight of no triangle in T exceeds one third of the weight of T. Then there exists a balanced $O(\delta)$-cheap $(\frac{1}{\delta})^{O(1)}$-cut with vertices at basic DP-points.*

4.3 An Abstract Crossing-Free Structure

Triangulations and spanning trees are special cases of planar straight-line graphs (PSLGs). We shall consider an abstract family (set) F_a of some finite PSLGs having five properties (satisfied by triangulations and spanning trees as shown in Section 6) presented later in this page.

We shall use the following conventions in order to specify these properties and design an approximation algorithm for counting the number of PSLGs in F_a whose vertex set is an n-point planar point set S. We shall denote the latter set by $F_a(S)$.

We shall call a member in F_a a (crossing-free) *structure*, and a member in $F_a(S)$ a structure on S. Next, we shall call any subgraph of a structure a *substructure*.

Let P be a polygon with holes. The restriction of a structure G to P is the substructure consisting of all edges and vertices of G within P. (E.g., if G is a triangulation then the restriction is a partial triangulation, and if G is a spanning tree then the restriction is a forest, in general).

We say that a substructure is within P if all its vertices and all its edges are within P. Next, we shall call a substructure $H = (V_H, E_H)$ within P *maximal* if there is no other substructure $H' = (V_{H'}, E_{H'})$ within P, where $V_H = V_{H'}$, and $E_H \subsetneqq E_{H'}$. (E.g., if H is a partial triangulation within P then it cannot be extended to any larger partial triangulation by adding more edges, similarly, if H is a forest within P then it cannot be extended to any larger forest within P by adding more edges.)

We shall assume that the family F_a has the following properties.

1. One can decide if a PSLG with at most n vertices is a structure, i.e., belongs to F_a, in at most $2^{O(n \log n)}$ time.

2. If a structure has n vertices then it has $\Omega(n)$ edges[1].

3. Any substructure is in particular a substructure of a structure on the vertex set of the substructure.

4. Any extension of the restriction of a structure G to a simple polygon P with holes to a maximal substructure on the vertices of G within P uses at most $O(l)$ additional edges, where l is the number of edges of G with endpoints in P crossed by the boundaries of P.

5. Suppose that polygons P_1, P_2 with holes form a partition of a polygon P with holes. The union of a substructure within P_1 with a substructure within P_2 is a substructure.

[1] The corresponding property from the preliminary ICALP 2015 version of this paper additionally required any two structures with the same set of vertices to have the same number of edges.

By the definitions, F_a has also the following properties.

Lemma 4.1 (Property 6)**.** *A maximal substructure H within the bounding box of the structure that H is a subgraph is a structure.*

Lemma 4.2 (Property 7)**.** *Suppose that for $j = 1, \ldots, k'$, R_j is a maximal substructure within the polygon P_j with holes, and the polygons P_1 through $P_{k'}$ are pairwise non-overlapping and their union forms a polygon P with holes. Let $R'_1, \ldots, R'_{k'}$ be another sequence of maximal substructures within $P_1, \ldots, P_{k'}$, respectively, where R_j and R'_j have the same vertex set for $j = 1, \ldots, k'$. If $R_i \neq R'_i$ for some $i \in \{1, \ldots, k'\}$, each edge extension of $\bigcup_{j=1}^{k'} R_j$ to a maximal substructure within P is different from any edge extension of $\bigcup_{j=1}^{k'} R'_j$ to a maximal substructure within P.*

Proof. The proof is by contradiction. The joint edge extension of both sequences would contain $R_j \cup R_{j'}$ within P_j which would contradict the maximality of both R_j and $R_{j'}$ within P_j. □

4.4 Dynamic Programming

Our dynamic programming approximation algorithm for $|F_a(S)|$ is termed Algorithm 4.1.

4.4.1 Time complexity

The cardinality of T does not exceed n^3. Then, by the analogy with the dynamic programming algorithm of Adamaszek and Wiese for nearly maximum independent set of triangles [2], we call a polygon in the list $\mathbb{P}$ in Algorithm 4.1 a *DP cell* and observe that the number of DP cells is $(3n^3)^{O(k)} = n^{O(k)}$ (see Proposition 2.1 in [2]). Consequently, the number of possible partitions of a DP cell into at most k DP cells is $O(\binom{n^{O(k)}}{k})$, i.e., $n^{O(k^2)}$.

It follows that if we neglect the cost of computing the exact number of maximal substructures contained within a DP cell including at most Δ input points, then Algorit 4.1 runs in $n^{O(k^2)}$ time.

We can compute the exact number of maximal substructures contained within a DP cell with at most Δ input points in $2^{O(\Delta \log \Delta)}$ time as follows. By enumerating all PSLGs on the subset of S contained in the DP cell, and using Property 1 and the fact that the number of PSLGs on at most Δ vertices is $2^{O(\Delta \log \Delta)}$, we can list all structures on this subset in $2^{O(\Delta \log \Delta)} \times 2^{O(\Delta \log \Delta)} = 2^{O(\Delta \log \Delta)}$ time. Hence, by Property 3, we

Algorithm 4.1 Approximately counting structures on a finite planar point set.

Input: A set S of n points with integer coordinates in the Euclidean plane and natural number parameters k and Δ.

Output: An approximate number of structures on the vertex set S, i.e., an approximate $|F_a(S)|$.

1: $T \leftarrow$ the set of all triangles with vertices in S that do not contain any other point in S;
2: $\mathbb{P} \leftarrow$ a list of polygons (possibly with holes) with at most k vertices in total at DP points induced by T, topologically sorted with respect to geometric containment;
3: **for each** polygon $Q \in \mathbb{P}$ containing at most Δ points in S **do**
4: $\quad as(Q) \leftarrow$ exact number of maximal substructures on the vertex set $S \cap Q$ within Q;
5: **end for**
6: **for each** polygon set $Q \in \mathbb{P}$ containing more than Δ points in S **do**
7: $\quad as(Q) \leftarrow 0$;
8: $\quad$ **for each** partition of Q into polygons $Q_1, \ldots, Q_{k'} \in \mathbb{P}$, where $k' \leq k$, no Q_j contains more than two thirds of points in $S \cap Q$, and $as(Q_1)$ through $as(Q_{k'})$ are defined **do**
9: $\quad\quad as(Q) \leftarrow as(Q) + \prod_{j=1}^{k'} as(Q_j)$;
10: $\quad$ **end for**
11: **end for**
12: Output $as(B)$, where B is the bounding box of T.

can exactly count all maximal substructures (on this subset) within the cell by pruning the aforementioned structures and checking maximality also in $2^{O(\Delta \log \Delta)}$ time. We conclude with the following lemma.

Lemma 4.3. *Algorithm 4.1 runs in* $n^{O(k^2)}2^{O(\Delta \log \Delta)}$ *time.*

4.4.2 A comparison of algorithm 4.1 with Prior Algorithms

The QPTAS of Adamaszek and Wiese for maximum weight independent set of polygons [2] is based on dynamic programming. For each polygon (possibly with holes) with at most k vertices at the DP points induced by the input polygons, termed a DP cell, an approximate maximum weight independent subset of the input polygons contained in the DP cell is computed. The computation is done by considering all possible partitions of the DP cell into at most k smaller DP cells. For each such partition, the union of the approximate solutions for the component DP cells is computed. Then, a maximum weight union is picked as the approximate solution for the DP cell.

Our algorithm, i.e., Algorithm 4.1, is in part similar to that of Adamaszek and Wiese [2]. For each DP cell, an approximate number of maximal substructures within the cell is computed instead of an approximate maximum number of non-touching input triangles within the cell. Further modification of the dynamic programming of Adamaszek and Wiese are as follows.

1. Solely those partitions of a DP cell into at most k component DP cells are considered where no component cell contains more than two thirds of the input points in the partitioned cell. (Alternatively, one could generalize the concept of a DP cell to a set of polygons with holes and consider only partitions into two DP cells obeying this restriction.)

2. While a partition of a DP cell into at most k cells is processed, instead of the union of the solutions to the subproblems for these cells, the product of the numerical solutions for the component DP cells is computed.

3. Instead of taking the maximum of the solutions induced by the partition of a DP cell into at most k DP cells, the sum of the numerical solutions induced by these partitions is computed.

4. When the number of points contained in a DP cell does not exceed the threshold number Δ then the exact number of maximal substructures within the cell is computed.

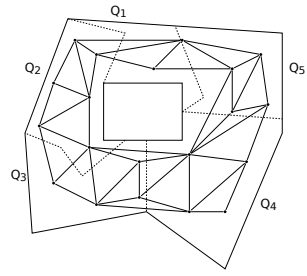


Figure 4.1: An example of a maximal partial triangulation within a DP cell and a partition of the DP cell into smaller DP cells $Q_1, \ldots, Q_5$ crossing some triangles in the triangulation.

Algorithm 4.1 also in part resembles the approximation counting algorithm for the number of triangulations of a planar point set due to Alvarez, Bringmann, Ray, and Seidel [7]. The main difference is in the used implicit recursive partition tool. Algorithm 4.1 uses balanced α-cheap l-cuts within the dynamic programming framework from [2] instead of the simple cycle planar separator theorem [7, 53]. Thus, Algorithm 4.1 recursively partitions a DP cell defining a subproblem into at most k smaller DP cells while the algorithm in [7] recursively splits a subproblem by a simple cycle that yields a balanced partition. The new partition tool gives a better running time since the number of possible partitions is much smaller so the dynamic programming/recursion has lower complexity and the threshold for the base case can be much lower. Since the algorithm in [7] in particular lists all simple cycles on $O(\sqrt{n})$ vertices, it runs in at least $2^{O(\sqrt{n}\log n)}$ time independently of the precision of the approximation.

4.5 Approximation Factor

4.5.1 Under-counting

The potential under-counting stems from the fact that when a DP cell is partitioned into at most k smaller DP cells then the possible combinations of structure edges crossing the boundaries of the cells are not counted. Furthermore, in the leaf DP cells, i.e., those including at most Δ points from S, we count only maximal substructures while the restriction of a structure on S to a DP cell does not have to be a maximal substructure within the cell. See Q_5 in Fig. 4.1.

Intuitively, the general idea of the proof of our upper bound on under-counting

is as follows. For each structure $W \in F_a(S)$, there is a substructure counted by Algorithm 4.1 that can be obtained by removing $O(\varepsilon n)$ edges from W and augmenting the resulting substructure with $O(\varepsilon n)$ other edges. The final substructure is a union of maximal substructures contained in leaf DP cells.

Lemma 4.4. *Let S be a set of n points in the plane and let $\varepsilon > 0$. For each $W \in F_a(S)$, there is a substructure $W^* \subseteq W$ on S containing at least a $1 - O(\varepsilon)$ fraction of the edges of W and a substructure $M(W^*)$ on S which is an extension of W^* by $O(\varepsilon n)$ edges such that the estimation returned by Algorithm 4.1 with k set to $\log^{O(1)}(n)/\varepsilon^{O(1)}$ is not less than $|\bigcup_{W \in F(S)}\{M(W^*)\}|$.*

Proof. Let $W \in F_a(S)$ and let $T(W)$ be any triangulation of S that is an extension of W. By adapting the idea of the proof of the approximation ratio of the QPTAS in [2], consider the following tree U of DP cells obtained by recursive applications of balanced α-cheap l-cuts.

At the root of U, there is the bounding box. By Fact 4.2, there is a balanced α-cheap l-cut, where $l = \alpha^{-O(1)}$, that splits the box into at most k children DP cells such that only α fraction of the triangular faces of $T(W)$ is crossed by the cut. The construction of U proceeds recursively in children DP cells and stops in DP cells that contain at most Δ points in S.

Note that the height of U is not greater than $\log_{3/2} n$.

For a node u of U, let W_u be the substructure that is the restriction of W to the vertices and edges of W contained in the DP cell Q_u associated with u. Analogously, let $T(W)_u$ be the partial triangulation of the points in $S \cap Q_u$ that is the restriction of $T(W)$ to (the vertices and edges of $T(W)$ contained in) Q_u. Clearly, W_u is a subgraph of $T(W)_u$. Next, let W_u^* be the substructure that is the union of W_t over the the leaves t of the subtree of U rooted at u. Note that W_u^* is a subgraph of W_u. Analogously, let $T(W)_u^*$ denote the restriction of $T(W)_u$ to the union of $T(W)_t$ over the the leaves t of the subtree of U rooted at u. Clearly, W_u^* is a subgraph of $T(W)_u^*$.

By induction on the height $h(u)$ of u in U, we obtain that the partial triangulation $T(W)_u^* \subseteq T(W)_u$ contains a $(1-\alpha)^{h(u)}$ fraction of triangular faces of $T(W)_u$. Set α to $\frac{O(\varepsilon)}{\log(n/\varepsilon)}$. It follows in particular that for the root r of U, $T(W)_r^* \subseteq T(W)$ contains at least a $(1-\alpha)^{\log_{3/2} n/\varepsilon} \geq 1 - O(\varepsilon)$ fraction of triangular faces in $T(W)$. Set $T(W)^*$ to $T(W)_r^*$ and W^* to W_r^*. By Property 2 ensuring that W has $\Omega(n)$ edges and the fact that each triangular face has three edges, we conclude that analogously W^* contains a $1 - O(\varepsilon)$ fraction of the edges of W. Thus, the number of edges in W missing in W^* is $O(\varepsilon n)$.

For a leaf t of U, let $M(W_t)$ be an (edge) extension of W_t to a maximal substructure within the leaf cell Q_t. By Property 4, the number of edges extending W_t to $M(W_t)$ is

bounded by a constant times the number of edges in W crossing the boundary of Q_t and having an endpoint within Q_t.

For a node u of U, let $M(W_u^*)$ be a substructure within Q_u that is the union of $M(W_t)$ over the leaves t of the subtree of U rooted at u. We have also $M(W^*) = M(W_r^*)$ by $W^* = W_r^*$. It follows that the number of edges extending W^* to $M(W^*)$ is bounded by a constant times the number of edges of W missing in W^*, i.e., $O(\varepsilon n)$.

We shall show by induction on $h(u)$ that Algorithm 4.1 counts at least the number of $M(W_u^*)$ while computing an estimation for Q_u.

If $h(u) = 0$, i.e., u is a leaf in U then $W_u^* = W_u$ and consequently in particular $M(W_u^*) = M(W_u)$ is counted by Algorithm 4.1.

Suppose in turn that u is an internal node in U with k' children $u_1, \ldots, u_{k'}$. When the estimation for Q_u is computed by Algorithm 4.1, the sum of products of estimations yielded by different partitions of Q_u into at most k DP cells is computed. In particular, the partition into $Q_{u_1}, \ldots, Q_{u_{k'}}$ is considered. By the induction hypothesis, the estimation for Q_{u_j} includes $M(W_{u_j}^*)$ for $j = 1, \ldots, k'$. Hence, the product of these estimations counts also $M(W_u^*) = \bigcup_{j=1}^{k'} M(W_{u_j}^*)$.

By $M(W^*) = M(W_r^*)$, to obtain the lemma it remains to show that the bound $\log^{O(1)}(n/\varepsilon)/\varepsilon^{O(1)}$ on k is sufficiently large. Following the proof of Lemma 2.1 in [2], observe that each DP cell Q_u at each level of U is an intersection of at most $O(\log(n/\varepsilon))$ polygons, each with at most l edges and vertices at basic DP points. Hence, by $\alpha = \frac{O(\varepsilon)}{\log(n/\varepsilon)}$ and $l = \alpha^{-O(1)}$, the resulting polygons have at most $O(l^2 \log^2(n/\varepsilon))$ $= \log^{O(1)}(n/\varepsilon)/\varepsilon^{O(1)}$ edges and vertices at basic and additional DP points. □

Theorem 4.1. *The under-counting factor of Algorithm 4.1 with k set to $\log^{O(1)}(n/\varepsilon)/\varepsilon^{O(1)}$ is at most $2^{O(\varepsilon n \log n)}$.*

Proof. Consider any structure $W \in F_a(S)$. By Lemma 4.4, the number of edges of W that are missing in the substructure $W^* \subseteq W$ is $O(\varepsilon n)$. It follows that the number of ways of completing W^* to a structure W' in $F_a(S)$ satisfying $(W')^* = W^*$ is not greater than the number of subsets of at most $O(\varepsilon n)$ edges of the complete Euclidean graph on S, which is $2^{O(\varepsilon n \log n)}$.

By Lemma 4.4, the estimation returned by Algorithm 4.1 with k set to $\log^{O(1)}(n/\varepsilon)/\varepsilon^{O(1)}$ is not less than $|\bigcup_{W \in F(S)} \{M(W^*)\}|$.

Now it remains to show that the maximum number of substructures $(W')^*$, $W' \in F_a(S)$, for which $M((W')^*) = M(W^*)$ is at most $2^{O(\varepsilon n \log n)}$. By Lemma 4.4, the number of edges extending $(W')^*$ to $M((W')^*)$ is at most $O(\varepsilon n)$. Consequently, the maximum number of such substructures $(W')^*$ is upper bounded by the number

of subsets of at most $O(\varepsilon n)$ edges of $M(W^*)$ (whose removal may form a substructure $(W')^*$ satisfying $M((W')^*) = M(W^*)$). The latter number is $2^{O(\varepsilon n \log n)}$.

We conclude that for $W \in F(S)$, the number of other structures $W' \in F(S)$ for which $M((W')^*) = M(W^*)$ is at most $2^{O(\varepsilon n \log n)} 2^{O(\varepsilon n \log n)} = 2^{O(\varepsilon n \log n)}$. Now, the theorem follows from Lemma 4.4. □

4.5.2 Over-counting

The reason for over-counting in the estimation returned by our algorithm is as follows. The same structure or more generally substructure within a DP cell may be cut in the number of ways proportional to the number of considered partitions of the DP cell into at most k smaller DP cells. This reason is similar to that for over-counting of the approximation triangulation counting algorithm of Alvarez, Bringmann, Ray, and Seidel [7] based on the planar simple cycle separator theorem. Therefore, our initial recurrences and calculations are similar to those derived in the analysis of the over-counting from [7].

Lemma 4.5. *Let Q be an arbitrary DP cell processed by Algorithm 4.1 which contains more than Δ input points. Recall the calculation of the estimation for Q by summing the products of estimations for smaller DP cells $Q_1, \ldots, Q_{k'}$ over $n^{O(k^2)}$ partitions of Q into $Q_1, \ldots, Q_{k'}$, $k' \leq k$. Substitute the true value of the number of maximal substructures (on input points) within each such smaller cell Q_i for the estimated one in the calculation. Let r be the resulting value. The number of maximal substructures (on input points) within Q is at least $r/n^{O(k^2)}$.*

Proof. Note that r is the sum of the number of different combinations of maximal structures within smaller DP cells $Q_1, \ldots, Q_{k'}$ over $n^{O(k^2)}$ partitions of Q into smaller cells $Q_1, \ldots, Q_{k'}$, $k' \leq k$. Importantly, each such combination can be completed to some maximal substructure within Q (Property 5) but no two different combinations coming from the same partition $Q_1, \ldots, Q_{k'}$ can be extended to the same maximal substructure within Q by Property 7 (Lemma 4.2).

Let M be the set of maximal substructures W within Q for which there is a partition into smaller DP cells $Q_1, \ldots, Q_{k'}$, $k' \leq k$, such that for $i = 1, \ldots, k'$, W constrained to Q_i is a maximal substructure within Q_i. Note that for each $W \in M$, the number of the combinations that can be completed to W cannot exceed that of the considered partitions, i.e., $n^{O(k^2)}$, as each of the combinations has to come from a distinct partition $Q_1, \ldots, Q_{k'}$.

Thus, there is a binary relationship between maximal substructures within Q that belong to M and the aforementioned combinations. It is defined on all the maximal

substructures in M and on all the combinations, and a maximal substructures in M is in relation with at most $n^{O(k^2)}$ combinations. This yields the lemma. □

By Lemma 4.5, we can express the over-counting factor $L(Q, \Delta)$ of Algorithm 1 for a DP cell Q by the following recurrence:

$$L(Q, \Delta) = \sum_{(Q_1, \ldots, Q_{k'})} \prod_{j=1}^{k'} L(Q_j, \Delta) \leq n^{O(k^2)} \prod_{j=1}^{k^*} L(Q_j^*, \Delta)$$

where the summation is over all partitions of Q into DP cells $Q_1, \ldots, Q_{k'}$, such that $k' \leq k$, and $Q_1^*, \ldots, Q_{k^*}^*$ is a partition that maximizes the term $\prod_{j=1}^{k'} L(Q_j, \Delta)$. When Q contains at most Δ input points, Algorithm 4.1 computes the exact number of maximal substructures on these points within Q. Thus, we have $L(Q, \Delta) = 1$ in this case.

Following [7], it will be more convenient to transform our recurrence by taking logarithm of both sides. For any DP cell P, let $L'(P, \Delta) = \log L(P, \Delta)$. We obtain now:

$$L'(Q, \Delta) \leq O(k^2 \log n) + \sum_{j=1}^{k^*} L'(Q_j^*, \Delta)$$

Lemma 4.6. *Let B be a bounding box for a set S of n points in the plane. The equality $L'(B, \Delta) = O(k^2 \Delta^{-1} n \log^2 n)$ holds.*

Proof. Let U be the recurrence tree and let D be the set of non-leaf nodes whose all children are leaves in U. For each node $d \in D$, the corresponding DP cell includes at least $\Delta + 1$ points in S. It follows that $|D| \leq n/\Delta$. Any node in D has depth $O(\log n)$ in U. Hence, more generally, non-leaf nodes of U are placed on $O(\log n)$ height levels of U, where each level includes at most n/Δ nodes. Each subproblem corresponding to a non-leaf node of U contributes at most $O(k^2 \log n)$ to $L'(B, \Delta)$. Consequently, the total contribution of non-leaf nodes of U to $L'(B, \Delta)$ is $O(k^2 \log n \times (n/\Delta) \log n)$. Finally, recall that the subproblems corresponding to leaves of U do not contribute to the estimation. □

Lemma 4.6 and Property 6 (Lemma 4.1) immediately yield the following corollary.

Theorem 4.2. *Let B be a bounding box for a set of n points in the plane. Set the parameter k in Algorithm 4.1 as in Theorem 4.1. If for $\varepsilon > 0$ the parameter Δ in Algorithm 4.1 is set to $\frac{c}{\varepsilon} k^2 \log^2 n$ for sufficiently large constant c then the over-counting factor is at most $2^{\varepsilon n}$.*

4.6 Main Results

Lemma 4.7. *Triangulations and spanning trees on finite planar point sets satisfy the five properties of F_a.*

Proof. Properties 1, 2, 3 and 5 are clearly satisfied by triangulations and spanning trees.

To show that Property 4 holds for triangulations, consider an extension of the restriction of a triangulation G to a simple polygon P with holes to a maximal partial triangulation on the vertices of G within P. All the edges within P added by the extension have to be incident to vertices of triangular faces of G with at least one edge crossed by the boundary of P. Observe, that such a triangular face has to have at least one vertex within P that is an endpoint of an edge of G crossed by the boundaries of P. Let l be the number of edges of G with an endpoint within P crossed by the boundaries of P. It follows that the number of aforementioned triangles is at most $2l$ and consequently the number of the endpoints of the edges within P added by the extension does not exceed $3 \times 2l = O(l)$. Hence, the total number of the added edges is also $O(l)$.

To show in turn that Property 4 holds for spanning trees, consider the forest which is the restriction of a spanning tree G to a simple polygon P with holes. Let t be the number of connected components of the forest. It follows that the number l of edges of the spanning tree G with at least one endpoint within P crossed by the boundaries of P is at least $t - 1$. On the other hand, any edge extension of the forest to a maximal forest within P may add at most $t - 1 \leq l$ edges to the forest. □

By combining Lemmata 4.7 and 4.3 with Theorems 4.1, 4.2 with ε set to $\varepsilon/\log n$, we obtain our main result.

Theorem 4.3. *There exists an approximation algorithm for the number of triangulations of (or, the number of spanning trees on) a set S of n points with integer coordinates in the plane with a running time of at most $n^{(\log(n)/\varepsilon)^{O(1)}}$ that returns a number at most $2^{\varepsilon n}$ times smaller and at most $2^{\varepsilon n}$ times larger than the number of triangulations of (or, spanning trees on, respectively) S.*

Corollary 4.1. *There exists a $(1+\varepsilon)$-approximation algorithm with a running time of at most $n^{(\log(n)/\varepsilon)^{O(1)}}$ for the base of the number of triangulations of (or, spanning trees on) a set of n points with integer coordinates in the plane.*

Proof. Let c^n be the number of triangulations of (or, the number of spanning trees on) the input n point set, and let Λ be the number returned by the algorithm from Theorem 4.3. We have $\max\{\frac{c^n}{\Lambda}, \frac{\Lambda}{c^n}\} \leq 2^{\varepsilon n}$ by Theorem 4.3. By taking the n-th root

on both sides, we obtain $\max\{\frac{c}{\Lambda^{\frac{1}{n}}}, \frac{\Lambda^{\frac{1}{n}}}{c}\} \leq 2^{\varepsilon}$. Now it is sufficient to observe that $2^{\varepsilon} < 1 + \varepsilon$ for $\varepsilon < \frac{1}{2}$. □

4.7 Final Remarks

One can a bit refine the dynamic programming (Algorithm 4.1) and consider solely partitions of a DP cell obtained by intersection with polygons with holes on most k DP-points. The number of such partitions is only $n^{O(k)}$, so the whole dynamic programming would take $n^{O(k)}2^{O(\Delta \log \Delta)}$ time. This however does not change the form of the main results.

Adamaszek and Wiese presented also an extension of their theorem on α-cheap cut of an independent set of triangles (Fact 4.2) to include independent polygons with at most K edges (Lemma 3.1 [2]). This makes possible to generalize our QPTAS for counting triangulations to include the approximation of the number of maximum weight partitions into K-gons.

The other popular crossing-free structures like perfect matchings and cycle covers (see Table 4.1) do not satisfy all the five properties of F_a. It is an intriguing open problem if they admit similar quasi-polynomial time approximation algorithms.

Paper 5

Detecting Monomials with k Distinct Variables

5.1 Introduction

Koutis initiated a group algebra approach to derivation of fully parametrized algorithms for combinatorial problems in [49]. He considered the k-path problem, which is to determine if a graph contains a simple path of length at least k, and packing problems in [49]. Williams continued this approach in [68] and improved Koutis' parametrized upper time bound for the k-path problem to $O^*(2^k)$ (the O^* notation suppresses polynomial in n and k factors). The aforementioned bounds have been obtained by a fixed-parameter reduction to the detection of the so called multilinear monomial in the sum-product expansion of a polynomial represented by arithmetic circuit of polynomial size using only addition and multiplication. A monomial is *multilinear* if no variable occurs in it more than once. Williams showed that a multilinear monomial in the sum-product expansion of the aforementioned polynomial can be detected in $O^*(2^l)$ time, where l is the degree of the polynomial (Theorem 3.1 in [68]). In the subsequent paper [50], Koutis and Williams derived new fixed-parameter upper time bounds for several other combinatorial problems by fixed-parameter reductions to the problem of multilinear monomial detections. Chen et al. generalized the FPT method of Koutis and Williams to detecting a monomial in which no variable occurs p or more times, where p is a prime number, parametrized by the degree of the polynomial [17]. A further generalization to include non-necessarily prime p was given in [18]. For the recent application of the algebraic method to include a natural generalization of the

k-path problem, see [1].

In this paper, we continue the topic of detecting monomials having some special property in the sum-product expansion of a polynomial in n variables, represented by an arithmetic circuit of size polynomial in n, using only addition and multiplication. We focus on special properties of a monomial expressed in terms of the number of distinct variables occurring in it. We consider two basic problems: detecting a monomial with at least k distinct variables, and detecting a monomial with at most k distinct variables. We also study the corresponding optimization versions searching for monomials with maximum or minimum number of distinct variables, respectively.

Our first result is a randomized FPT algorithm for detection of a monomial having at least k distinct variables, parametrized by the threshold parameter k. It relies on a slight generalization of the group algebra method from [49, 68] given in [50]. Our algorithm runs in $O^*(2^k)$ time, where the O^* notation suppresses polynomial in the size of input factors. It yields also an FPT algorithm for the decision version of the max q-cover problem [30] with the threshold k on the number of covered elements, parametrized by k.

We can also provide a deterministic FPT algorithm for detection of a monomial having at most k distinct variables, parametrized by the degree of the polynomial, provided that in the circuit representing the polynomial no multiplication gate precedes an addition gate on a directed path from an input gate to the output gate. It runs in $O^*(2^l)$ time, where l is the degree of the polynomial and the notation O^* suppresses polynomial in l and the size of the input factors. To obtain this algorithm, we design an $O^*(2^l)$-time algorithm for the hitting set problem, where l is the number of sets to be hit and O^* suppresses polynomial in the input size factors.

Next, we observe that detecting a monomial with at most k distinct variables is $W[2]$-hard with respect to the parameter k. We also observe that detecting a monomial with at least n distinct variables is $W[2]$-hard with respect to the degree of the polynomial divided by n.

We also consider the problem of finding the smallest k such that the polynomial has a monomial with at most k distinct variables as well as the problem of finding the largest k such that the polynomial has a monomial with at least k distinct variables. We show that the minimization problem cannot be approximated within $O(2^{\log^{1-\varepsilon} n})$ for any $\varepsilon > 0$, unless $NP \subset DTIME(n^{poly\log(n)})$ while the maximization problem cannot be approximated within $1 - \frac{1}{e} + \varepsilon$ for any $\varepsilon > 0$, unless $NP = P$.

Finally, we observe that analogous algorithmic, hardness, and inapproximability results on detection of implicants of Boolean functions represented by monotone Boolean circuits, with at most (or, at least, respectively) k distinct variables hold. As by-product, we infer also that deciding whether or not two monotone Boolean circuits

compute different functions is NP-complete.

Organization. In the next section, we define a monotone arithmetic circuit, a monotone Boolean circuit, and state a lemma from [50] generalizing the main results from [49, 68]. In Section 5.3, we present and analyze our FPT algorithms while in Section 5.4, we show our hardness and inapproximability results. In Section 5.5, we discuss the extensions to include the Boolean case.

5.2 Preliminaries

A monotone arithmetic circuit is a directed acyclic graph where each leaf (i.e., vertex of indegree 0) is labeled either with a variable or a real non-negative constant (input gates), each non-leaf vertex has fan-in two and it is labeled with either $+$ (addition gate) or $\times$ (multiplication gate), and a single vertex is distinguished as an output gate. A monotone Boolean circuit is defined analogously, with $\vee$ labels instead of $+$ labels, $\wedge$ labels instead of $\times$ labels and Boolean $0, 1$ constants instead of real constants.

For a positive integer k, the set of positive integers not greater than k will be denoted by $[k]$.

In Lemma 1 in [50], Koutis and Williams provided a slight generalization of their main results from [49, 68]. We can rephrase the generalization for our purposes as follows.

Fact 5.1 (See [50].)**.** *Let $P(x_1, \ldots, x_n, z)$ be a polynomial represented by a monotone arithmetic circuit of size polynomial in n. There is a randomized algorithm that for every P runs in $O^*(2^k t^2)$ time and outputs "YES" with high probability if there is a monomial of the form $z^t Q(x_1, \ldots, x_n)$, where $Q(x_1, \ldots, x_n)$ is a multilinear monomial of degree at most k, in the sum-product expansion of P, and always outputs "NO" if there is no such monomial $z^t Q(x_1, \ldots, x_n)$ in the expansion.*[1]

5.3 FPT Algorithms

5.3.1 Detecting a monomial with at least k variables

In this section, we present an FPT algorithm with respect to the polynomial degree for the "at least k distinct variables" monomial property. Our FPT algorithm relies on Fact 5.1.

[1] A slightly improved version of this fact with t^2 replaced by $t \log t$ can be found in the latest version of the paper on the web page of the authors.

Theorem 5.1. *Let k, n be natural numbers such that $k \leq n$. Next, let $P(x_1, \ldots, x_n)$ be a polynomial represented by a monotone arithmetic circuit of size polynomial in n. There is a randomized algorithm that for every P runs in $O^*(2^k)$ time and outputs "YES" with high probability if there is a monomial with at least k distinct variables in the sum-product expansion of P, and always outputs "NO" if there is no such monomial in the expansion.*

Proof. To begin with, we modify the circuit C to the circuit D as follows. We create a new input gate labeled with z, n multiplication gates representing zx_i and n addition gates representing $1 + zx_i$, for $i = 1, \ldots, n$, respectively. We connect each direct descendant of the input gate labeled with x_i with the addition gate representing $1 + zx_i$ instead, for $i = 1, \ldots, n$, respectively. Thus, the circuit D represents the polynomial $P(1 + zx_1, \ldots, 1 + zx_n)$.

Note that each monomial in the sum-product expansion of $P(1 + zx_1, \ldots, 1 + zx_n)$ can be obtained from some monomial in the sum-product expansion of $P(x_1, \ldots, x_n)$ by substitution of zx_i for some occurrences of some variables x_i and substitution of 1 for some occurrences of some variables x_j. Hence, there is a monomial with at least k distinct variables in the sum-product expansion of $P(x_1, \ldots, x_n)$ if and only if there is a monomial of the form $z^k Q(x_1, \ldots, x_n)$, where $Q(x_1, \ldots, x_n)$ is a multilinear monomial with precisely k distinct variables in $\{x_1, \ldots, x_n\}$, in the sum-product expansion of $P(1 + zx_1, \ldots, 1 + zx_n)$. By Fact 5.1, we can detect the existence of such a multilinear monomial in $O^*(2^k k^2)$ time. □

Applications. The max q-cover problem [30] is as follows: for given subsets $S_1, S_2, \ldots, S_m$ of a ground set X, find a subfamily of $\{S_1, \ldots, S_m\}$ containing at most q subsets maximizing the number of elements in X covered by the subsets included. In the decision version of this problem, there is also given a positive integer k and the objective is to decide if there is a subfamily of $\{S_1, \ldots, S_m\}$ containing at most q subsets whose union covers at least k elements.

Theorem 5.2. *The decision version of the max q-cover problem with the threshold k on the number of covered elements admits an FPT algorithm with respect to the parameter k.*

Proof. For $i = 1, \ldots, |X|$, associate the variable x_i with the i-th element in X. We shall denote elements in X by their numbers. Next, for $j = 1, \ldots, m$, let M_j denote the monomial $\prod_{i \in S_j} x_i$. It is clear that a monomial of the polynomial $\left(\sum_{j=1}^{m} M_j\right)^q$ with at least k distinct variables corresponds to a union of at most q subsets S_j covering at least k elements in X and *vice versa*. The theorem follows from Theorem 5.1.

□

Note that the known FPT algorithms for the hitting set with bounded set size [28, 56] translate to FPT algorithms for set cover with a bound on maximum number of sets that can cover a single element.

5.3.2 Detecting a monomial with at most k variables

In this subsection, we present an FPT algorithm with respect to the polynomial degree for the "at most k distinct variables" monomial property for polynomials represented by restricted monotone arithmetic circuits. The restriction does not allow any multiplication gate to precede an addition gate on a directed path from an input gate to the output gate. We shall call a monotone arithmetic circuit obeying this requirement an *addition-multiplication circuit*. A polynomial represented by such a circuit is simply a product of polynomials of degree at most one.

Recall that a subset X' of an universe X is a hitting set for a family of subsets $U_1, \ldots, U_l$ of X if for $i = 1, \ldots, l$, $X' \cap U_i \neq \emptyset$. We begin the following straightforward observation.

Lemma 5.1. *Let $P(x_1, \ldots, x_n)$ be a polynomial of maximum degree l of the form $\prod_{i=1}^{l} \sum_{x_j \in U_i} x_j$, where $U_i \subseteq \{x_1, \ldots, x_n\}$ for $i = 1, \ldots, l$. For each monomial M (in the sum-product expansion) of P, the set of variables occurring in M forms a hitting set for the subsets U_1 through U_l. Contrary, for each hitting set X' for the subsets U_1 through U_l there is a monomial M of P such that the set of variables occurring in M equals X'.*

The next lemma demonstrates that the hitting set admits an FPT algorithm with respect to the cardinality of the input family of subsets of the universe to be hit.

Lemma 5.2. *Let U be a family (more precisely, a multi-set) of l subsets $U_1, U_2, \ldots, U_l$ of $\{x_1, \ldots, x_n\}$. A minimum cardinality hitting set for U can be found in $2^l(ln)^{O(1)}$ time.*

Proof. We shall iteratively construct a subset F of the Cartesian product of the set of all subfamilies of $U = \{U_1, \ldots, U_l\}$ and the set $\{0, 1, \ldots, l\}$. Whenever F is augmented by a new element (S, k) in the j-th iteration then the subfamily S can be hit by k element subset of $\{x_1, \ldots, x_j\}$. For x_j, $j = 1, \ldots, n$, let $U(x_j)$ denote the subfamily of U consisting of all U_i containing x_j.

We initialize F by setting it to $\{(\emptyset, 0)\}$. Next, for $j = 1, \ldots, n$, we iterate the following step: For each $(S, k) \in F$ augment F by $(S \cup U(x_j), k + 1)$ provided that $(S \cup U(x_j), m) \notin F$ for all $m \leq k + 1$.

Observe that after the n-th iteration, if $(U, k) \in F$ then there is a hitting set for U of size k which can be found by backtracking. Contrary, if the minimum size of hitting set for U is k then the final F will contain (U, k). Thus, it is sufficient to

find the smallest k such that $(U,k) \in F$ and find the corresponding hitting set by backtracking.

Since the cardinality of F never exceeds $2^l(n+1)$, the whole procedure takes $2^l(ln)^{O(1)}$ time. □

By combining Lemmata 5.1, 5.2, we obtain our main result in this subsection.

Theorem 5.3. *Let l be a natural number and let k be a natural number not larger than l. Next, let $P(x_1,\ldots,x_n)$ be a polynomial of maximum degree l represented by an addition-multiplication circuit of size polynomial in n. There is an algorithm that for every P runs in $O^*(2^l)$ time and outputs "YES" if there is a monomial with at most k distinct variables in the sum-product expansion of P, and otherwise outputs "NO".*

Proof. Since sums with constant components as well as multiplicative non-zero constant factors are irrelevant and can be omitted, we may assume w.l.o.g. that the polynomial P is of the form stated in Lemma 5.1. Thus, it is sufficient to combine Lemmata 5.1, 5.2 to obtain the theorem. □

5.4 Hardness and Inapproximability Results

5.4.1 Detecting a monomial with at most k variables

Since the decision version of the set cover problem can be easily encoded as a problem of detecting a monomial with at most k variables, we obtain the following theorem.

Theorem 5.4. *Let P be a polynomial in n variables represented by a monotone arithmetic circuit with $O(n^2)$ gates. The problem of deciding if P has a monomial with at most k distinct variables is NP-complete as well as $W[2]$-hard for the parameter k.*

Proof. Consider an instance of the set cover problem with ground set X, a family of subsets $S_1,\ldots,S_n$ of X whose union covers X, and a positive integer k. We may assume w.l.o.g. that $|X| \leq n$ since the $W[2]$-hardness of set cover follows from that of dominating set [28]. For $j = 1,\ldots,n$, associate with S_j the variable y_j. Let PS stand for the polynomial $\prod_{x\in X}(\sum_{j\ s.t.\ x\in S_j} y_j)$. Note that the polynomial PS has n variables and it can be represented by a monotone arithmetic circuit of size $O(n^2)$. It is clear that X can be covered with $\leq k$ of the subsets $S_1,\ldots,S_n$ iff PS has a monomial with $\leq k$ distinct variables. This many-one reduction is clearly fixed-parameter with respect to k. Since the set cover problem is NP-complete and $W[2]$-complete with respect to k, we conclude that the decision version of the minimum-variable monomial problem is NP-complete and $W[2]$-hard for the parameter k.

□

Chechik et al. [16] have recently studied among other things the following *secluded path problem*: for a graph and two of its vertices, find a path connecting the two vertices that minimize the number of neighbors of vertices on the path. Note that in particular all vertices on such a path are accounted to the set of the path neighbors. They proved among other things that the secluded path problem is NP-hard and, unless $NP \subset DTIME(n^{poly\log(n)})$, inapproximable within a factor of $O(2^{\log^{1-\varepsilon} n})$ for any $\varepsilon > 0$.

For a vertex v in a graph $G = (V, E)$, let $N(v)$ denote the (closed) neighborhood of v, i.e., the set of all vertices adjacent to v in G, augmented by v. A *neighborhood walk* in G is a sequence of vertex neighborhoods $N(v_1), N(v_2), \ldots, N(v_l)$ in G such that for $j = 1, \ldots, l-1$, $\{v_j, v_{j+1}\} \in E$. The length of the walk is $l-1$.

We define recursively the polynomial $Q_l(i, j)$ whose monomials are in one-to-one correspondence with neighborhood walks of length $l-1$ in G starting from $N(v_i)$ and ending with $N(v_j)$ as follows:

$$Q_1(i,i) = \prod_{v_k \in N(v_i)} x_k$$

$$Q_l(i,j) = \sum_{v_q \in N(v_j)} Q_{l-1}(i,q) \prod_{v_k \in N(v_j)} x_k$$

Two following lemmata are straightforward.

Lemma 5.3. *There is a neighborhood walk of length $l-1$ starting from $N(v_i)$ and ending with $N(v_j)$ in G such that the union of its vertex neighborhoods has cardinality at most k iff $Q_l(i,j)$ has a monomial having at most k different variables.*

Lemma 5.4. *For a graph on n vertices, $i, j \in [n]$, the polynomial $\sum_{l=1}^{n} Q_l(i,j)$ can be represented by a monotone arithmetic circuit of size $O(n^2)$. Furthermore, there is a path in G with at most k neighbors starting from v_i and ending with v_j iff the polynomial $\sum_{l=1}^{n} Q_l(i,j)$ has a monomial having at most k different variables.*

Lemma 5.4 yields the following theorem.

Theorem 5.5. *Let P be a polynomial in n variables represented by a monotone arithmetic circuit with $O(n^c)$ gates. For $c \geq 2$, the problem of determining the smallest k such that P has a monomial with at most k distinct variables cannot be approximated within $O(2^{\log^{1-\varepsilon} n})$ for any (fixed) $\varepsilon > 0$, unless $NP \subset DTIME(n^{poly\log(n)})$. On the other hand, for any positive constant c, it can be solved exactly in $O(n^{k_0+c})$ time, where k_0 is the minimum number of distinct variables in a monomial of P.*

Proof. Lemma 5.4 yields a many-one fixed-parameter reduction from the secluded path problem to the problem of determining the smallest k such that a polynomial in n variables represented by a circuit of polynomial size has a monomial with precisely k distinct variables. Let m be a minimum number of neighbors on a path connecting two vertices v_i and v_j of a graph G on n vertices. By Lemma 5.4, if the aforementioned minimum-variable monomial problem had an $O(2^{\log^{1-\varepsilon} n})$ approximation algorithm then we could answer that there is a path connecting this pair of vertices and having at most $O(m/2^{\log^{1-\varepsilon} n})$ neighbors (importantly, observe that that the number n of variables in the polynomial $\sum_{l=1}^{n} Q_l(i,j)$ is equal to the number of vertices in G). This would contradict the lower bound on the approximability of the secluded path problem established in [16].

We can obtain an exact solution to the minimum -variable monomial problem by simply evaluating the polynomial $\sum_{l=1}^{n} Q_l(i,j)$ for $q = 1, 2, \ldots$ on all assignments of zero-one values with q ones until a non-zero value is produced. □

5.4.2 Detecting a monomial with at least k variables

By a reduction from the max q-cover problem (see Applications in Section 5.3.1), we can obtain also similar although weaker results on inapproximability of the symmetric problem where monomials with maximum number of distinct variables are sought.

Theorem 5.6. *Let P be a polynomial in n variables represented by a monotone arithmetic circuit with polynomial in n number of gates. The problem of determining the largest k such that P has a monomial with precisely k distinct variables cannot be approximated within $1 - \frac{1}{e} + \varepsilon$ for any $\varepsilon > 0$, unless $NP = P$. The decision version of this problem is NP-hard.*

Proof. Use the notation and the polynomial $\left(\sum_{j=1}^{m} M_j\right)^q$ from the proof of Theorem 5.2. A monomial of this polynomial with the maximum number of distinct variables corresponds to a union of at most q subsets S_j covering the same maximum number of elements in X and *vice versa*. This combined with the fact that max q-cover cannot be approximated within $1 - \frac{1}{e} + \varepsilon$ for any $\varepsilon > 0$, unless $NP = P$ (see [30]) yields the theorem. □

If we set k to n in the proof of Theorem 5.6, then we obtain a fixed-parameter reduction of the set cover problem to the problem of detecting if the constructed polynomial of degree $O(nq)$ has a monomial with precisely n distinct variables, with respect to the parameter q. Since the set cover problem is $W[2]$-complete, we obtain the following theorem.

Theorem 5.7. *Let P be a polynomial with n variables and maximum degree l represented by a monotone arithmetic circuit with polynomial in n number of gates. The problem of determining if P has a monomial with precisely n distinct variables is $W[2]$-hard for the parameter l/n.*

5.5 Extensions to the Boolean Case

By substituting $\vee$ gates for $+$ gates and $\wedge$ gates for $\times$ gates in a monotone arithmetic circuit, we obtain a corresponding monotone Boolean circuit (i.e., a directed acyclic graph whose non-leaf nodes are labeled with either $\vee$ or $\wedge$, leaves are labeled with distinct variables, and a single node is distinguished as an output node). It computes a Boolean function which is a disjunction of Boolean monomials, called *implicants*, corresponding to the monomials of the polynomial represented by the original arithmetic circuit.

Hence, all our hardness and inapproximability results on detecting monomials with at most or at least k distinct variables and their optimization versions have their analogous Boolean counterparts. The latter are simply obtained by replacing "monotone arithmetic circuits" with "monotone Boolean circuits" and "monomials in the sum-product expansion ..." with "implicants of the Boolean function computed by the input circuit" *etc*. For instance, the Boolean counterpart to Theorem 5.5 is as follows.

Corollary 5.1. *Given a monotone Boolean circuit with n input variables, and $O(n^2)$ gates, the problem of determining the smallest k such that the Boolean function computed by the circuit has an implicant with precisely k distinct variables cannot be approximated within $O(2^{\log^{1-\varepsilon} n})$ for any $\varepsilon > 0$, unless $NP \subset DTIME(n^{poly\log(n)})$. The decision version of this problem is NP-complete.*

By the reverse aritmetization of a monotone Boolean circuit consisting in replacing $\vee$ gates by $+$ gates and $\wedge$ gates by $\times$, respectively, we can also immediately extend our FPT algorithms for the arithmetic case to include the Boolean case.

As a by-product, we obtain the following theorem of interest in its own rights.

Theorem 5.8. *Given two monotone Boolean circuits with the same set of n input variables, $O(n^2)$ gates, the problem of determining if the Boolean functions computed by them are different is NP-complete.*

Proof. Let B be the monotone Boolean function computed by a given monotone Boolean circuit with n variables $x_1, \ldots, x_n$ and a single output node. Next, for

$1 \leq l < n$, let D_l be the Boolean function that is a disjunction of all monotone implicants with precisely l distinct variables in $\{x_1, \ldots, x_n\}$.

D_l can be easily computed by a monotone Boolean circuit of polynomial size as follows. Let D_l^j be the Boolean function that is a disjunction of all monotone implicants with precisely l distinct variables in $\{x_1, \ldots, x_j\}$. For $1 < l$ and $j < n$, we have $D_l^j = D_{l-1}^{j-1} \wedge x_j \vee D_l^{j-1}$.

Let $k < n$. Note that $D_{k+1} = D_{k+1}^n$ is true exactly on all assignments of variables in $\{x_1, \ldots, x_n\}$ with at least $k+1$ ones. For this reason, the functions $D_{k+1} \vee B$ and D_{k+1} are different iff B can be satisfied by an assignment with at most k ones, i.e., iff B has an implicant with at most k variables. But the latter problem is NP complete by Corollary 5.1. □

Theorem 5.8 is interesting since no negation is used by any of the two circuits, otherwise the theorem would follow trivially from the NP-completeness of the satisfiability problem.

Bibliography

[1] Hasan Abasi, Nader H. Bshouty, Ariel Gabizon, and Elad Haramaty. On *r*-simple *k*-path. In Erzsébet Csuhaj-Varjú, Martin Dietzfelbinger, and Zoltán Ésik, editors, *Mathematical Foundations of Computer Science 2014 — 39th International Symposium, MFCS 2014, Budapest, Hungary, August 25–29, 2014. Proceedings, Part II*, volume 8635 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2014.

[2] Anna Adamaszek and Andreas Wiese. A QPTAS for maximum weight independent set of polygons with polylogarithmically many vertices. In Chandra Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5–7, 2014*, pages 645–656. SIAM, 2014.

[3] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

[4] Noga Alon, Wenceslas Fernandez de la Vega, Ravi Kannan, and Marek Karpinski. Random sampling and approximation of MAX-CSPs. *J. Comput. Syst. Sci.*, 67(2):212–243, 2003.

[5] Noga Alon, Zvi Galil, and Oded Margalit. On the exponent of the all pairs shortest path problem. *J. Comput. Syst. Sci.*, 54(2):255–262, 1997.

[6] Victor Alvarez, Karl Bringmann, Radu Curticapean, and Saurabh Ray. Counting crossing-free structures. In Tamal K. Dey and Sue Whitesides, editors, *Symposuim on Computational Geometry 2012, SoCG 2012, Chapel Hill, NC, USA, June 17–20, 2012*, pages 61–68. ACM, 2012.

[7] Victor Alvarez, Karl Bringmann, Saurabh Ray, and Raimund Seidel. Counting triangulations and other crossing-free structures approximately. *Comput. Geom.*, 48(5):386–397, 2015.

[8] Victor Alvarez and Raimund Seidel. A simple aggregative algorithm for counting triangulations of planar point sets and related problems. In Guilherme Dias da Fonseca, Thomas Lewiner, Luis Mariano Peñaranda, Timothy M. Chan, and Rolf Klein, editors, *Symposuim on Computational Geometry 2013, SoCG 2013, Rio de Janeiro, Brazil, June 17–20, 2013*, pages 1–8. ACM, 2013.

[9] Matthew Andrews, Mohammad Taghi Hajiaghayi, Howard J. Karloff, and Ankur Moitra. Capacitated metric labeling. In Dana Randall, editor, *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23–25, 2011*, pages 976–995. SIAM, 2011.

[10] Nikhil Bansal and Ryan Williams. Regularity lemmas and combinatorial algorithms. *Theory of Computing*, 8(1):69–94, 2012.

[11] Andreas Björklund and Andrzej Lingas. Fast boolean matrix multiplication for highly clustered data. In Frank K. H. A. Dehne, Jörg-Rüdiger Sack, and Roberto Tamassia, editors, *Algorithms and Data Structures — 7th International Workshop, WADS 2001, Providence, RI, USA, August 8–10, 2001, Proceedings*, volume 2125 of *Lecture Notes in Computer Science*, pages 258–263. Springer, 2001.

[12] Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996.

[13] Peter J. Cameron. *Combinatorics: topics, techniques, algorithms*. Cambridge University Press, 1994.

[14] Timothy M. Chan. All-pairs shortest paths with real weights in $O(n^3/\log n)$ time. *Algorithmica*, 50(2):236–243, 2008.

[15] Timothy M. Chan. More algorithms for all-pairs shortest paths in weighted graphs. *SIAM J. Comput.*, 39(5):2075–2089, 2010.

[16] Shiri Chechik, Matthew P. Johnson, Merav Parter, and David Peleg. Secluded connectivity problems. In Hans L. Bodlaender and Giuseppe F. Italiano, editors, *Algorithms — ESA 2013: 21st Annual European Symposium, Sophia Antipolis, France, September 2–4, 2013. Proceedings*, volume 8125 of *Lecture Notes in Computer Science*, pages 301–312. Springer, 2013.

[17] Shenshi Chen. Monomial testing and applications. In Michael R. Fellows, Xuehou Tan, and Binhai Zhu, editors, *Frontiers in Algorithmics and Algorithmic Aspects in Information and Management — Third Joint International Conference, FAW-AAIM 2013, Dalian, China, June 26–28, 2013. Proceedings*, volume 7924 of *Lecture Notes in Computer Science*, pages 106–117. Springer, 2013.

[18] Zhixiang Chen, Bin Fu, Yang Liu, and Robert T. Schweller. Algorithms for testing monomials in multivariate polynomials. In Weifan Wang, Xuding Zhu, and Ding-Zhu Du, editors, *Combinatorial Optimization and Applications — 5th International Conference, COCOA 2011, Zhangjiajie, China, August 4–6, 2011. Proceedings*, volume 6831 of *Lecture Notes in Computer Science*, pages 16–30. Springer, 2011.

[19] Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. *J. Symb. Comput.*, 9(3):251–280, 1990.

[20] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms (3. ed.)*. MIT Press, 2009.

[21] Mark de Berg, Otfried Cheong, Mark van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer, 2008.

[22] Wenceslas Fernandez de la Vega and Marek Karpinski. Polynomial time approximation of dense weighted instances of MAX-CUT. *Random Struct. Algorithms*, 16(4):314–332, 2000.

[23] Wenceslas Fernandez de la Vega, Marek Karpinski, Ravi Kannan, and Santosh Vempala. Tensor decomposition and approximation schemes for constraint satisfaction problems. In Harold N. Gabow and Ronald Fagin, editors, *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22–24, 2005*, pages 747–754. ACM, 2005.

[24] Wenceslas Fernandez de la Vega, Marek Karpinski, and Claire Kenyon. Approximation schemes for metric bisection and partitioning. In J. Ian Munro, editor, *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2004, New Orleans, Louisiana, USA, January 11–14, 2004*, pages 506–515. SIAM, 2004.

[25] Wenceslas Fernandez de la Vega and Claire Kenyon. A randomized approximation scheme for metric MAX-CUT. *J. Comput. Syst. Sci.*, 63(4):531–541, 2001.

[26] Victor J. Dielissen and Anne Kaldewaij. Rectangular partition is polynomial in two dimensions but NP-complete in three. *Inf. Process. Lett.*, 38(1):1–6, 1991.

[27] Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

[28] Rodney G. Downey and Michael Ralph Fellows. *Parameterized complexity*. Springer, 2012.

[29] Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. A tight bound on approximating arbitrary metrics by tree metrics. *J. Comput. Syst. Sci.*, 69(3):485–497, 2004.

[30] Uriel Feige. A threshold of ln n for approximating set cover. *J. ACM*, 45(4):634–652, 1998.

[31] Philippe Flajolet and Marc Noy. Analytic combinatorics of non-crossing configurations. *Discrete Mathematics*, 204(1–3):203–229, 1999.

[32] Martin Fürer and Shiva Prasad Kasiviswanathan. Approximate distance queries in disk graphs. In Thomas Erlebach and Christos Kaklamanis, editors, *Approximation and Online Algorithms — 4th International Workshop, WAOA 2006, Zurich, Switzerland, September 14–15, 2006, Revised Papers*, volume 4368 of *Lecture Notes in Computer Science*, pages 174–187. Springer, 2006.

[33] François Le Gall. Powers of tensors and fast matrix multiplication. In Katsusuke Nabeshima, Kosaku Nagasaka, Franz Winkler, and Ágnes Szántó, editors, *International Symposium on Symbolic and Algebraic Computation, ISSAC 2014, Kobe, Japan, July 23–25, 2014*, pages 296–303. ACM, 2014.

[34] Leszek Gasieniec and Andrzej Lingas. An improved bound on boolean matrix multiplication for highly clustered data. In Frank K. H. A. Dehne, Jörg-Rüdiger Sack, and Michiel H. M. Smid, editors, *Algorithms and Data Structures — 8th International Workshop, WADS 2003, Ottawa, Ontario, Canada, July 30 – August 1, 2003, Proceedings*, volume 2748 of *Lecture Notes in Computer Science*, pages 329–339. Springer, 2003.

[35] Sariel Har-Peled. Quasi-polynomial time approximation scheme for sparse subsets of polygons. In Siu-Wing Cheng and Olivier Devillers, editors, *30th Annual Symposium on Computational Geometry, SoCG 2014, Kyoto, Japan, June 8–11, 2014*, page 120. ACM, 2014.

[36] Dorit S. Hochbaum. *Approximation algorithms for NP-hard problems*. PWS Publishing Co., 1996.

[37] Michael Hoffmann, Micha Sharir, Adam Sheffer, Csaba D. Tóth, and Emo Welzl. Counting plane graphs: Flippability and its applications. In Frank Dehne, John Iacono, and Jörg-Rüdiger Sack, editors, *Algorithms and Data Structures — 12th International Symposium, WADS 2011, New York, NY, USA, August 15–17, 2011. Proceedings*, volume 6844 of *Lecture Notes in Computer Science*, pages 524–535. Springer, 2011.

[38] Xiaohan Huang and Victor Y. Pan. Fast rectangular matrix multiplication and applications. *J. Complexity*, 14(2):257–299, 1998.

[39] Piotr Indyk. *High-dimensional computational geometry*. PhD thesis, Stanford University, 2000.

[40] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In Jeffrey Scott Vitter, editor, *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23–26, 1998*, pages 604–613. ACM, 1998.

[41] Piotr Indyk, S. E. Schmidt, and Mikkel Thorup. On reducing approximate mst to closest pair problems in high dimensions, 1999.

[42] Witold Lipski Jr. Finding a manhattan path and related problems. *Networks*, 13(3):399–409, 1983.

[43] Ravindran Kannan. Spectral methods for matrices and tensors. In Leonard J. Schulman, editor, *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5–8 June 2010*, pages 1–12. ACM, 2010.

[44] Richard M. Karp and J. Michael Steele. Probabilistic analysis of heuristics. *The traveling salesman problem*, pages 181–205, 1985.

[45] Marek Karpinski. Approximability of the minimum bisection problem: An algorithmic challenge. In Krzysztof Diks and Wojciech Rytter, editors, *Mathematical Foundations of Computer Science 2002 — 27th International Symposium, MFCS 2002, Warsaw, Poland, August 26–30, 2002, Proceedings*, volume 2420 of *Lecture Notes in Computer Science*, pages 59–67. Springer, 2002.

[46] Marek Karpinski, Andrzej Lingas, and Dzmitry Sledneu. Optimal cuts and bisections on the real line in polynomial time. *CoRR*, abs/1207.0933, 2012.

[47] J. Mark Keil. Polygon decomposition. *Survey, Dept. Comput. Sc. Univ. Saskatchewan*, 1996.

[48] Subhash Khot. Ruling out PTAS for graph min-bisection, densest subgraph and bipartite clique. In *45th Symposium on Foundations of Computer Science (FOCS 2004), 17–19 October 2004, Rome, Italy, Proceedings*, pages 136–145. IEEE Computer Society, 2004.

[49] Ioannis Koutis. Faster algebraic algorithms for path and packing problems. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *Automata, Languages and Programming — 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7–11, 2008, Proceedings, Part I: Tack A: Algorithms, Automata, Complexity, and Games*, volume 5125 of *Lecture Notes in Computer Science*, pages 575–586. Springer, 2008.

[50] Ioannis Koutis and Ryan Williams. Limits and applications of group algebras for parameterized problems. In Susanne Albers, Alberto Marchetti-Spaccamela, Yossi Matias, Sotiris E. Nikoletseas, and Wolfgang Thomas, editors, *Automata, Languages and Programming — 36th International Colloquium, ICALP 2009, Rhodes, Greece, July 5–12, 2009, Proceedings, Part I*, volume 5555 of *Lecture Notes in Computer Science*, pages 653–664. Springer, 2009.

[51] Andrzej Lingas. A geometric approach to boolean matrix multiplication. In Prosenjit Bose and Pat Morin, editors, *Algorithms and Computation — 13th International Symposium, ISAAC 2002 Vancouver, BC, Canada, November 21–23, 2002, Proceedings*, volume 2518 of *Lecture Notes in Computer Science*, pages 501–510. Springer, 2002.

[52] Kurt Mehlhorn. *Data Structures and Algorithms 3: Multi-dimensional Searching and Computational Geometry*, volume 3 of *EATCS Monographs on Theoretical Computer Science*. Springer, 1984.

[53] Gary L. Miller. Finding small simple cycle separators for 2-connected planar graphs. *J. Comput. Syst. Sci.*, 32(3):265–279, 1986.

[54] J. Ian Munro. Efficient determination of the transitive closure of a directed graph. *Inf. Process. Lett.*, 1(2):56–58, 1971.

[55] S. Muthukrishnan, Viswanath Poosala, and Torsten Suel. On rectangular partitionings in two dimensions: Algorithms, complexity, and applications. In Catriel Beeri and Peter Buneman, editors, *Database Theory — ICDT 1999: 7th International Conference, Jerusalem, Israel, January 10–12, 1999, Proceedings.*, volume 1540 of *Lecture Notes in Computer Science*, pages 236–256. Springer, 1999.

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

[57] Alfredo García Olaverri, Marc Noy, and Javier Tejel. Lower bounds on the number of crossing-free subgraphs of K_n. *Comput. Geom.*, 16(4):211–221, 2000.

[58] Wojciech Rytter. Fast recognition of pushdown automaton and context-free languages. *Information and Control*, 67(1–3):12–22, 1985.

[59] Harald Räcke. Optimal hierarchical decompositions for congestion minimization in networks. In Cynthia Dwork, editor, *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17–20, 2008*, pages 255–264. ACM, 2008.

[60] Jörg-Rüdiger Sack and Jorge Urrutia. *Handbook of computational geometry*. Elsevier, 1999.

[61] Raimund Seidel. On the all-pairs-shortest-path problem. In S. Rao Kosaraju, Mike Fellows, Avi Wigderson, and John A. Ellis, editors, *Proceedings of the 24th Annual ACM Symposium on Theory of Computing, May 4–6, 1992, Victoria, British Columbia, Canada*, pages 745–749. ACM, 1992.

[62] Micha Sharir and Adam Sheffer. Counting triangulations of planar point sets. *Electr. J. Comb.*, 18(1), 2011.

[63] Micha Sharir, Adam Sheffer, and Emo Welzl. On degrees in random triangulations of point sets. *J. Comb. Theory, Ser. A*, 118(7):1979–1999, 2011.

[64] Micha Sharir, Adam Sheffer, and Emo Welzl. Counting plane graphs: Perfect matchings, spanning cycles, and kasteleyn's technique. *J. Comb. Theory, Ser. A*, 120(4):777–794, 2013.

[65] Micha Sharir and Emo Welzl. On the number of crossing-free matchings, cycles, and partitions. *SIAM J. Comput.*, 36(3):695–720, 2006.

[66] R. Sibson. Locally equiangular triangulations. *Comput. J.*, 21(3):243–245, 1978.

[67] Manuel Wettstein. Counting and enumerating crossing-free geometric graphs. In Siu-Wing Cheng and Olivier Devillers, editors, *30th Annual Symposium on Computational Geometry, SoCG 2014, Kyoto, Japan, June 8–11, 2014*, page 1. ACM, 2014.

[68] Ryan Williams. Finding paths of length k in $O^*(2^k)$ time. *Inf. Process. Lett.*, 109(6):315–318, 2009.

[69] Ryan Williams. Faster all-pairs shortest paths via circuit complexity. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 – June 03, 2014*, pages 664–673, 2014.

[70] Virginia Vassilevska Williams and Ryan Williams. Subcubic equivalences between path, matrix and triangle problems. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23–26, 2010, Las Vegas, Nevada, USA*, pages 645–654. IEEE Computer Society, 2010.

[71] Raphael Yuster. Efficient algorithms on sets of permutations, dominance, and real-weighted APSP. In Claire Mathieu, editor, *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009, New York, NY, USA, January 4–6, 2009*, pages 950–957. SIAM, 2009.

[72] Uri Zwick. Exact and approximate distances in graphs — a survey. In Friedhelm Meyer auf der Heide, editor, *Algorithms — ESA 2001: 9th Annual European Symposium, Aarhus, Denmark, August 28–31, 2001, Proceedings*, volume 2161 of *Lecture Notes in Computer Science*, pages 33–48. Springer, 2001.

[73] Uri Zwick. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *J. ACM*, 49(3):289–317, 2002.