Uniwersytet Wrocławski
Wydział Matematyki i Informatyki

Równoważności pomiędzy problemami na ciągach, drzewach i grafach

Autor:
mgr Bartłomiej Dudek

Rozprawa sporządzona pod opieką promotorów:
dr Paweł Gawrychowski, prof. UWr
prof. dr hab. Krzysztof Loryś

2024

University of Wrocław
Faculty of Mathematics and Computer Science

Equivalences between Some Problems on Strings, Trees and Graphs

Author:
mgr Bartłomiej Dudek

Doctoral dissertation supervised by:
dr Paweł Gawrychowski, prof. UWr
prof. dr hab. Krzysztof Loryś

2023

Streszczenie

Tradycyjnie, w teorii złożoności algorytmów, problemy dzieli się na *łatwe* i *trudne* w zależności od tego, czy są rozwiązywalne w czasie wielomianowym. Jednak z perspektywy praktycznych zastosowań, takich jak biologia czy analiza dużych zbiorów danych, takie podejście może okazać się niewystarczające. Na przykład, dla niektórych *łatwych* problemów, obecnie najszybsze algorytmy, choć mają wielomianową złożoność asymptotyczną, to stopień tych wielomianów jest zbyt wysoki, by mogły być używane w praktyce. To rodzi naturalne pytanie o faktyczną trudność tych problemów:

Czy możemy liczyć na poprawę tych algorytmów? A może istnieje jakiś fundamentalny powód, dla którego nie ma więcej postępów?

W tej rozprawie rozważamy różne zagadnienia na ciągach, drzewach i grafach. Dla niektórych z nich prezentujemy wydajne algorytmy, a w wielu przypadkach pokazujemy również tzw. warunkowe ograniczenia dolne. Nasze badania wpisują się w obszar tzw. złożoności drobnoziarnistej, której celem jest zrozumienie zależności między problemami o złożoności wielomianowej.

Pierwszym głównym wynikiem zaprezentowanym w tej pracy jest uzyskanie podkwadratowej równoważności między wszystkimi wariantami problemu 3LDT, który jest uogólnieniem klasycznego problemu 3SUM. W 3SUM naszym celem jest znalezienie w danym zbiorze trzech elementów, które sumują się do zera. Pokazujemy, że dla wszystkich tzw. nietrywialnych współczynników $\alpha_1, \alpha_2, \alpha_3$ oraz t, wykrycie, czy dany zbiór zawiera trzy elementy x_1, x_2, x_3 takie, że $\sum_i \alpha_i x_i = t$, jest równoważne problemowi 3SUM. To również odpowiada na pytanie otwarte postawione przez Jeffa Ericksona ponad 20 lat temu, dotyczące trudności wykrywania w zbiorze trzech liczb, które tworzą ciąg arytmetyczny.

Drugim głównym wynikiem w tej pracy jest pokazanie, że następujące trzy, pozornie niezwiązane ze sobą, problemy są tak naprawdę równoważne: zliczanie 4-cykli w grafie, obliczanie odległości kwartetowej między dwoma drzewami oraz zliczanie 4-wzorów w permutacjach. Z tej równoważności wynika, że wszystkie te trzy problemy mogą być rozwiązane w tej samej złożoności czasowej (z dokładnością do czynników polilogarytmicznych), która w tej chwili wynosi $\mathcal{O}(n^{1.48})$. W przypadku dwóch ostatnich problemów, uzyskujemy poprawę w stosunku do najszybszych do tej pory znanych algorytmów, a także wspólny powód, dla którego trudno byłoby uzyskać algorytm o złożoności czasowej lepszej niż $\mathcal{O}(n^{4/3})$.

Najbardziej skomplikowaną technicznie częścią tej pracy jest redukcja z obliczania odległości kwartetowej do zliczania 4-cykli, która wykorzystuje m.in. tzw. dekompozycję top trees. Aby

lepiej przybliżyć tę strukturę, wprowadzamy ją w osobnej sekcji i pokazujemy, że spowolnienie algorytmu kompresji przy użyciu drzew top trees może prowadzić do lepszego współczynnika kompresji.

Ostatnim uzyskanym przez nas wynikiem jest redukcja problemu rozpoznawania języka bezkontekstowego w trybie “online” do wielu instancji mnożenia macierzy przez ciąg wektorów, również w trybie “online”. Dzięki temu uzyskaliśmy algorytm rozpoznawania języka bezkontekstowego “online” działający w czasie $n^3/2^{\Omega(\sqrt{\log n})}$, który jest pierwszym usprawnieniem najlepszego dotychczas znanego podejścia do tego problemu z 1985 roku.

Abstract

Traditionally, in the complexity theory, *easy* and *hard* problems are defined with respect to their polynomial time solvability. However, from the point of view of applicability in real-life applications, say biological or other big data, such an approach is too coarse. In particular, for some *easy* problems, the asymptotically fastest algorithms have polynomial time complexity with degree of the polynomial being quite high. This raises a natural question about the underlying difficulty of these problems:

Can we hope to improve these algorithms, or is there a fundamental reason for the lack of progress?

In this thesis we consider various problems on strings, trees and graphs. For some of them we show efficient algorithms, and in many cases, we also provide conditional lower bounds. That research falls within the area of fine-grained complexity whose goal is to create a map of polynomial-time problems by understanding the relationships between them.

Our first main contribution is establishing the subquadratic equivalence between all variants of the 3LDT problem, which generalizes the well-known 3SUM problem. In 3SUM, the goal is to find three elements from a given set that sum to zero. We show, that for all, so-called, non-trivial coefficients $\alpha_1, \alpha_2, \alpha_3$ and t, detecting whether a set contains three elements x_1, x_2, x_3 such that $\sum_i \alpha_i x_i = t$ is equivalent 3SUM. This also answers an open question posed by Jeff Erickson over 20 years ago regarding the difficulty of detecting three numbers in a set that form an arithmetic progression.

Our second contribution reveals that the following three seemingly unrelated problems are in fact equivalent: counting 4-cycles in a graph, computing the quartet distance between two trees and counting 4-patterns in permutations. This equivalence implies that all these problems can be solved with the same time complexity (up to polylogarithmic factors), currently $\mathcal{O}(n^{1.48})$. For the latter two problems, this allows us to improve the state-of-the-art algorithms, while also identifying a common barrier to reducing their complexity below $\mathcal{O}(n^{4/3})$.

The most technically challenging reduction in this thesis connects computing the quartet distance to counting 4-cycles, which, among other techniques, utilizes the top tree decomposition. To introduce this concept we also show that slowing down a top tree compression algorithm might lead to a better compression ratio.

As a final contribution, we reduce the Online Context-Free Recognition problem to multiple instances of Online Matrix-Vector Multiplication. This gives us an $n^3/2^{\Omega(\sqrt{\log n})}$ algorithm, which is the first improvement for this problem since 1985.

Acknowledgments

First and foremost, I would like to express my sincere gratitude to my advisors, Paweł Gawrychwyski and Krzysztof Loryś. Their guidance, support, and insightful advice during my PhD studies - and long before - have had a profound impact on both my academic journey and personal growth. Our first meeting over 16 years ago marked the start of a relationship I feel privileged to have maintained ever since. I am also incredibly fortunate to have had the opportunity to collaborate with Paweł on multiple fascinating scientific problems. He has shown me the excitement of research, and to this day, continues to guide me through the world of academia.

I would also like to thank all my co-authors, both during my PhD and before: Garance Gourdel, Piotr Ostropolski-Nalewaja, and Tatiana Starikovskaya with whom I explored algorithms on strings; Dan Alistarh, Adrian Kosowski, David Soloveichik, and Przemysław Uznański with whom I worked on population protocols; Panagiotis Charalampopoulos and Karol Pokorski with whom I tackled algorithms on graphs, and Paweł Gawrychowski with whom I approached various topics that are difficult to categorize under a single label. I learned a lot during all these collaborations.

I would also like to extend special thanks to Tatiana, who was a great host during my research visits to ENS Paris, was always very supportive, and provided a valuable different perspective in our discussions.

I would also like to thank all the faculty and colleagues at the Institute of Computer Science at the University of Wrocław for creating a wonderful atmosphere and a great place to learn and grow. In particular, I have greatly enjoyed sharing my office with Karol, and Piotrek. Although we are now in different rooms, we remain very good friends.

I would also like to thank my trip and kayak companions: Panos, Tomek and Karol for the unforgettable memories that we created during our near-conference activities.

Finally, my deepest gratitude goes to my parents, my sister and my wife. Throughout this journey, you have always remained my constant support and encouragement, and I certainly wouldn't have made it here without you.

Contents

Chapter 1

Introduction

Understanding the hardness of computational problems is one of the central directions of research in computer science. With the recent very rapid growth in the size of data available, it becomes even more pressing need to design efficient algorithms that will be able to handle data from the real-life applications. Ideally, the most desirable solutions run in time roughly proportional to the size of input. This guarantees that if we are able to store the input, probably we will be also able to process it in reasonable time. However, for multitude of problems we are still very far from obtaining such an efficient solution, even if they are very commonly used in everyday life. For example, it is already challenging to check how much two files differ or to pair people at a party so that everyone talks to someone they know. More computationally difficult questions include finding the shortest route vising a specific set of locations, or the largest group of people that everyone knows each other, just to name a few.

After designing an algorithm, the next step is to estimate its time complexity, which measures how much the algorithm's runtime increases as the input size grows. For example, if the algorithm runs in $\mathcal{O}(n^2)$ time, (where n represents the size of the input), it means that if the input grows by a factor of 10, the algorithm will take 100 times longer. This helps us understand the limits of the algorithm's efficiency and the amount of data it can process within a reasonable time. For instance, running $\mathcal{O}(n^2)$ time algorithm on the input of size $n = 10^8$ would take months and it would be entirely infeasible to run $\mathcal{O}(2^n)$ time algorithm for $n = 100$.

The traditional notion of *easy* and *hard* problems is defined with respect to polynomial time solvability. However, for many such easy problems the best known algorithms have very high complexities, making them intractable in practice, in spite of a significant effort from the algorithmic community. This suggests that some of the known algorithms are optimal (or at least very close to optimal). For example, for the 3SUM problem asking about existence of 3 numbers in a set that sum up to zero, we do not know any algorithm significantly faster than quadratic time, although this question has been already studied for decades. Unfortunately, proving unconditional statements on the optimal complexity doesn't seem within our reach, unless we are willing to work in a severely restricted model of computation. However, we can consider *conditional lower bounds* that originate from the following question:

> *If we believe that a solution for some problem is optimal, what does it tell us about other problems?*

This approach spurred a recent systematic effort to create a map of polynomial-time solvable problems and connect them to a few believable conjectures on the complexities of some basic problems, such as APSP, 3SUM or SETH. The study of such connections that help us better understand relationships between problems solvable in polynomial-time and state *conditional lower bounds* is called the area of *fine-grained complexity*. In the next section we outline some of its core results, but we also refer the reader to the comprehensive survey by Virginia Vassilevska Williams [Wil18b] for a summary of the effort in this field.

1.1 Fine-grained complexity

A relationship between two problems can be described with a *fine-grained reduction*, where an instance of one problem is reduced to one or more instances of the other problem, while having a control on the size of the obtained instance(s) and the overall running time of the reduction. This kind of reduction indicates that any improvement in the complexity of one problem will directly lead to improvements in the other. This can provide some explanation, why there has be no progress for the other problem, as it shares the same core difficulty which would have to be addressed first. When we establish reductions between two problems in both directions, the two problems are said to be *fine-grained equivalent*.

Fine-grained reductions and equivalences are inspired by the concepts of NP-hardness and NP-completeness. In the case of NP-complete problems, we know that if any of them could be solved in polynomial time, all other NP-complete problems would be solved in polynomial time as well. However, for problems within P, there exist multiple equivalence classes, but they are more *fine-grained*, which means that we are more careful about the exact exponent in the complexity of the algorithms. For instance, if any problem from a class of subcubically equivalent problems was solved in truly subcubic time, say $\mathcal{O}(n^{2.99})$, than every problem from this class would also admit a truly subcubic solution.

There are three central hypotheses in fine-grained complexity: SETH, 3SUM and APSP and we describe them in detail in the next paragraphs.

SETH. The question whether P=NP is one of the most fascinating and important questions in computer science and is one of the seven Millenium Prize Problems, solving which will be awarded with 1 million dollars[1]. The problem asks if every problem for which the solution can be verified in polynomial time also has an algorithm finding a solution in polynomial time. Currently we know hundreds of problems that are NP-hard, which means that solving any of them in polynomial time would prove that P=NP. However, we are far from finding any such algorithm and it is widely believed that P$\neq$NP.

[1] To this date, only one of these problems, the Poincaré Conjecture, has been solved.

One of the core problems in NP is k-SAT, which asks about satisfiability of a Boolean formula in Conjunctive Normal Form with clauses of at most $k \geq 3$ literals, on n variables. 2-SAT is solvable in linear time, but already for $k = 3$ the fastest known approaches run in exponential c^n time, with $c < 1.307$ for randomized solutions [HKZZ19; Sch21] and $c \leq \frac{4}{3}$ for deterministic algorithms [MS11]. Impagliazzo and Patturi posed Exponential Time Hypothesis (ETH) [IP01] which states that k-SAT cannot be solved in $2^{o(n)}$ time and immediately implies that P$\neq$NP. This hypothesis provides a reason why not only decision [IPZ01], but also various approximation [Mar07; Man17], parametrized [CCF+05; LMS18] and counting [DHM+14; BKM19] problems require exponential time. However, in the area of fine-grained complexity, the following stronger version of ETH becomes even more useful:

Hypothesis 1.1.1 (SETH [IP01; Wil05])**.** *For every $\varepsilon > 0$ there exists k such that solving k-SAT on n variables cannot be solved in $\mathcal{O}(2^{(1-\varepsilon)n})$-time by a randomized algorithm.*

First, it shows that various exponential-time problems cannot be solved in $\mathcal{O}(2^{(1-\varepsilon)n})$-time [CDL+16], for example Hitting Set or Not-All-Equal Sat. Moreover, SETH also leads to tight bounds for a number of polynomial-time algorithms. It implies that the Orthogonal Vector problem requires quadratic time [Wil05], which in turn provides a quadratic lower bound for multiple problems, for example: Unweighted Diameter [RW13], Frechet Distance [Bri14], Longest Common Subsequence and Dynamic Time Warping [ABW15b; BK17], Longest Palindromic Subsequence [BK15], Subtree Isomorphism [ABH+18], Max Inner Product [ARW17], Succinct Stable Matching [KMPS19] and Empirical Risk Minimization [BIS17].

Another notable example of application of SETH is the quadratic time lower bound for Edit Distance. The Edit Distance between two strings is a classic problem that appears for example in comparing human genomes and, for this reason, teams of researchers tried to improve the state-of-the-art quadratic-time textbook approach from early 70's by Wagner and Fisher [WF74], but with no success. Finally, in 2015 Backurs and Indyk showed that, assuming SETH, the current algorithms are optimal and a faster approach does not exist [BI15]. This result was celebrated even in The Boston Globe headlined: *For 40 Years, Computer Scientists Looked for a Solution That Doesn't Exist* [Har15] and Quanta Magazine [Pav15] and can be thought as one of the cornerstones of the fine-grained complexity.

3SUM. Already in 1995 [GO95], Gajentaan and Overmars observed that several geometry problems, such as Colinearity, Separator, Motion Planing, are solved in quadratic time and there is a common reason why we cannot improve their efficiency. They showed that solving any of these problems in subquadratic time would give us a faster algorithm for the 3SUM problem, which asks if there are 3 numbers x, y, z in a given set such that $x + y = z$. Although it does not provide a formal lower bound, it suggests any attempt to improve these geometric problems should first focus on 3SUM, which seems to be easier to work with.

In mid 1990's, first quadratic lower bounds for 3SUM [ES93; ES95; Eri99b] were found in a more restricted linear decision tree model in which we allow checking the sign of an arbitrary linear combination of at most 3 inputs. This confirmed the belief that 3SUM in fact requires quadratic time. However, in 2014 Grønlund and Pettie [GP18] showed that we shouldn't really rely on the arguments we have so far, for two reasons. First, they provided a 4-linear decision tree of depth $\mathcal{O}(n^{1.5}\sqrt{\log n})$ which means that linear decision tree complexity of 3SUM is much smaller than quadratic (in fact, later this was improved to $\mathcal{O}(n^{1.5})$ [GS17] and finally to $\mathcal{O}(n\log^2 n)$ [KLM19]), but this fact does not translate to an algorithm. Secondly, they were the first to design a (mildly) subquadratic (by less than a logarithm) algorithm for 3SUM over real numbers. To sum up, the discovery of Gajentaan and Overmars attracted a lot of attention to 3SUM, but follow-up works did not discover any truly subquadratic algorithm that runs in $\mathcal{O}(n^{2-\varepsilon})$ for some $\varepsilon > 0$, even for integers. This makes the following hypothesis widely believable:

Hypothesis 1.1.2 (3SUM [Păt10; GO95])**.** *For every $\varepsilon > 0$ 3SUM on n integers in $\{-n^3, \ldots, n^3\}$ cannot be solved in $\mathcal{O}(n^{2-\varepsilon})$ time by a randomized algorithm.*

3SUM can be solved in quadratic time by sorting the set and processing it in both directions with the two pointer technique. Moreover, if the considered numbers are integers bounded by U, with Fast Fourier Transform we can obtain algorithm running in $\mathcal{O}(n + U\log U)$-time. To this date, 3SUM admits only polylogarithmic improvements [Fre17; GS17; GP18], with the fastest approach running in $\mathcal{O}(n^2(\log\log n)^{\mathcal{O}(1)}/\log^2 n)$ by Chan [Cha18] in the real RAM model and in $\mathcal{O}(n^2/\max\{\frac{w}{\log^2 w}, \frac{\log^2 n}{(\log\log n)^2}\})$ expected time in the word RAM model with machine words of size w [BDP08]. In his seminal work [Păt10], Pătraşcu showed that 3SUM on integers is subquadratically equivalent to Convolution 3SUM, in which we are given an array A and need to find its two indices i, j such that $A[i] + A[j] = A[i+j]$. This problem has immediate quadratic time solution and is very useful in establishing reductions from 3SUM. By now we know a myriad of problems that are 3SUM-hard, especially in geometry [GO95; dBdGO97; Eri99c; AEK05; SEO03; CEH04; AHI+01; AH08; ACH+98; BH01; EHM06; BvKT98], but also in dynamic algorithms and data structures [AW14; KPP16; Păt10; Dah16; AWY18], string algorithms [AWW14; ACLL14], finding exact weight subgraphs [WW13; AL13] and in partial matrix multiplication and reporting variants of convolution [GKLP16].

APSP. The third central problem in fine-grained complexity is All-Pairs Shortest Paths (APSP) [WW18; AGW15], which asks about the distance between every pair of nodes in the given graph. This can be solved in cubic time with the textbook Floyd–Warshall algorithm [Flo62; War62] which comprises of three nested loops and dates to early 60's. Since then only polylogarithmic improvements have been achieved [Fre76; Dob90; Tak92; Tak04; Han04; Tak05; Zwi06; Han08a; Han08b; Cha08; Cha10; HT16], with the fastest approach running in time $\frac{n^3}{2^{\Omega(\sqrt{\log n})}}$ by Williams [Wil18a]. Due to the lack of progress in decreasing degree of the polynomial of n, the following hypothesis is widely believable:

Hypothesis 1.1.3 (APSP Hypothesis [RZ11; WW18])**.** *For every $\varepsilon > 0$ there exists $c > 0$ such that APSP on n nodes with edge weights from $\{-n^c, \ldots, n^c\}$ cannot be solved in $\mathcal{O}(n^{3-\varepsilon})$ time by a randomized algorithm.*

We already know a number of problems that are subcubic equivalent to APSP, for example: Negative Triangle, $(\min, +)$-Product, Metricity, Second Shortest Path [WW18], Maximum Submatrix [BDT16], Radius, Median and Betweenness Centrality [AGW15]. This equivalence means that if any of the above problems can be solved in strongly subcubic time, then all other can be as well. There are also some APSP-hard problems for which only a reduction from APSP is known, e.g. incremental (s, t)-Shortest Path, Bipartite Maximum Weight Matching [AW14] Tree Edit Distance [BGMW20] and some dynamic problems [RZ11; AW14; Dah16; AD16; AWY18; GJ21].

High level goal. One of the high level goals in fine-grained complexity is to understand relationship between the stated conjectures. To best of our knowledge, no explicit relation is known between any pair of the three main hypotheses: SETH, APSP, 3SUM, but Pătraşcu and Williams [PW10] showed that an $n^{o(k)}$ algorithm for kSUM would refute SETH. We also know that there is no deterministic reduction from SETH to 3SUM or APSP assuming Nondeterministic SETH (NSETH) [CGI+16]. Although there are some doubts on the veracity of NSETH [Wil16; Wil18b], it is a clear barrier why we don't have a reduction from SETH to 3SUM or APSP. Interestingly, there are some problems whose lower bounds are based on more than one hypothesis: Zero Edge-Weight Triangle [Păt10; WW18] and Listing Triangles on Sparse Graphs [KPP16; Păt10; WX20] are hard under both 3SUM and APSP, Local Alignment under both 3SUM and SETH [AWW14]. Finally, Triangle-Collection, Δ-Matching-Triangles and some dynamic problems (e.g. Single-Source-Reachability) are hard under all the three main hypotheses simultaneously [AWY18]!

Boolean matrix multiplication and OMv. Boolean matrix multiplication is a common tool in many polynomial-time algorithms. Given two Boolan matrices A and B of dimensions $n \times n$, the goal is to compute a matrix C such that $C[i, j] = \bigvee_k A[i, k] \wedge B[k, j]$. The straightforward approach computes this in $\mathcal{O}(n^3)$ time, but faster algebraic, Strassen-like [Str69], methods achieve significantly faster running time $\mathcal{O}(n^\omega)$ for $\omega < 2.372$ [DWZ23; WXXZ24]. These algorithms embed the Boolean semiring into the field of rationals which allows subtraction and cancellation of the terms. Then they recursively call themselves for clever combinations of submatrices of the original matrix and combine the obtained results, extensively utilizing the subtraction operation. Although these solutions are fast in theory, they are impractically slow. For this reason there is an informal notion of *combinatorial* algorithms which, informally, avoid "such" kind of techniques and are efficient in practice. It is conjectured in the BMM Hypothesis that there is no truly subcubic combinatorial algorithm for Boolean matrix multiplication [AW14; WW18] and to this date we only know subpolynomial improvements for this problem [ADKF70; BW12; Cha15; Yu18] with the fastest approach running in $n^3/2^{\Omega(\sqrt[7]{\log n})}$ time [AFK+24]. This hypothesis provides some explanation, why we do not know efficient combinatorial algorithms for multiple problems, e.g. context-free grammar parsing [Lee02], triangle listing and detection [WW18], $2k$-cycle detection [DKS17] and various dynamic problems [RZ11; AW14; CGLS18; BHG+21; JX22; HLSW23].

Due to the very vague definition of combinatorial algorithms, another hypothesis regarding matrix multiplication is desired. An important step in this direction is the introduction of the following Online Boolean Matrix-Vector Multiplication (OMv) problem: given a $n \times n$ Boolean matrix M and a sequence of n-element Boolean vectors $v_1, \ldots, v_n$, the task is to output Mv_i before seeing v_{i+1}, for all $i = 1, \ldots, n-1$. Henzinger et al. [HKNS15] discovered relations between OMv and various dynamic problems and, to that date, the best algorithm for this problem run in $\mathcal{O}(n^3/\log^2 n)$ [Wil07]. This suggested that the following conjecture is plausible:

Hypothesis 1.1.4 (OMv Hypothesis [HKNS15])**.** *For every $\varepsilon > 0$ OMv cannot be solved in $\mathcal{O}(n^{3-\varepsilon})$ time by a randomized algorithm.*

Soon after, Larsen and Williams [LW17] designed $n^3/2^{\Omega(\sqrt{\log n})}$ time algorithm. This does not refute the OMv Hypothesis, but significantly improves the known upper bound. Now the OMv Hypothesis is widely believed and is used in conditional lower bounds for multiple dynamic problems: [HKNS15; Dah16; CGLS18; LR21; GJ21; JX22].

Short cycles. Detecting and counting cycles are one of the most basic questions about a graph. Last years brought a rapid progress in understanding of the hardness of listing and detecting short cycles in a graph. Detecting a triangle in a dense graph can be easily done in $\mathcal{O}(n^\omega) = \mathcal{O}(n^{2.372})$ time by plugging in the fastest known matrix multiplication algorithm [DWZ23; WXXZ24]. Somewhat surprisingly, these two problems are, in a certain sense, equivalent: a truly subcubic algorithm for detecting triangles implies a truly subcubic algorithm for Boolean matrix multiplication [WW18]. For the more practically relevant case of a sparse undirected graph with m edges, Alon et al. [AYZ97] designed an $\mathcal{O}(m^{2\omega/(\omega+1)}) = \mathcal{O}(m^{1.41})$ time algorithm counting triangles and listing m of them. For $\omega = 2$ this runs in $\mathcal{O}(m^{4/3})$ time which is optimal under both APSP and 3SUM [KPP16; Pătt10; WX20]. Interestingly, listing triangles is equivalent to some range query problems [DKPW20].

Going one step further, 4-cycles can also be counted in $\mathcal{O}(n^\omega)$ time [AYZ97], but interestingly one can *find* a $2k$-cycle, for $k \geq 2$, in $\mathcal{O}(n^2)$ time, as shown by Yuster and Zwick [YZ97]. If the graph is given as an adjacency matrix, this is clearly optimal, but it seems plausible to conjecture that this is also optimal if the graph is given as adjacency lists [YZ97]. Returning to sparse graphs, Alon et al. [AYZ97] showed how to find a 4-cycle in $\mathcal{O}(m^{4/3})$ time, and recently Dahlgaard et al. [DKS17] provided a very nontrivial extension to finding any $2k$-cycle in $\mathcal{O}(m^{2k/(k+1)})$ time. Moreover, they showed that this is optimal, assuming that quadratic time is optimal for dense graphs, by using a general combinatorial result of Bondy and Simonovits that a graph with $m = 100kn^{1+1/k}$ edges must contain a $2k$-cycle [BS74]. Only every recently Abboud et al. showed another reason why an $m^{1+o(1)}$-time algorithm for 4-cycle detection is unlikely: $\Omega(m^{1.1194})$ time is needed for 4-cycle detection unless we can detect a triangle in $\sqrt{n}$-degree graphs in $\mathcal{O}(n^{2-\delta})$ time, which would be a breakthrough even when $\omega = 2$ [ABKZ22]. Abboud et al. and Jin and Xu showed that listing t 4-cycles cannot run in $\tilde{\mathcal{O}}(n^{2-\varepsilon} + t)$ or $\tilde{\mathcal{O}}(m^{4/3-\varepsilon} + t)$ time, under the 3SUM Hypothesis [ABF23; JX23].

1.2 Our contribution

In this dissertation we mainly focus on establishing fine-grained equivalences within two classes of problems. In Chapter 4 we answer an open question that was posed by Jeff Erickson over 20 years ago, who asked if checking if a given set contains distinct elements satisfying $x + y = 2z$ is 3SUM-hard. We show that this problem and all variants of 3LDT are equivalent to 3SUM under subquadratic reductions. Informally, this means that looking for a triple of elements satisfying any non-trivial linear combination, for example: $5x - 3y + z = 7$ is as hard as 3SUM. Moreover, we prove that they are also equivalent to their Convolution variants, where we are given an array and require that both elements and indices of their positions satisfy the same linear combination, and there is no difference if we operate on a single set (array) or three of them. We obtain this result by establishing a sequence of reductions between different variants of 3LDT and Convolution 3LDT. One important step in our reasoning is the application of Behrend's set to partition a set into $2^{\mathcal{O}(\sqrt{\log n})}$ progression-free sets, which allows us to reduce from 3-partite to 1-partite instances. Secondly, by a careful adaptation of the result by Chan and He we obtain a deterministic reduction from 3SUM to Convolution 3SUM that decreases the size of the universe of the considered numbers by a factor of n. This result is based on a joint work with Paweł Gawrychowski and Tatiana Starikovskaya that was published in STOC 2020 [DGS20] and later extended in the full version, including, for example, the equivalences between Convolution and non-Convolution variants of 3LDT.

In Chapter 6 we show that the following three seemingly unrelated problems are in fact equivalent: counting 4-cycles in a graph, computing the quartet distance between two trees and counting 4-patterns in permutations. This equivalence means that all these problems can be solved in the same time complexity (up to polylogarithmic factors), which currently is $\mathcal{O}(n^{1.48})$. By applying all the state-of-the-art results concerning counting 4-cycles in the graph we design faster algorithms for the other two problems and provide a reason, why significant further improvements would be very surprising. This conclusion is reached by a number of reductions between counting 4-cycles in different types of graphs, counting different kinds of 4-patterns in permutations and computing the quartet distance. These reductions exploit the more structured nature of the considered instances and make use of various algorithmic techniques, including orthogonal range queries, divide and conquer paradigm, top trees and heavy-light decomposition. This work is based on two results with Paweł Gawrychowski: [DG19] that shows equivalence between counting 4-cycles and computing the quartet distance and was published in STOC 2019; and [DG20] that shows equivalence between counting 4-cycles and counting 4-patters in permutations and was published in ISAAC 2020 where was awarded Best Paper.

The most technical part of this dissertation, the reduction from computing the quartet distance to counting 4-cycles is presented in Section 6.5 and uses the so-called top trees. As the introduction to the concept of top trees, in Chapter 5 we show how they can be used to obtain optimal tree compression. Somewhat surprising message from this chapter is that slowing down an algorithm might lead to better worst-case bounds on its compression-rate. This result is based on joint work with Paweł Gawrychowski that was published in CPM 2018 and later its full version was published in Theoretical Computer Science [DG24b].

In Chapter 3 we consider the Online Context-Free Recognition problem in which are given a context-free grammar and, for each prefix of the input we need to determine if it belongs to the language defined by the grammar. We show, that this problem can be reduced to multiple instances of OMv that have different sizes and run in parallel. This gives us an algorithm running in total $n^3/2^{\Omega(\sqrt{\log n})}$ time, improving upon the $\mathcal{O}(n^3/\log^2 n)$ approach from 1985. This result was published in CPM 2024 [DG24a].

In Chapter 2 we present a more detailed overview of our contribution, its motivation and the related work. The order of sections in Chapter 2 corresponds to the order of topics in this section. The main chapters are organized in a slightly different order, starting with simpler results and leaving the most complex ones for the end.

Chapter 2

More detailed overview of our contribution and previous work

In this chapter we provide a more detailed overview of our results, explain motivation behind studying these particular problems and describe a landscape of previous and following works in that areas.

2.1 3LDT

The well-known 3SUM problem is to decide, given a set X of n integers, whether any three distinct elements x_1, x_2, x_3 of X satisfy $x_1 + x_2 = x_3$. This can be easily solved in quadratic time by first sorting X, checking all candidates for x_3 and for each of them scanning the sorted sequence with two pointers. For many years no faster algorithm was known, and it was conjectured that no significantly faster algorithm exists. This assumption led to strong lower bounds for multiple problems in computational geometry [GO95] and, more recently, became a central problem in the field of fine-grained complexity [Wil15]. Furthermore, it has been proven that in some restricted models of computation 3SUM requires $\Omega(n^2)$ time [Eri99b; AC05].

However, in 2014 Grønlund and Pettie [GP18] proved that the decision tree complexity of 3SUM is only $\mathcal{O}(n^{1.5}\sqrt{\log n})$, which ruled out any almost quadratic-time lower bounds in the decision tree model. This was later improved by Gold and Sharir to $\mathcal{O}(n^{1.5})$ [GS17] and finally to $\mathcal{O}(n \log^2 n)$ by Kane et al. [KLM19]. The upper bounds for the decision tree model were later used to design a series of algorithms for a version of 3SUM in the real RAM model where the set X can also contain real numbers. In this model, Grønlund and Pettie [GP18] derived an $\mathcal{O}(n^2(\log\log n)^2/\log n)$ time randomized algorithm and an $\mathcal{O}(n^2(\log\log n/\log n)^{2/3})$ time deterministic algorithm. The best deterministic bound was soon improved to $\mathcal{O}(n^2\log\log n/\log n)$ by Gold and Sharir [GS17] and (independently) by Freund [Fre17] and then to $\mathcal{O}(n^2(\log\log n)^{\mathcal{O}(1)}/\log^2 n)$ by Chan [Cha18]. These results immediately imply similar bounds for the integer version of 3SUM. In the word RAM model with machine words of size w, Baran, Demaine and Pătraşcu [BDP08] provided an algorithm with $\mathcal{O}(n^2/\max\{\frac{w}{\log^2 w}, \frac{\log^2 n}{(\log\log n)^2}\})$ expected time.

Even though asymptotically faster than $\mathcal{O}(n^2)$, these algorithms are not strongly subquadratic, meaning working in $\mathcal{O}(n^{2-\varepsilon})$ time, for some $\varepsilon > 0$. This motivates the popular modern version of the conjecture, which is that the 3SUM problem cannot be solved in $\mathcal{O}(n^{2-\varepsilon})$ time (even in expectation), for any $\varepsilon > 0$, on the word RAM model with words of size $\mathcal{O}(\log n)$ [Păt10]. By now we have multiple examples of other problems that can be shown to be hard assuming this conjecture, especially in geometry [GO95; dBdGO97; Eri99c; AEK05; SEO03; CEH04; AHI+01; AH08; ACH+98; BH01; EHM06; BvKT98], but also in also in dynamic algorithms and data structures [AW14; KPP16; Păt10; Dah16; AWY18], string algorithms [AWW14; ACLL14; KPP16], finding exact weight subgraphs [WW13; AL13] and finally in partial matrix multiplication and reporting variants of convolution [GKLP16].

In particular, it is well-known that the 3SUM problem defined above is subquadratic-equivalent to its 3-partite variant, where we are given three sets S_1, S_2, S_3 containing at most n integers each, and must decide whether there is $x_1 \in S_1$, $x_2 \in S_2$, and $x_3 \in S_3$ such that $x_1 + x_2 = x_3$. To reduce 3-partite 3SUM to 1-partite, we can add a multiple of some sufficiently big number M to all elements in every set and take the union, for example:

$$X = \{3M + x : x \in S_1\} \cup \{M + x : x \in S_2\} \cup \{4M + x : x \in S_3\}.$$

M is chosen so that the only possibility for the three elements of X to satisfy $x_1 + x_2 = x_3$ is that they correspond to three elements belonging to distinct sets S_1, S_2, and S_3. To show the reduction from 1-partite 3SUM to 3-partite, a natural approach would be to take $S_1 = S_2 = S_3 = X$. However, this does not quite work as in the 1-partite variant we desire x_1, x_2, x_3 to be *distinct*. In the folklore reduction, this technicality is overcome using the so-called color-coding technique by Alon et al. [AYZ95].

A natural generalization of 3SUM is 3-variate linear degeneracy testing, or 3LDT [AC05]. In this problem, we are given a set X of n integers, integer parameters $\alpha_1, \alpha_2, \alpha_3$ and t, and must decide whether there are 3 distinct numbers $x_1, x_2, x_3 \in X$ such that $\sum_{i=1}^{3} \alpha_i x_i = t$. Similar to 3SUM, the 3LDT problem can be considered in the 3-partite variant as well.

A particularly natural variant of the 1-partite 3LDT problem is as follows: given a set X of n numbers, are there three distinct $x_1, x_2, x_3 \in X$ such that $x_1 + x_2 - 2x_3 = 0$? In other words, we want to check if a set contains three distinct elements that form an arithmetic progression. Following Erickson [Eri99a], we call this problem Average. It is easy to see that Average reduces to $\mathcal{O}(\log n)$ instances of 3-partite 3SUM where the j-th instance consists of the sets $S_j, X \setminus S_j, Y$ such that $S_j = \{x_i \in X : \text{ the } j\text{-th bit of } i \text{ is } 1\}$ (when $X = \{x_1, \ldots, x_n\}$) and $Y = \{2x : x \in X\}$. However, a reverse reduction seems more elusive and in fact according to Erickson [Eri99a] it is not known whether Average is 3SUM-hard[1]. This suggests the following question.

Question 2.1.1. *Can we design a reduction from 3SUM to Average? Or is Average easier than 3SUM?*

[1] Also see https://cs.stackexchange.com/questions/10681/is-detecting-doubly-arithmetic-progressions-3sum-hard/10725#10725.

A more ambitious question would be to provide a complete characterisation of all variants of 3LDT parameterized by $\alpha_1, \alpha_2, \alpha_3, t$. We know that in the restricted 3-linear decision tree model solving every variant where all coefficients α_i are nonzero requires quadratic time [Eri99b; AC05], but by now we know that this model is not necessarily the most appropriate for such problems. Because of that, we focus on the classical word RAM model, which is also the model under which most problems that are hard conditional on the 3SUM conjecture have been studied.

Question 2.1.2. *Which variants of 3LDT are easier than others in the word RAM model? Or are they all equivalent?*

In this work, we show that all non-trivial variants of 3LDT are subquadratic-equivalent in the word RAM model, which, in particular, implies that Average is subquadratic-equivalent to 3SUM. Thus, we completely resolve both Question 2.1.1 and Question 2.1.2.

In his seminal work [Păt10], Pătraşcu introduced a more structured variant of the 3SUM problem called Conv3SUM.[2] In this problem, we are given an integer array A and must decide whether there exist two distinct indices i, j such that $A[i] + A[j] = A[i + j]$. This variant is more useful for establishing reductions from 3SUM and led to a number of 3SUM-hardness results, e.g. for dynamic problems [Păt10; AW14; KPP16] and string algorithms [ACLL14; KPP16]. Pătraşcu [Păt10] showed that Conv3SUM is subquadratic-equivalent to 3SUM. Recently, Kopelowitz et al. [KPP16] refined the reduction of Pătraşcu to achieve a smaller slowdown factor, and Chan and He [CH20] presented a simple deterministic reduction. Hence it is natural to consider also Conv3LDT variants of 3LDT, in which both indices of the elements and their values need to satisfy the given linear equation. We further extend our techniques to show that all non-trivial variants of Conv3LDT are subquadratic-equivalent to each other and to non-trivial variants of 3LDT, with the size of the universe increased by the factor of n while switching from convolution to non-convolution variants of 3LDT.

We provide formal definitions and statements of our results in Section 4.1.

Follow-up work. Using divide and conquer paradigm and techniques from the conference version of our paper, Radoszewski et al. [RRS+21] showed that ConvAverage is 3SUM-hard (though increasing the size of the universe by the factor of n in the reduction). This allowed them to prove 3SUM-hardness of various problems related to Abelian squares in a text.

A direct attempt at generalizing our results to show the equivalence between kSUM and kLDT for $k > 3$ seem problematic. For concreteness, consider the variant of 4LDT in which we ask about existence of 4 distinct numbers x_1, x_2, x_3, x_4 such that $x_1 + x_2 = x_3 + x_4$. The sets avoiding existence of such four numbers are called Sidon sets and are well-studied in additive combinatorics [OBr04]. It is easy to see that any Sidon subset of $[N]$ contains at most $\mathcal{O}(\sqrt{N})$ elements. However, to apply our approach we need to partition an arbitrary set into few such subsets, which is impossible. Therefore another idea is required to prove 4SUM-hardness of

[2] According to [CH20], the earliest reference to a problem similar to Conv3SUM is from 2005: https://3dpancakes.typepad.com/ernie/2005/08/easy_but_not_th.html

4LDT. In their very recent result, Jin and Xu [JX23] found a different approach and showed that all non-trivial variants of 4LDT are 3SUM-hard. As an intermediate step, they showed 3SUM-hardness of 3SUM on Sidon sets. This required an analysis of the additive energy of a given 3SUM instance (defined as the number of quadruples $(a, b, c, d) \in X^4$ such that $a + b = c + d$): for high additive energy they applied Balog-Szemerédi-Gowers Theorem [BS94] and for smaller additive energy they applied a self reduction based on a very non-trivial hashing. 3SUM-hardness of 3SUM on Sidon sets also allowed them to establish fascinating new 3SUM-hardness results for various graph problems, like 4-Cycle Enumeration. Similar conclusions also follow from the work by Abboud et al. [ABF23] that appeared at the same time as [JX23].

Related work. Multiple variants of 3SUM have been also considered, e.g. clustered 3SUM and 3SUM for monotone sets in 2D that are surprisingly solvable in truly subquadratic time [CL15]; algebraic 3SUM, a generalization which replaces the sum function by a constant-degree polynomial [BCI+19]; and $3\mathsf{SUM}^+$ in which, given three sets A, B, S one needs to return $(A + B) \cap S$ [GP18; BDP08]. An interesting generalisation of the 3SUM conjecture states that there is no algorithm preprocessing two lists of n elements A, B in $n^{2-\Omega(1)}$ time and answering queries "Does c belong to $A + B$?" in $n^{1-\Omega(1)}$ time. Very recently, this conjecture was falsified in two independent papers [KP19; GGH+20].

Surprisingly few reductions to 3SUM are known. It is known that 3SUM is equivalent to convolution 3SUM [Păt10], which is widely used in the proofs of 3SUM-hardness. Interestingly, 3SUM that is solved in $\mathcal{O}(n^2)$ time is fine-grained equivalent to MonoConvolution, which is solved in $\mathcal{O}(n^{3/2})$ time [LPW20]. In MonoConvolution we are given three sequences a, b, c and for every index i ask if there is j such that $a_j = b_{i-j} = c_i$. In addition, Jafargholi and Viola [JV16] showed that solving 3SUM in $\tilde{\mathcal{O}}(n^{1+\varepsilon})$ time for some $\varepsilon < 1/15$ would lead to a surprising algorithm for triangle listing.

2.2 Counting 4-Cycles in Graphs

Cycles are arguably one of the most basic structures that can appear in a graph and counting or detecting them has been already studied for decades. Perhaps the most fundamental example is counting triangles, that is, 3-cycles in a simple undirected graph on n nodes. Of course, this can be easily solved in $\mathcal{O}(n^\omega) = \mathcal{O}(n^{2.38})$ time by plugging in the fastest known matrix multiplication algorithm [WXXZ24]. Somewhat surprisingly, Vassilevska Williams and Williams [WW18] proved that these two problems are, in a certain sense, equivalent: a truly subcubic algorithm for detecting triangles implies a truly subcubic algorithm for Boolean matrix multiplication. For the more practically relevant case of a sparse undirected graph with m edges, Alon et al. [AYZ97] designed an $\mathcal{O}(m^{2\omega/(\omega+1)}) = \mathcal{O}(m^{1.41})$ time algorithm for counting triangles (their algorithm is stated for finding a triangle, but can be easily extended). Going one step further, 4-cycles can also be counted in $\mathcal{O}(n^\omega)$ time [AYZ97], but interestingly one can *find* a $2k$-cycle, for $k \geq 2$, in $\mathcal{O}(n^2)$ time, as shown by Yuster and Zwick [YZ97]. If the graph is given as an adjacency matrix, this is clearly optimal, but it seems plausible to conjecture that this is also optimal if the graph is given as adjacency lists.

Conjecture 1 (Yuster and Zwick [YZ97]). *For every $\varepsilon > 0$, there is no algorithm that detects 4-cycles in a graph on n nodes in $\mathcal{O}(n^{2-\varepsilon})$ time.*

Returning to sparse graphs, Alon et al. [AYZ97] showed how to find a 4-cycle in $\mathcal{O}(m^{4/3})$ time, and recently Dahlgaard et al. [DKS17] provided a very nontrivial extension to finding any $2k$-cycle in $\mathcal{O}(m^{2k/(k+1)})$ time. Moreover, they showed that this is optimal, if one is willing to believe Conjecture 1, using a general combinatorial result of Bondy and Simonovits that a graph with $m = 100kn^{1+1/k}$ edges must contain a $2k$-cycle [BS74]. See also Abboud and Vassilevska Williams [AW14] for a similar conjecture on the complexity of detecting a 3-cycle.

Conjecture 2 (Dahlgaard et al. [DKS17]). *For every $\varepsilon > 0$, there is no algorithm that detects a 4-cycle in a graph with m edges in $\mathcal{O}(m^{4/3-\varepsilon})$ time.*

Even though we are far from proving Conjecture 2, very recently Abboud et al. [ABKZ22] showed another reason, why we do not have efficient algorithm for 4-cycle detection. They showed that there exists $\delta > 0$ such that detecting a 4-cycle in $\mathcal{O}(m^{1.1193})$ time would give a breakthrough $n^{2-\delta}$ algorithm for triangle detection in graphs of degree $\sqrt{n}$, which is not known to follow even from optimal matrix multiplication algorithms.

A related question is to find an occurrence of an induced subgraph. Vassilevska Williams et al. [WWWY15] provide a systematic study of this question for all induced four-node graphs. They also provide an algorithm that can be used to *count* occurrences of a 4-cycle (not necessarily induced) in $\mathcal{O}(m^{1.48})$ time. Also, Abboud et al. [AWY18] consider a certain generalisation of detecting 3-cycles in which the nodes are colored and we are asked to check if there exists a 3-cycle for every possible triple of distinct colors.

In this work we study two problems that are seemingly unrelated to the problem of counting cycles in a graph: counting 4-patterns in a permutation and computing quartet distance between trees. We show that they are in fact all equivalent to counting 4-cycles in sparse graphs. This means that we can reduce one problem to another with only polylogarithmic overhead which, combined with the state-of-the-art algorithm for counting 4-cycles, gives us $\mathcal{O}(n^{1.48})$-time algorithms for counting 4-patterns and computing quartet distance. Moreover, this immediately provides two reasons, why an efficient algorithm would be a breakthrough: an algorithm running in $\mathcal{O}(n^{4/3-\varepsilon})$-time for any of the two problems would improve the best approach for 4-cycle detection (which is clearly easier than counting) and an algorithm running in $\mathcal{O}(m^{1.1193})$-time would also provide an unexpectedly fast approach for triangle detection in graphs of degree $\sqrt{n}$. Now we describe each of the two problems in more detail.

2.2.1 Permutation Patterns

Permutations are arguably the most basic combinatorial objects. A natural question in discrete mathematics is to count permutations with certain properties, like consisting of a given number of cycles or having no fixed points. A whole class of such questions is obtained by fixing a permutation σ, called the pattern, and defining a permutation π to avoid σ if σ is not a subpermutation of π, or in other words if π does not contain a subsequence that is order-isomorphic to σ. For example, 21 is avoided only by $12 \dots n$. Otherwise, we say that π contains σ. One of the first results concerning pattern avoidance is by Erdős and Szekeres [ES35], who proved that every permutation of at least $(k-1)(\ell-1)+1$ elements contains either $12 \cdots k$ or $\ell \cdots 21$. Another classical result in pattern avoidance is due to Knuth [Knu68], who showed that π can be sorted by a stack if and only if π avoids 231. Together with the systematic study of patterns in permutations by Simion and Schmidt [SS85], this sparked an interest in counting and characterising permutations that avoid a given pattern (or multiple patterns). A remarkable result in this area is by Marcus and Tardos [MT04], who showed that the number of permutations of length n avoiding σ is bounded by $c(\sigma)^n$, where $c(\sigma)$ is a function independent of n. This was conjectured in early 1990s independently by Stanley and Wilf. For further discussion we refer the reader to surveys and textbooks [Bón12; Kit11; Vat15].

We approach pattern avoidance from an algorithmic perspective. We cannot hope for an efficient algorithm for arbitrary patterns, as in general it is NP-hard to check if π contains σ [BBL98] when σ is part of the input. However, if we restrict our attention to patterns of length k, we might hope to check if a given permutation on n elements avoids such pattern faster than using the trivial algorithm in $\mathcal{O}(n^k)$ time. Indeed, Albert et al. [AAAH01] and Ahal and Rabinovich [AR08] improved this complexity to $\mathcal{O}(n^{2k/3+1})$ and $n^{0.47k+o(k)}$, respectively. In a recent breakthrough result, Guillemot and Marx [GM14] developed a fixed-parameter tractable (FPT) algorithm that runs in $2^{\mathcal{O}(k^2 \log k)} \cdot n$ time. Later, by refining the proof of Marcus and Tardos [MT04], Fox [Fox13] removed the $\log k$ factor in the exponent to arrive at $2^{\mathcal{O}(k^2)} \cdot n$ complexity. For $k \geq \Omega(n/\log n)$, $\mathcal{O}(1.79^n), \mathcal{O}(1.618^n)$ and $\mathcal{O}(1.415^n)$ time algorithms are known [BL16; BKM19; GR22]. Hence even though the problem is NP-hard, by now we have a range of efficient algorithms for different special cases of checking pattern avoidance.

However, some applications bring the need to not only detect but also count occurrences of the pattern. A basic example is calculating the so-called Kendall's τ correlation coefficient [Ken38], which requires counting inversions. Generalizations of Kendall's test used in statistics require counting occurrences of larger patterns. Bergsma and Dassios [BD14] and Yanagimoto [Yan70] used patterns of length 4 in their tests. Finally, patterns of length 5 appear in the Hoeffding's dependence coefficient [Hoe48]. Also see Heller et al. [HHK+16] for a general family of such tests. We refer the reader to [EL21] for a more detailed description of the viewpoint of permutations in nonparametric statistics of bivariate data. Unfortunately, hardly any of the aforementioned algorithms for detecting patterns generalize to counting. A recent result by Berendsohn et al. [BKM19] shows that this is, in fact, inevitable, as if patterns of length k can be counted in $f(k)n^{o(k/\log k)}$ time then the exponential-time hypothesis fails. This shows that we cannot hope for a general FPT algorithm, and considering the applications in statistics we should focus on understanding the best possible exponent for small values of k.

Patterns of length k can be trivially counted in $\mathcal{O}(n^k)$ time, which was improved by Albert et al. [AAAH01] to $\mathcal{O}(n^{2k/3+1})$ and then by Berendsohn et al. [BKM19] to $\mathcal{O}(n^{k/4+o(k)})$ time.

However, it is clear that among all patterns of the same length k some are easier to count than the others. For example, occurrences of $12\cdots k$ can be easily counted in $\tilde{\mathcal{O}}(nk)$ time[3] using dynamic programming and range queries. This motivates a systematic study of the complexity of counting occurrences of different patterns of fixed small length. For $k = 2$, this is exactly the well-known exercise of counting inversions (or in other words, the pattern 21) in a permutation (or its reverse), which can be solved in $\mathcal{O}(n \log n)$ time with merge sort or in $\mathcal{O}(n\sqrt{\log n})$ in the Word RAM model [CP10]. For $k = 3$, all patterns can be counted in $\tilde{\mathcal{O}}(n)$ time by using appropriate range counting structures. For $k = 4$, various algorithms were designed to compute efficiently the Bergsma-Dassios test [BD14], which asks about the value

$$\tau^*(\pi) = \frac{\#_{1234}(\pi)+\#_{1243}(\pi)+\#_{2134}(\pi)+\#_{2143}(\pi)+\#_{3412}(\pi)+\#_{3421}(\pi)+\#_{4312}(\pi)+\#_{4321}(\pi)}{\binom{n}{4}} - \frac{1}{3}.$$

First approaches brought the complexity down to $\mathcal{O}(n^2)$ [HH16; WDL16; WDM18] and finally, Even-Zohar and Leng [EL21] observed that the patterns counted in this test possess some structural property that allows to design an $\tilde{\mathcal{O}}(n)$ time algorithm. For the remaining patterns of size 4, they obtained an algorithm working in $\tilde{\mathcal{O}}(n^{1.5})$ time. Defining the k-profile of a permutation π to be the sequence of $k!$ numbers with the number of occurrences for every possible pattern σ of length k, this brings us to the following natural open question:

Question 2.2.1 (Even-Zohar and Leng [EL21])**.** *What is the computational complexity of finding the full 4-profile of a given permutation of length n?*

In fact, Even-Zohar and Leng [EL21] showed that among all the twenty-four 4-patterns, there are eight that can be counted in $\tilde{\mathcal{O}}(n)$ time, while the remaining ones can be counted in $\tilde{\mathcal{O}}(n^{1.5})$ time. Additionally, they showed that all patterns of the second type are equivalent in terms of computational complexity, that is after counting one of them, we can retrieve all the other in $\tilde{\mathcal{O}}(n)$ time. These two types in fact coincide with the notion of concordant and discordant patterns as defined by Bergsma and Dassios [BD14]. Using the notation of Fox [Fox13], the permutation matrix of patterns of the second type contains J_2 as an interval minor. This raises the challenge of finding a reason why some 4-patterns seem harder to count than the others.

Question 2.2.2. *Why some 4-patterns seem more difficult to count than the others?*

Very recently, Beniamini and Lavee [BL24] showed efficient algorithms for counting patterns of length $5 \le k \le 7$.

Related work. Many efforts have been devoted to understand which patterns are more difficult to detect [AAAH01; ALLV16; BBL98; GV09; Iba97; YS05]. Recently Jelínek and Kynčl [JK17] established that it is possible to detect σ in polynomial time if σ avoids α, for $\alpha \in \{1, 12, 21, 132, 213, 231, 312\}$ and NP-complete otherwise. This was later strengthened by Berendsohn et al. [BKM19] by considering treewidth of the incidence graph of σ. Even though the

[3] $\tilde{\mathcal{O}}(.)$ hides factors polylogarithmic in n.

problem is NP-hard in general, more efficient algorithms are known for many families of patterns, such as vincular [BS00], bivincular [BCDK10], mesh [BC11], boxed mesh [AKV13] and consecutive [EN03]. See the survey by Bruner and Lackner [BL13] for a more detailed description of these variants.

2.2.2 Quartet Distance

Many branches of science study evolutionary relationships between objects. The canonical example is biology with species or gene relationships, but similar questions arise also in linguistics looking into related natural languages [GDG09; NWRE05; WWMA12], or archaeology studying how ancient manuscripts changed over time [Bun71]. In most cases the hierarchical structure is represented as a tree, called a phylogenetic tree in biological applications. In this work we focus on *unrooted* phylogenetic trees that describe the relationship between species mapped to its leaves without making any assumptions about the ancestry. The main goal is to understand the true relationship between the objects in question based on often incomplete or noisy data. An additional difficulty is that the obtained tree depends on the inference method (e.g. Q* [BG00], neighbor joining [SN87]) and the assumed model. See [Gus97, Chapter 17] for an overview of available models and construction methods. Consequently, we might be able to infer multiple trees that should be compared to determine if our results are consistent.

The most common approach for comparing multiple trees is to define a measure of dissimilarity between two trees. Various metrics have been already defined, e.g. the symmetric difference metric [RF79], the nearest-neighbor interchange metric [WS78], the subtree transfer distance [AS01], the Robinson and Foulds distance [RF81], the quartet distance [EMM85] and the triplet distance [Dob75]. Each of them has its particular advantages and disadvantages, see the discussion in [BD86; SP93], but the quartet-based reconstruction is perhaps the most studied (see, e.g., [BG00; BJK+99; JKL98; JWMV03; SR10; SY12; SvH96]). Most importantly, according to Bryant et al. [BTKL00], as opposed to some other methods, it is able to distinguish both between transformations that affect a large number of leaves and those that affect only a few of them. The idea is to consider the basic unit of information in such a tree, which is a subtree induced by four leaves (called a quartet). See Figure 2.1 for an illustration of the four possible topologies induced by a quartet.

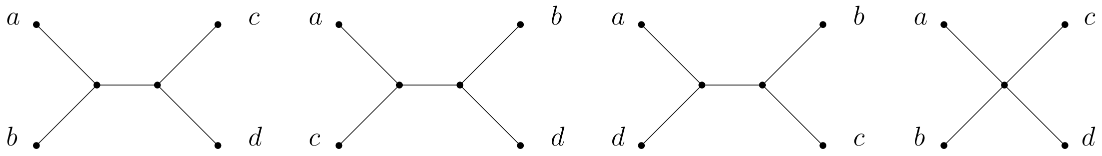


Figure 2.1: Four possible topologies of a tree induced by four leaves. Note that in the induced subtree there might be some internal nodes on each of the edges.

Definition 2.2.3. *Given two trees, each on the same set of leaves, the quartet distance is the number of quartets that are related by different topologies in both trees.*

Note that in this context we may assume that there are no internal nodes of degree 2 in both trees. We assume that the set of leaves corresponds to the full set of species.

Quartet distance has been studied from multiple angles. From the combinatorial perspective, an intriguing question is to investigate the maximum possible quartet distance between two trees on n leaves. A conjecture of Bandelt and Dress [BD86] is that this is always $(\frac{2}{3}+o(1))\binom{n}{4}$, with the best known bound being $(0.69+o(1))\binom{n}{4}$ by Alon et al. [ANS16]. From the algorithmic perspective, a long-standing challenge is to compute the quartet distance efficiently. For trees with all internal nodes of degree 3, a series of papers [BTKL00; SP93] has culminated in an $\mathcal{O}(n\log n)$ time algorithm by Brodal et al. [BFP04]. For the more challenging general case the complexity has been decreased from $\mathcal{O}(n^3)$ [CMPR05] to $\mathcal{O}(n^{2.688})$ [NKMP11] and then, for trees with all internal degrees bounded by d, further to $\mathcal{O}(n^2d^2)$ [CMPR05], $\mathcal{O}(n^2d)$ [CMP+06], $\mathcal{O}(nd^9\log n)$ [SPM+07], and finally to $\mathcal{O}(nd\log n)$ by Brodal et al. [BFM+13]. Even though some reconstruction methods produce trees with all internal degrees bounded by 3, called *fully-resolved*, trees that are not fully resolved do appear in some contexts, see e.g. [Bun71] and its refinements. This suggests the following question.

Question 2.2.4. *Can we beat $\mathcal{O}(nd\log n)$ for computing the quartet distance between two trees on n leaves and all internal degrees bounded by d?*

A related measure is the triplet distance, defined for *rooted* phylogenetic trees, where we count triplets of leaves that are related by the same topology in both trees [Dob75]. A successful line of research [BDF11; CPQ96; SBF+13] has resulted in an $\mathcal{O}(n\log n)$ time algorithm by Brodal et al. [BFM+13] for computing the triplet distance between two arbitrary trees. The algorithms designed for computing the triplet and quartet distance are based on similar ideas, see the survey by Sand et al. [SHJ+13]. Thus it is plausible that, with some additional insight, we might be able to design an $\mathcal{O}(n\log n)$ time algorithm for computing the quartet distance between two arbitrary trees, without any assumption on their degrees, similarly as for the triplet distance. Note that the fastest currently known algorithm for the general case works in $\mathcal{O}(n^2\log n)$ time [BFM+13]. This suggests the following question.

Question 2.2.5. *Can we design an $\tilde{\mathcal{O}}(n)$ time algorithm for computing the quartet distance between two trees on n leaves?*

2.2.3 Our Contribution

We present equivalence between the problems of counting 4-cycles in different classes of graphs, counting 4-patterns in different variants of permutations and computing the quartet distance between two trees. We connect complexity of the above problems by designing a sequence of reductions shown in Figure 2.2.

First, in Theorem 2.2.6 we show that we can use algorithm for counting 4-cycles in simple undirected graphs to count 4-cycles in, possibly directed, multigraphs and vice versa, with at most polylogarithmic overhead.

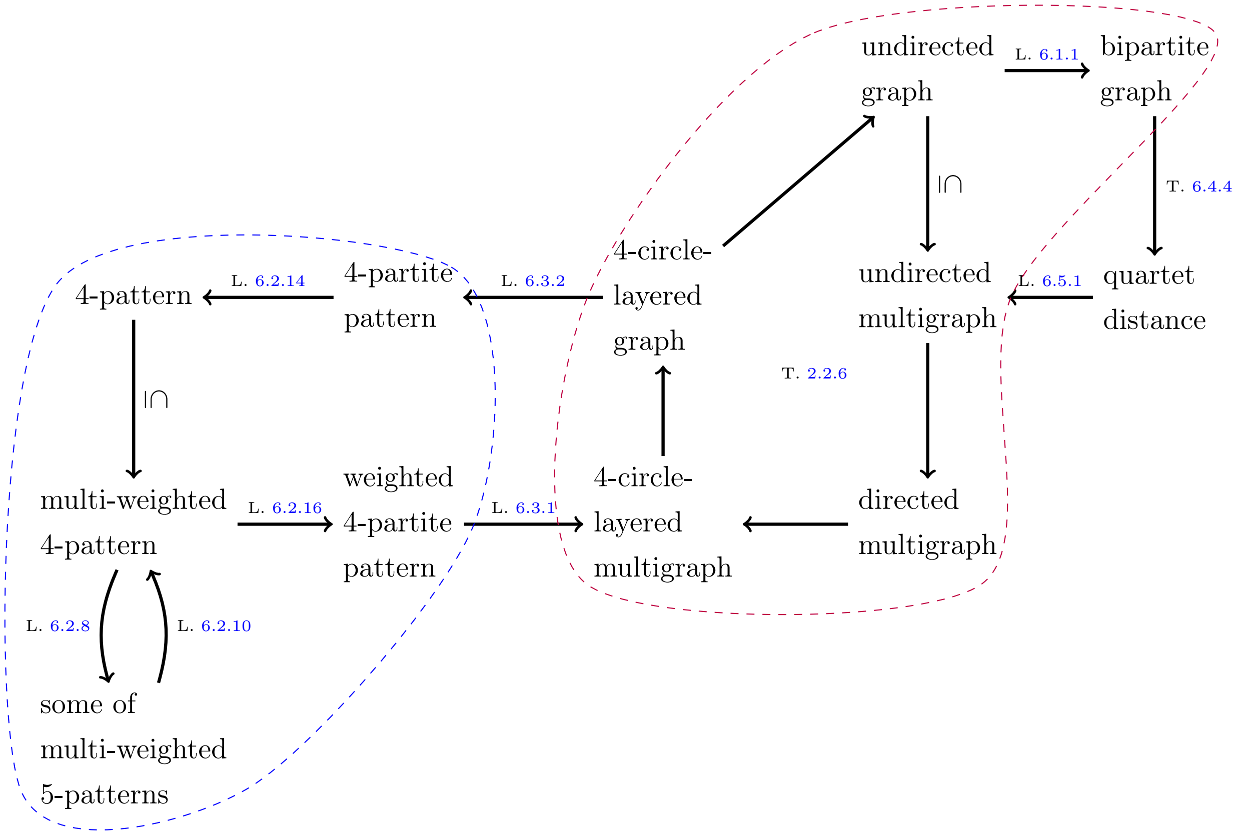


Figure 2.2: Sequence of reductions between different counting problems. Nodes in the left group correspond to the problem of counting patterns in permutations of different variants and nodes in the right group correspond to counting 4-cycles in different graphs. When we write L. X or T. Y it abbreviates reference to Lemma X or Theorem Y respectively. $A \subseteq B$ denotes that the problem A is a special case of problem B.

Theorem 2.2.6. *Counting 4-cycles in undirected multigraphs on m edges with multiplicities bounded by U can be reduced to $\mathcal{O}(\log^4 U)$ instances of counting 4-cycles in undirected simple graphs on $\mathcal{O}(m)$ edges.*

At first this might seem to be an unnecessary complication to consider counting 4-cycles in different classes of graphs, as it is plausible that the $\mathcal{O}(m^{1.48})$ algorithm of Vassilevska Williams et al. [WWWY15] can be extended, with some effort, to work for multigraphs, and used in our algorithms for counting 4-patterns or computing the quartet distance. However, it is not completely clear if every algorithm for counting 4-cycles can be similarly extended, so further improvements in the complexity of counting 4-cycles might or might not translate into an improvement for computing the quartet distance or counting 4-patterns.

The equivalence between all the three discussed problems: counting 4-cycles, counting 4-patterns and computing quartet distance gives us a better understanding of their complexity and allows us to partially answer Questions 2.2.1,2.2.2,2.2.4 and 2.2.5. We discuss our results separately for 4-patterns and quartet distance, to be able to provide a larger context and more detailed technical overview of the methods used.

Permutation Patterns

As in the previous works we divide the permutation patterns into two types and we call them trivial and non-trivial respectively. Our main contribution is a two-way reduction between counting occurrences of a non-trivial 4-pattern in a permutation and counting 4-cycles in an undirected sparse graph. This provides a reasonable answer for Question 2.2.2, as any $\tilde{\mathcal{O}}(n)$ time algorithm for such patterns would imply an exciting breakthrough for counting 4-cycles, and confirms that the two types of 4-patterns identified in the previous works are inherently different.

We partially answer Question 2.2.1 about the exact complexity of computing 4-profile of permutation of length n. Our two-way reductions imply that, by plugging in the asymptotically faster known algorithm for counting 4-cycles in a sparse graph [WWWY15], we are able to compute the full 4-profile of a permutation of length n in $\mathcal{O}(n^{1.48})$ time. In the other direction, we argue that an $\mathcal{O}(n^{4/3-\varepsilon})$ time algorithm is unlikely, as long as one is willing to believe Conjecture 2.

Our reductions regarding patterns in permutations are summarised in the left part of Figure 2.2. The main corollary from our reductions is that we can use an algorithm for counting 4-cycles in sparse graphs to count non-trivial 4-patterns and vice versa:

Theorem 2.2.7. *For every $\gamma \geq 1$, an algorithm for counting 4-cycles in a graph on m edges in $\tilde{\mathcal{O}}(m^\gamma)$ time implies an algorithm for counting non-trivial 4-patterns in a permutation of length n in $\tilde{\mathcal{O}}(n^\gamma)$ time and vice versa.*

Now we can plug in the fastest known algorithm for counting 4-cycles that runs in $\mathcal{O}(m^{\frac{4\omega-1}{2\omega+1}}) = \mathcal{O}(m^{2-\frac{3}{2\omega+1}})$ time [WWWY15]. As $\omega < 2.372$ [DWZ23; WXXZ24], we obtain a more efficient algorithm for computing the full 4-profile:

Corollary 2.2.8. *There exists an algorithm counting 4-patterns in permutation of length n in $\mathcal{O}(n^{1.48})$ time.*

We also note that a side-result of our reductions is an alternative proof for the equivalence between all the non-trivial 4- patterns, which avoids using the notion of corner tree formulas and a computer-aided argument used in [EL21]. In the other direction, as we reduce counting 4-cycles to counting a non-trivial 4-pattern, we have:

Corollary 2.2.9. *For every $\varepsilon > 0$, there exists no algorithm that can count non-trivial 4-patterns in permutation of length n in $\mathcal{O}(n^{4/3-\varepsilon})$ time unless Conjecture 2 is false.*

We stress that even though we use Conjecture 2 about detecting 4-cycles, the reduction proceeds by creating multiple instances and subtracting some of the obtained result. Hence, it does not imply anything about the complexity of detecting 4-patterns, and in fact for this problem Guillemot and Marx [GM14] showed an $\mathcal{O}(n)$ time algorithm.

We also extend the problem of counting patterns to the multi-weighted case in which every element is decorated with a tuple of weights. Then the weight of a specific occurrence of the pattern is the product of appropriate weights, which we define precisely later. We use the multi-weighted patterns to provide reductions between patterns of different length which gives us an infinite family of patterns counting which is equivalent to counting non-trivial 4-patterns.

Overview of the methods. Most of our reductions exploit the additional structure of pattern occurrences in the plane which is divided by a horizontal and a vertical line. We group the occurrences by shapes corresponding to the number of points in each quadrant and count them separately. It turns out that the hard case is when the four points are all in distinct quadrants and this is the heart of our main reductions between counting 4-patterns and 4-cycles. All other shapes can be counted in almost linear time with a careful application of range queries. To simplify the presentation, we split the reductions into many steps, between different classes of graphs and patterns so as to work with 4-partite patterns and graphs which have more structure for our application. Our reductions are based on the divide and conquer paradigm, applied to each of the four half-planes separately. For each half-plane we proceed as follows. We construct a full binary tree with leaves corresponding to coordinates of the half-plane and we group occurrences of the pattern by the lowest common ancestor (LCA) of coordinates in this half-plane.

Our reduction from counting 4-cycles to counting 4-patterns uses somewhat similar techniques to Berendsohn et al. [BKM19]. However, their approach works for arbitrary subgraphs on k nodes, which comes at a cost of increasing the size of permutation pattern and in our case would result in a pattern of 29 elements. This would not give us the desired connection between counting 4-cycles and 4-patterns, so we need a new argument tailored for 4-cycles.

Quartet Distance

We answer Question 2.2.4 and partially Question 2.2.5 by connecting the complexity of computing the quartet distance with the complexity of counting 4-cycles in a simple undirected graph. By providing reductions in both directions we show that these problems are equivalent up to polylogarithmic factors. The sequence of reductions is presented in the right part of Figure 2.2.

Theorem 2.2.10. *For every $\gamma \geq 1$, an algorithm for counting 4-cycles in a graph on m edges in $\tilde{\mathcal{O}}(m^{\gamma})$ time implies an algorithm for computing the quartet distance between trees on n leaves in $\tilde{\mathcal{O}}(n^{\gamma})$ time and vice versa.*

Our reduction from counting 4-cycles in a simple graph to computing the quartet distance implies that an $\mathcal{O}(n^{4/3-\varepsilon})$ time algorithm for computing the quartet distance between two trees on n leaves would imply a surprisingly fast $\mathcal{O}(m^{4/3-\varepsilon})$ time algorithm for counting, and thus also detecting, 4-cycles, thereby refuting Conjecture 2. Note that we create a node of the tree for every edge of the original graph and hence the complexity of the algorithm for detecting 4-cycles implied by our reduction depends on the number of edges, not nodes. This provides a reasonable explanation of why there has been no $\tilde{\mathcal{O}}(n)$ time algorithm for computing the quartet distance.

Corollary 2.2.11. *There exists no algorithm that can compute the quartet distance between trees on n leaves in $\mathcal{O}(n^{4/3-\varepsilon})$ time unless Conjecture 2 is false.*

In the other direction, the reduction from computing the quartet distance to multiple instances of counting 4-cycles in a simple graph allows us to significantly improve on the best known complexity of the former problem by plugging in the state-of-the-art algorithms for the latter problem. Recall that 4-cycles in a sparse simple graph with m edges can be counted in $\mathcal{O}(m^{1.48})$ time [WWWY15] (the $\mathcal{O}(m^{4/3})$ algorithm based on capped k-walks [DKS17] cannot be applied here, as it merely detects, but does not count the cycles). Using this algorithm we obtain that the quartet distance between two trees on n leaves can be computed in $\mathcal{O}(n^{1.48})$ time, which is a substantial improvement on the previously known quadratic time bound. Furthermore, for trees with all internal nodes having degrees bounded by d the running time of the obtained algorithm is $\tilde{\mathcal{O}}(nd^{0.96})$, and if we carefully analyse parameters of the graphs generated by the reduction and switch to the more efficient algorithms for counting 4-cycles in dense graphs, the complexity further decreases to $\tilde{\mathcal{O}}(\min\{n^{1.16}d^{0.43}, nd^{0.69}\})$.

Theorem 2.2.12. *There exists an algorithm for computing the quartet distance between two trees on n leaves and all internal nodes having degrees bounded by d in $\tilde{\mathcal{O}}(\min\{n^{1.48}, n^{1.16}d^{0.43}, nd^{0.69}\})$ time.*

An important ingredient of our proof is the reduction from counting 4-cycles in a multigraph with edge multiplicities bounded by U to $\mathcal{O}(\log^4 U)$ instances of counting 4-cycles in simple graphs of roughly the same size in Theorem 2.2.6. In the beginning of Section 2.2.3 we discuss the motivation for considering simple multigraphs separately, so as to be able to use the state-of-the-art algorithms for counting 4-cycles in simple graphs in our algorithm. Furthermore, we don't see how to provide a direct reduction from counting 4-cycles in a multigraph to computing the quartet distance, so switching to multigraphs wouldn't allow us to state an equivalence between these two problems.

We note that Jansson and Lingas [JL14] show how to reduce computing triplet distance in so called galled tree to counting triangles in many graphs. However, this results in $\mathcal{O}(n^{2.687})$ algorithm, but in fact an $\mathcal{O}(n \log n)$ solution which does not require this idea exists [JRS17]. We use a significantly different approach that gives us more control on sizes of the obtained subproblems and bound the overall running time. Furthermore we need to deal with additional technical complications due to the fact that we are working with 4-cycles instead of triangles.

Overview of the methods. The $\mathcal{O}(nd \log n)$ complexity of the fastest known algorithm for computing the quartet distance suggests that a difficult instance consists of two trees with high internal degrees, and indeed the trees obtained in our reduction have small depth but very high degrees. We start with reducing counting 4-cycles in a simple graph to counting 4-cycles in a simple bipartite graph, which is easily achieved by duplicating the nodes. Then, we construct two trees of depth 2, each consisting of the root with its children corresponding to the nodes of the graph. Finally, each edge of the graph corresponds to a leaf attached, in every tree, to the child of the root corresponding to its appropriate endpoint. The main difficulty in this reduction is that we need to carefully analyse all possible quartets and bipartite graphs on four edges to argue that, with some additional linear-time computation, we can extract the number of 4-cycles from the quartet distance.

In the other direction, our reduction is more involved. We first notice that due to the algorithm of Brodal et al. [BFM+13] we only need to show how to efficiently count quartets that are unresolved in both trees, that is, stars (the rightmost topology in Figure 2.1). As a first approximation, we could iterate over the potential central nodes of the star in both trees and create a bipartite multigraph such that counting matchings of size 4 there gives us the number of quartets with these central nodes. There are at least two issues with this approach. First, we need to prove that counting such matchings in a multigraph can be reduced to counting 4-cycles in a simple graph. Second, we cannot afford to create a separate instance for every pair of central nodes, and furthermore even if we were able to decrease their number we would still need to have some control on the total size of the obtained bipartite graphs.

We overcome the first difficulty in two steps. We begin with reducing counting matchings of size 4 in a multigraph to counting 4-cycles in a multigraph. This requires a careful analysis of all possible multigraphs on four edges and extends a similar reasoning used in the other direction of the reduction. Then, we use our reduction from counting 4-cycles in a multigraph with multiplicities bounded by U to $\mathcal{O}(\log^4 U)$ instances of counting 4-cycles in simple graphs of roughly the same size as the original multigraph.

The second difficulty is more fundamental. To avoid iterating over all pairs of central nodes, we apply a certain hierarchical decomposition of both trees known as the top tree decomposition [AHdLT05; BGLW15]. A similar decomposition has been already used by Brodal et al. [BFM+13], but we apply it to both trees simultaneously. This allows us to decrease the number of explicitly considered pairs of central nodes to only $\mathcal{O}(n \log^2 n)$ and consider the remaining pairs aggregately in batches. The remaining pairs have a simple structure, but counting them efficiently requires providing a mechanism for answering certain queries on a tree. This is implemented with the standard heavy-light decomposition and follows the high-level idea used by Brodal et al. [BFM+13].

2.2.4 Note on the Relationship between 3SUM and Counting 4-Cycles

3SUM and Counting 4-Cycles problems are solved in different time regime and are seemingly unrelated, but very recently Jin and Xu [JX23] and Abboud et al. [ABF23] showed a fascinating connection between them. They proved that under 3SUM hypothesis, no algorithm listing 4-cycles with $n^{o(1)}$ delay has $n^{2-\varepsilon}$ or $m^{4/3-\varepsilon}$ time preprocessing for any $\varepsilon > 0$. On a high level, both the above results use similar techniques: they consider the additive energy of a given 3SUM instance (defined as the number of quadruples $(a, b, c, d) \in X^4$ such that $a + b = c + d$) and for high additive energy apply Balog-Szemerédi-Gowers Theorem [BS94] and for smaller additive energy apply a self reduction based on a non-trivial hashing. This gives them a more structured instance of 3SUM, which can be reduced to an instance of triangle listing in a graph with a little number of 4-cycles (intuitively: the additive energy of the 3SUM instance depends on the number of 4-cycles in the obtained graph). Finally they reduce listing triangles in a graph with a little number of 4-cycles to listing 4-cycles, which concludes the lower bound.

Although the above description sounds not too involved, both the above works are highly non-trivial and use very interesting techniques. We were thrilled to see these results as the instances with large additive energy were the main obstacle for us to settle the hardness of 4LDT. Indeed, from the work of Jin and Xu [JX23] follows 3SUM-hardness of all non-trivial variants of 4LDT.

2.3 Top trees

Labeled trees are fundamental data structures in computer science. Generalizing strings, they can be used to compactly represent hierarchical dependencies between objects and have multiple applications. In many of them, such as XML files, we need to operate on very large trees that are in some sense repetitive. Therefore, it is desirable to design compression schemes for trees that are able to exploit this. Known tree compression methods include DAG compression that uses subtree repeats and represents a tree as a Directed Acyclic Graph [BLMN15; BGK03; FGK03], compression with tree grammars that focuses on the more general tree patterns and represents a tree by a tree grammar [BLM08; GJ16; JL16; LM06], and finally succinct data structures [FLMM09; Jac89].

In this work we analyze tree compression with top trees introduced by Bille et al. [BGLW15]. It is able to take advantage of internal repeats in a tree while supporting various navigational queries directly on the compressed representation in logarithmic time. At a high level, the idea is to hierarchically partition the tree into *clusters* containing at most two boundary nodes that are shared between different clusters. A representation of this hierarchical partition is called the top tree. Then, the top DAG is obtained by identifying isomorphic subtrees of the top tree. Bille et al. [BGLW15] proved that the size of the top DAG produced by this approach is always $\mathcal{O}(n/\log_\sigma^{0.19} n)$ for a tree on n nodes labeled with labels from Σ where $\sigma = \max\{2, |\Sigma|\}$. Furthermore, they showed that top DAG compression is always at most logarithmically worse than the classical DAG compression (and Bille et al. [BFG17] constructed a family of trees for which this logarithmic upper bound is tight). Later, Hübschle-Schneider and Raman [HR15] improved the bound on the size of the top DAG to $\mathcal{O}(\frac{n}{\log_\sigma n} \log\log_\sigma n)$ using a more involved reasoning based on the heavy path decomposition. As pointed out by Bille et al. [BGLW15], $\Omega(\frac{n}{\log_\sigma n})$ is an information-theoretical lower bound, as there are σ^n different strings of length n over Σ.

A natural question is to close the gap between the information-theoretic lower bound of $\Omega(\frac{n}{\log_\sigma n})$ and the upper bound of $\mathcal{O}(\frac{n}{\log_\sigma n} \log\log_\sigma n)$. We show that the latter is tight for the top tree construction algorithm of Bille et al. [BGLW15].

Theorem 2.3.1. *There exists an infinite family of trees on n nodes labeled from an alphabet Σ for which the size of the top DAG produced by [BGLW15] is $\Omega(\frac{n}{\log_\sigma n} \log\log_\sigma n)$ where $\sigma = \max\{2, |\Sigma|\}$.*

This answers an open question explicitly mentioned by Lohrey et al. in the arXiv version of [LRS19], who developed a different algorithm for constructing a top tree which guarantees that the size of the top DAG matches the information-theoretic lower bound. A crucial ingredient of their algorithm is a partition of the tree T into $\mathcal{O}(n/k)$ clusters of size at most k, where $k = \Theta(\log_\sigma n)$. As a byproduct, they obtain a top tree of depth $\mathcal{O}(\log n)$ for each cluster. Then they consider a tree T' obtained by collapsing every cluster of T and run the algorithm of Bille et al. [BGLW15] on T'. Finally, the edges of T' are replaced by the top trees of their corresponding clusters of T constructed in the first phase of the algorithm to obtain the top tree of the whole T. While this method guarantees that the number of distinct clusters is $\mathcal{O}(\frac{n}{\log_\sigma n})$, its disadvantage is that the resulting procedure is non-uniform, and in particular needs to be aware of the value σ and n.

We show that a slight modification of the algorithm of Bille et al. [BGLW15] is, in fact, enough to guarantee that the number of distinct clusters, and so also the size of the produced top DAG, matches the information-theoretic lower bound. The key insight actually comes from the proof of Theorem 2.3.1, where we construct a tree with the property that some of its parts are compressed much faster than the others, resulting in a larger number of different clusters. The original algorithm proceeds in iterations, and in every iteration tries to merge adjacent clusters as long as they meet some additional conditions. Surprisingly, it turns out that the information-theoretic lower bound can be achieved by slowing down this process to avoid some parts of the tree being compressed much faster than the others. Informally, we show that it is enough to require that in the t-th iteration adjacent clusters are merged only if their size is at most α^t, for some constant $\alpha > 1$. The modified algorithm preserves nice properties of the original method such as the $\mathcal{O}(\log n)$ depth of the obtained top tree and fast navigation.

Theorem 2.3.2. *Let T be a tree on n nodes labeled from an alphabet Σ and $\sigma = \max\{2, |\Sigma|\}$. The algorithm of Bille et al. [BGLW15] with the additional restriction that in the t-th iteration we merge clusters of size at most $(10/9)^t$, creates a top DAG of T of size $\Theta(\frac{n}{\log_\sigma n})$.*

2.4 Online Context-Free Recognition

Context-free languages, introduced by Chomsky already in 1959 [Cho59], are one of the basic concepts considered in formal languages, with multiple applications in programming languages [ASU86], NLP [JM09], computational biology [DEKM98], and databases [KSSY13]. A context-free language is a language generated by a context-free grammar, meaning that each production rule is of the form $A \to \alpha$, where A is a non-terminal symbol, and α is a string of terminal and non-terminal symbols (possibly empty). It was already established by Chomsky [Cho59] that, without decreasing the expressive power, we can assume that the productions are of the form $A \to a$ and $A \to BC$, where A, B, C are non-terminal symbols, and a is a terminal symbol. From an algorithmic point of view, the natural (and very relevant with respect to the possible applications) question is whether, given such a grammar G and a string $w[1..n]$, we can efficiently check if $w \in \mathcal{L}(G)$. A simple application of the dynamic programming paradigm shows that this is indeed possible in $\mathcal{O}(n^3)$ time (ignoring the dependency on the size of the grammar). This is usually called the Cocke–Younger–Kasami (CYK) approach [CS70; Kas65; You67]. In 1975, Valiant [Val75] designed a non-trivial algorithm that solves this problem in $\mathcal{O}(BM(n))$ time, where $BM(n)$ denotes the complexity of multiplying two (Boolean) $n \times n$ matrices. Plugging in the currently best known bounds, $BM(n) = \mathcal{O}(n^{\omega})$, where $\omega < 2.372$ [DWZ23; WXXZ24]. See [Har78] for a somewhat more approachable description of Valiant's algorithm, and [Ryt95] for a very elegant simplification (achieving the same running time). This is of course a somewhat theoretical result, and given the practical nature of the problem it is not surprising that other approaches have been developed [PK09; RSCJ10; SBMN13; CSC13], with high worst-case time complexities, but good behaviour on instances that are relevant in practice. However, the worst-case time complexity has not seen any improvement. In 2002, Lee [Lee02] showed a conditional lower bound that provides some explanation for this lack of improvement: multiplying two (Boolean) $n \times n$ matrices can be reduced to parsing a string of length $\mathcal{O}(n^{1/3})$ for a grammar of size $\mathcal{O}(n^2)$. This does exclude a combinatorial $\mathcal{O}(gn^{3-\varepsilon})$ algorithm for parsing (more general problem than recognition), where g is the size of the grammar, but does not contradict the existence of e.g. $\mathcal{O}(g^2 n)$ time algorithm. However, in 2015 Abboud, Backurs, and Vassilevska Williams [ABW15a] showed a more general conditional lower bound: even for constant-size grammars, any improvement on the complexity of Valiant's recognition algorithm implies a breakthrough for the well-known k-Clique problem.

In some applications, the input string $w[1..n]$ is given character-by-character, and for each prefix $w[1..i]$ we should decide if it belongs to $\mathcal{L}(G)$ before reading the next character. This is known as the online CFG recognition. The goal is to minimise the total time to process all the characters. It is not hard to adapt the CYK approach to work in $\mathcal{O}(n^3)$ total time for this variant, but this seems difficult (or perhaps impossible) for Valiant's algorithm. Graham, Harrison, and Ruzzo [GHR80] designed a (slightly) subcubic algorithm, and Rytter [Ryt85] further improved the complexity to $\mathcal{O}(n^3/\log^2 n)$. Surprisingly, no further improvements were achieved. On the lower bound, it is known that on a Turing machine, $\Omega(n^2/\log n)$ steps are required [Sei86; Gal69]. This is however quite far from the upper bound, and assumes a somewhat restricted model of computation, and brings the natural question of understanding if a faster algorithm exists.

As the complexity of the offline CFG recognition is known to be close to that of (Boolean) matrix multiplication, it is natural to seek a connection between the complexity of its online variant with the so-called online matrix multiplication. As a tool for unifying the complexities of different dynamic problems, Henzinger, Krinninger, Nanongkai, and Saranurak [HKNS15] considered the Online Matrix-Vector Multiplication problem:

Definition 2.4.1 (Online Matrix-Vector Multiplication (OMv))**.** *Given a matrix $M \in \{0,1\}^{n\times n}$, and a sequence of vectors $v_1, \dots, v_n \in \{0,1\}^n$, the task is to output Mv_i before seeing v_{i+1}, for all $i = 1, \dots, n-1$.*

and conjectured that no $\mathcal{O}(n^{3-\epsilon})$ time algorithm exists (with the best known upper bound at the time being $\mathcal{O}(n^3/\log^2 n)$):

Hypothesis 1.1.4 (OMv Hypothesis [HKNS15])**.** *For every $\varepsilon > 0$ OMv cannot be solved in $\mathcal{O}(n^{3-\varepsilon})$ time by a randomized algorithm.*

Surprisingly, Larsen and Williams [LW17] were soon able to construct a faster $n^3/2^{\Omega(\sqrt{\log n})}$ time algorithm. This does not refute the OMv hypothesis, but significantly improves the known upper bound, essentially by saying that we can shave any number of logarithms from the time complexity.

Theorem 2.4.2 ([LW17])**.** *There exists a randomized algorithm for OMv that runs in total $n^3/2^{\Omega(\sqrt{\log n})}$ time and succeeds with high probability*[4].

This suggests the possibility of leveraging the progress on the complexity of Online Matrix-Vector Multiplication to improve the complexity of online CFG recognition to improve on the $\mathcal{O}(n^3/\log^2 n)$ time complexity from 1985.

Our contribution. We show that it is possible to use efficient OMv multiplication to speed up online context-free recognition:

Theorem 2.4.3. *Let G be a context-free grammar, and w be a length-n string, revealed one character at a time. There exists a randomized algorithm that determines, after having seen $w[t]$, if $w[1..t] \in \mathcal{L}(G)$, in $n^3/2^{\Omega(\sqrt{\log n})}$ total time and succeeds with high probability.*

Our solution is based on the classical CYK dynamic-programming approach from 1960s [CS70; Kas65; You67] in which we calculate the set of non-terminals deriving each of the infixes of w. In order to avoid the $\mathcal{O}(n^2)$ time for processing a new character $w[t]$, we maintain a division of the current prefix into segments of lengths that are powers of 2 present in the binary representation of t. For each of the segments, we build a structure responsible for processing suffixes $w[i..t]$ that start within the segment and end at t. We extensively use the approach from Theorem 2.4.2 for OMv, with a slight adaptation to matrices that grow in time. More precisely, we show that we can process a sequence of vectors $v_1, q_1, v_2, q_2, \ldots$ where $|q_i| = i$ in which we need to calculate $(v_1, \ldots, v_i) \times q_i$ online, before seeing v_{i+1}. This requires one more step of dividing the range of columns into segments of lengths that are powers of 2, and applying the structure from Theorem 2.4.2 for each of the segments separately. This results in the same running time as in the standard OMv problem, in which the matrix we multiply with does not change.

We note that our algorithm does not need to know the value of n in advance. In fact, our proof of Theorem 2.4.3 can be modified to show that the amortised time for processing the t-th character is $\mathcal{O}(t^2/2^{\Omega(\sqrt{\log t})})$, so in particular after having seen $w[t]$ we know whether $w[1..t] \in \mathcal{L}(G)$, with the total time spent on $w[1], w[2], \ldots, w[t]$ being $\mathcal{O}(t^3/2^{\Omega(\sqrt{\log t})})$, for every $t = 1, 2, \ldots$.

[4] By succeeding with high probability we mean that there exists a constant $c > 0$ such that the algorithm succeeds with probability at least $1 - 1/n^c$.

Chapter 3

Online Context-Free Recognition in OMv Time

3.1 Preliminaries

Consider a context-free grammar $G = (V_N, V_T, P, S)$. Without loss of generality we assume that G is in Chomsky normal form [Cho59; Sip97], that is every production in P is either $A \to BC$ or $A \to c$ for $A, B, C \in V_N$ and $c \in V_T$. By $v \overset{\star}{\Rightarrow} s$ we denote that string s can be derived from non-terminal v in the grammar G.

We are given a string w of length n character-by-character and for each $t = 1..n$ need to decide if the string $w[1..t]$ belongs to $\mathcal{L}(G)$ or not. For every t, the answer should be provided before reading the $(t+1)$-th character and we call such a procedure *online*. Our algorithm will compute the set of all non-terminals that produce every infix of w: $U[i,j] = \{v \in V_N : v \overset{\star}{\Rightarrow} w[i..j]\}$. Then the t-th bit of the output is whether S belongs to $U[1,t]$ or not.

Our approach has polynomial dependence on the size of the grammar G, which we omit while stating the complexity of the parsing algorithm.

In the analysis of our algorithm we will consider sums of non-constant number of distinct expressions containing the Ω notation. Unless stated otherwise, all the Ω symbols within one sum correspond to the same function, namely there exists one constant bounding all the expressions at the same time. An example of such sum appears in the following lemma that will be useful in the next section:

Lemma 3.1.1. *For every constant $a > 0$, we have $\sum_{k=0}^{\log n} 2^{ak-\Omega(\sqrt{k})} = n^a/2^{\Omega(\sqrt{\log n})}$.*

Proof. Let $t = \log n$. As we discussed before, various Ω symbols correspond to one particular Ω bound, which means that we can read the expression $(*) = \sum_{k=0}^{t} 2^{ak-\Omega(\sqrt{k})}$ as: there exists a constant $c > 0$ such that $(*) \le \sum_{k=0}^{t} 2^{ak-c\sqrt{k}}$. First we show for which $0 \le k < t$ we can upper bound the k-th summand by the last element from the sum:

$$ak - c\sqrt{k} < at - c\sqrt{t}$$
$$\Updownarrow$$
$$c(\sqrt{t} - \sqrt{k}) < a(t-k) = a(\sqrt{t} - \sqrt{k})(\sqrt{t} + \sqrt{k})$$
$$\Updownarrow$$
$$\frac{c}{a} - \sqrt{t} < \sqrt{k}$$

So in particular, for $k \geq k_0 = (\frac{c}{a})^2$ we have that $ak - c\sqrt{k} \leq at - c\sqrt{t}$. For $k < k_0$ we have $2^{ak-c\sqrt{k}} < 2^{ak_0}$, so:

$$(*) \leq k_0 \cdot 2^{ak_0} + \sum_{k=k_0}^{t} 2^{at-c\sqrt{t}} \leq \mathcal{O}(1) + t \cdot 2^{at-c\sqrt{t}} = \mathcal{O}(2^{at-0.9c\sqrt{t}}) = 2^{at-\Omega(\sqrt{t})}. \quad \square$$

3.2 Parsing context-free grammars online

Our algorithm processes characters from the input one-by-one. While processing the t-th character it has already computed $U[i,j]$ for $1 \leq i \leq j < t$ and needs to compute $U[i,t]$ for $1 \leq i \leq t$. We maintain a division of the interval $[1..(t-1)]$ into $c = \mathcal{O}(\log t)$ intervals: $[e_1, e_2), [e_2, e_3), \ldots, [e_c, e_{c+1})$ where $e_1 = 1, e_{c+1} = t$ and lengths of the intervals are exactly the powers of 2 in the binary representation of $t-1$, in the decreasing order. On a high level, for the j-th interval there is a data structure X_j responsible for computing $U[i,t]$ for $e_j \leq i < e_{j+1}$, based on the outputs from $X_{j'}$ for $j' > j$. We call such a data structure a *process*. We say that the *size* of process X_j is the length of the interval it corresponds to, that is $|[e_j, e_{j+1})| = e_{j+1} - e_j$. The processes are created and removed following the binary representation of t, and a process for interval $[a, a+2^k)$ exists only for $t = a + 2^k, \ldots a + 2^{k+1} - 1$. In the following theorem we describe the calculations performed in each of the processes.

Theorem 3.2.1. *Let $\mathcal{I} = [p, p+s)$ be an interval of positions from w. Consider the following sequence Q of at most s queries $Q_{p+s}, Q_{p+s+1}, \ldots$: in the t-th query we are given set $Q_t = \{(i,v) : v \overset{\star}{\Rightarrow} w[i..t], i \in [p+s, t]\}$ and need to compute $A_t = \{(i,v) : v \overset{\star}{\Rightarrow} w[i..t], i \in \mathcal{I}\}$. There exists a randomized algorithm answering online all queries from Q in total $s^3/2^{\Omega(\sqrt{\log s})}$ randomized time that succeeds with high probability.*

Before we prove the above theorem, we show how to apply it to obtain efficient algorithm for parsing context-free grammars online.

Theorem 2.4.3. *Let G be a context-free grammar, and w be a length-n string, revealed one character at a time. There exists a randomized algorithm that determines, after having seen $w[t]$, if $w[1..t] \in \mathcal{L}(G)$, in $n^3/2^{\Omega(\sqrt{\log n})}$ total time and succeeds with high probability.*

Proof. Consider Algorithm 1. We show that it correctly parses all prefixes of w online, in the desired time complexity. First, we show that the operations in Algorithm 1 satisfy the

Algorithm 1 Parsing context-free grammar online

```
    Input: Context-free grammar $G = (V_N, V_T, P, S)$

 1: $B := [\,]$                                    ▷ 1-based list of processes $B^1, B^2, \ldots$
 2: for $t = 1, 2, \ldots$ do
 3:     $Q_t := \{(t, v) : (v \to w[t]) \in P\}$
 4:     for $j = |B|$ to 1 do
 5:         $A := B^j.query(Q_t)$
 6:         $Q_t := Q_t \cup A$
 7:     let $r$ be maximal such that $\sum_{i=0}^{r-1} |B^{|B|-i}| = 2^r - 1$
 8:     remove the last $r$ processes from $B$
 9:     add new process for $\mathcal{I} = [t + 1 - 2^r, t + 1)$ to the end of $B$
10:     output whether $(1, S) \in Q_t$
```

requirements on queries described in Theorem 3.2.1. Indeed, we always create a process β with $\mathcal{I} = [t + 1 - 2^r, t + 1)$ and the subsequent queries are $Q_{t+1}, Q_{t+2}, \ldots$, so in particular the first query concerns the position right after the end of $\mathcal{I}$, as required. Observe that due to line 7 the sequence of sizes of the processes follows the binary representation of t so we create a process of size 2^k for t such that $t \equiv 2^k \pmod{2^{k+1}}$ and the last query that we possibly process at β is Q_{t+2^k}, so β is queried at most $t + 2^k - (t + 1) + 1 = 2^k = |\beta|$ times.

Now we calculate the complexity of the algorithm. We create a new process of size 2^k exactly $\lfloor \frac{n+2^k}{2^{k+1}} \rfloor < n/2^k$ times. Each process of size 2^k answers at most 2^k queries so we can directly apply Theorem 3.2.1 to bound the total running time of preprocessing and all queries processed by the process. Hence the total running time of the algorithm is upper bounded by:

$$\sum_{k=0}^{\log n} \frac{n}{2^k} \cdot \left(2^k\right)^3 / 2^{\Omega(\sqrt{k})} = n \cdot \sum_{k=0}^{\log n} 2^{2k - \Omega(\sqrt{k})} = n^3 / 2^{\Omega(\sqrt{\log n})}.$$

The last step follows by Lemma 3.1.1 and the claim holds. □

Proof of Theorem 3.2.1

In order to prove Theorem 3.2.1, we need to introduce some notation and insights following Rytter's variant of the Valiant's offline parser of context-free grammars [Ryt95]. We will operate on matrices of binary relations over the set of non-terminals V_N and we call such matrices *relational*. Formally, every element of a relational matrix is of the form $\{0, 1\}^{V_N \times V_N}$. To simplify the notation, our relational matrices will be indexed by intervals $\mathcal{J}_1, \mathcal{J}_2 \subseteq [1, n]$ of consecutive numbers, corresponding to substrings of the input string w. We define $\otimes$-multiplication of matrices A, B with indices $\mathcal{J}_a \times \mathcal{J}_c$ and $\mathcal{J}_c \times \mathcal{J}_b$ respectively, as:

$$(A \otimes B)[i, j]^{X,Y} = \bigvee_{\substack{k \in \mathcal{J}_c \\ Z \in V_N}} A[i, k]^{X,Z} \cdot B[k, j]^{Z,Y} \quad \text{for } i \in \mathcal{J}_a, j \in \mathcal{J}_b, X, Y \in V_N$$

Similarly, we define *relational vectors* as vectors of subsets of V_N, that is they are of the form: $\{0,1\}^{V_N}$, and matrix-vector product $M \otimes F$ of matrix M (indexed with $\mathcal{J}_a \times \mathcal{J}_c$) and vector F (indexed with $\mathcal{J}_c$) as:

$$(M \otimes F)[i]^X = \bigvee_{\substack{k \in \mathcal{J}_c \\ Z \in V_N}} M[i,k]^{X,Z} \cdot F[k]^Z \quad \text{for } i \in \mathcal{J}_a, X \in V_N$$

Lemma 3.2.2 ([Ryt85])**.** *We can compute relational matrix-matrix $\otimes$-product in $|V_N|^3$ multiplications of two Boolean matrices and relational matrix-vector $\otimes$-product in $|V_N|^2$ multiplications of a Boolean matrix and a vector. The Boolean matrices and vectors that we multiply have the same size as the relational ones.*

Proof. By definition of $\otimes$-product, in order to multiply two relational matrices we iterate over all triples X, Y, Z of non-terminals, create Boolean matrices $A'^{X,Z}, B'^{Z,Y}$ where $A'^{X,Z}[i,j] = A[i,j]^{X,Z}$ and $B'^{Z,Y}[i,j] = B[i,j]^{Z,Y}$ and calculate $A' \cdot B'$ using the standard Boolean $(\vee, \wedge)$-product. Then $(A \otimes B)[i,j]^{X,Y} = \bigvee_{Z \in V_N} (A'^{X,Z} \cdot B'^{Z,Y})[i,j]$.

Matrix-vector $\otimes$-multiplication can be calculated analogously. □

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

Theorem 3.2.1. *Let $\mathcal{I} = [p, p+s)$ be an interval of positions from w. Consider the following sequence Q of at most s queries $Q_{p+s}, Q_{p+s+1}, \ldots$: in the t-th query we are given set $Q_t = \{(i,v) : v \overset{\star}{\Rightarrow} w[i..t], i \in [p+s, t]\}$ and need to compute $A_t = \{(i,v) : v \overset{\star}{\Rightarrow} w[i..t], i \in \mathcal{I}\}$. There exists a randomized algorithm answering online all queries from Q in total $s^3/2^{\Omega(\sqrt{\log s})}$ randomized time that succeeds with high probability.*

Recall that $G = (V_N, V_T, P, S)$ is the considered grammar. Whenever we refer to $w[i, i-1]$ for any i, we mean an empty string.

Preprocessing. During the preprocessing phase we first run Rytter's algorithm [Ryt95] on $w[p..p+s-1]$ and compute $U[i,j]$ for all $p \le i \le j < p+s$ in $\mathcal{O}(s^\omega)$ time[1]. Based on that we define a relational matrix V with rows and columns indexed with $p..(p+s)$ by setting

$$V[i,j]^{X,Y} = 1 \iff \exists_{Z \in V_N} \Big((X \to ZY) \in P \wedge Z \overset{\star}{\Rightarrow} w[i..j-1] \Big) \quad \text{for } p \le i \le j \le p+s$$

Informally, this means that we can extend "to the left" every infix of w that starts at position j and can be derived from Y to an infix that starts at position i, ends at the same position and that can be derived from X. For empty infixes we set $V[i,i]^{X,Y} = 1 \iff X = Y$. When we do not specify the value of some entries of a matrix, it means that there are all zeros in that entry. For instance, for $i > j$ in V we have $V[i,j]^{X,Y} = 0$ for all $X, Y \in V_N$.

Now we calculate $V^\star = V^s$ with exponentiation by squaring, using $\otimes$-product at every step in total $\mathcal{O}(s^\omega \log s) = \tilde{\mathcal{O}}(s^\omega)$ time, by Lemma 3.2.2. Observe that $V^\star$ describes all possibilities

[1] In [Ryt95] is computed $VALID(k,\ell) = \bigcup_{k \le i \le j \le \ell} \{(A,i,j) : A \in U[i,j]\}$.

of extending an infix “to the left” at most s times. As $j - i \leq s$, we never need more than s steps to extend an infix starting at position j to an infix starting at position i and then:

$$V^\star[i,j]^{X,Y} = 1 \iff \exists_{\substack{k_1<\ldots<k_r \\ k_1=i,k_r=j \\ Z_1,\ldots,Z_r\in V_N \\ Z_1=X,Z_r=Y}} \left(\forall_{1\leq e<r} V[k_e,k_{e+1}]^{Z_e,Z_{e+1}} = 1\right) \quad \text{for } p \leq i \leq j \leq p+s$$

See Figure 3.1 for an illustration.

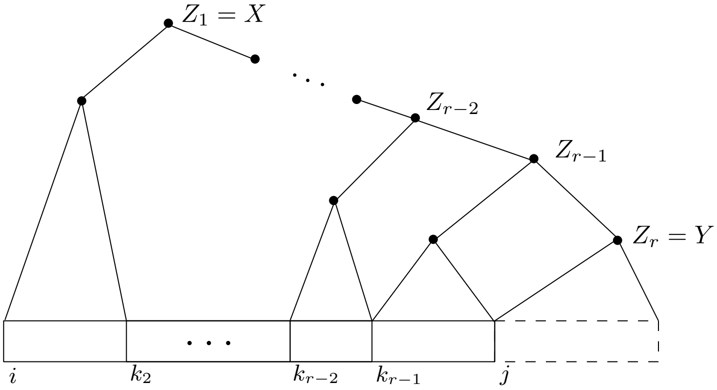


Figure 3.1: Illustration of the definition of $V^\star$. Note that we do not specify the endpoint of the last string, starting at position j and derived from Y, because we are only interested in the possible extensions “to the left” from such a string.

Invariant. During the process of answering queries, before receiving a subsequent query Q_t, we maintain a relational matrix H with similar properties as V, with rows $p..(p+s)$, but with columns $(p+s)..t$, that is:

$$H[i,j]^{X,Y} = 1 \iff \exists_{Z\in V_N}\left((X \to ZY) \in P \wedge Z \overset{\star}{\Rightarrow} w[i..j-1]\right) \quad \text{for } p \leq i \leq p+s \leq j \leq t$$

This matrix also describes extensions “to the left”, but from an infix starting at position $j \geq p+s$ to an infix starting at position $i \leq p+s$. In order to satisfy the invariant, at the end of preprocessing we initialize $H[i,p+s] = V[i,p+s]$ for $p \leq i \leq p+s$.

Query. From the input set Q_t we create a relational vector $F[(p+s)..t]$ such that

$$F[j]^Y = 1 \iff Y \overset{\star}{\Rightarrow} w[j..t] \iff (j,Y) \in Q_t.$$

Let $A = H \otimes F$. Then $A[i]^X = 1 \implies X \overset{\star}{\Rightarrow} w[i..t]$ for $p \leq i \leq p+s$. However, this is not an equivalence yet, because we need to consider a larger number of extensions “to the left” using infixes fully contained in $\mathcal{I} = [p, p+s)$. For that purpose we use matrix $V^\star$ and compute $A' = V^\star \otimes A$. Then we have $A'[i]^X = 1 \iff X \overset{\star}{\Rightarrow} w[i..t]$ for $p \leq i \leq p+s$ and we can construct the desired set A_t that can be returned from the procedure. As the last step of processing the query, we update matrix H by adding $(t+1)$-th column by definition: $H[i,t+1]^{X,Y} = 1 \iff \exists_{Z\in V_N}(X \to ZY) \in P \wedge A'[i]^Z = 1$.

Running time. The only operations that can take more than $\mathcal{O}(s)$ time in the above procedure are the matrix-vector $\otimes$ multiplications $H \otimes F$ and $V^\star \otimes A$. Recall that by Lemma 3.2.2 it suffices to show how to perform these operations efficiently for matrices and vectors over the Boolean semiring, not the relational ones. In the case of $V^\star \otimes A$ we have one matrix $V^\star$ subsequently multiplied by different vectors A, so we can directly apply Theorem 2.4.2 and process online all the queries in total $s^3/2^{\Omega(\sqrt{\log s})}$ time.

For the multiplications $H \otimes F$ we need a slightly different approach, because the matrix H changes in time. Similarly as in Algorithm 1 we will divide the columns of H into intervals following the binary representation of the width of H and split H into a number of smaller square matrices. For each of the small matrices we will use the algorithm for OMv from Theorem 2.4.2.

Lemma 3.2.3. *Consider the sequence of at most s operations, where in the j-th one we are given a binary vector v_j of length s and a binary vector q_j of length j and need to calculate $x_j = M_j \cdot q_j$ where M_j is the matrix with s rows and columns $v_1, \dots, v_j$. There exists a randomized algorithm answering online all the queries in total $s^3/2^{\Omega(\sqrt{\log s})}$ time that succeeds with high probability.*

Proof. Similarly as in Algorithm 1, we maintain a partition of the interval $[1..j]$ into $c = \mathcal{O}(\log j)$ intervals: $\mathcal{E}(j) = [e_1, e_2), [e_2, e_3), \dots, [e_c, e_{c+1})$ where $e_1 = 1, e_{c+1} = j + 1$ and lengths of the intervals are the powers of 2 in the binary representation of j, with $|[e_1, e_2)|$ being the largest one. Intervals correspond to subranges of columns of M_j and for an interval of length 2^k we divide its $s \times 2^k$ submatrix into $s/2^k$ square matrices of size $2^k \times 2^k$. For each such matrix we create a data structure for OMv multiplication, by Theorem 2.4.2.

In order to process a query, we first add the new column v_j, update the structure of intervals from $\mathcal{E}(j-1)$ to $\mathcal{E}(j)$ and run preprocessing for each of the newly-created matrices. To answer the query we divide q_j according to $\mathcal{E}(j)$ into vectors $q_j^1, \dots, q_j^c$ and multiply each vector q_j^i by all the matrices of size $|q_j^i| \times |q_j^i|$ and combine the results in one vector y_j of length s, see Figure 3.2. Then $x_j = \bigvee_{i=1}^c y_i$.

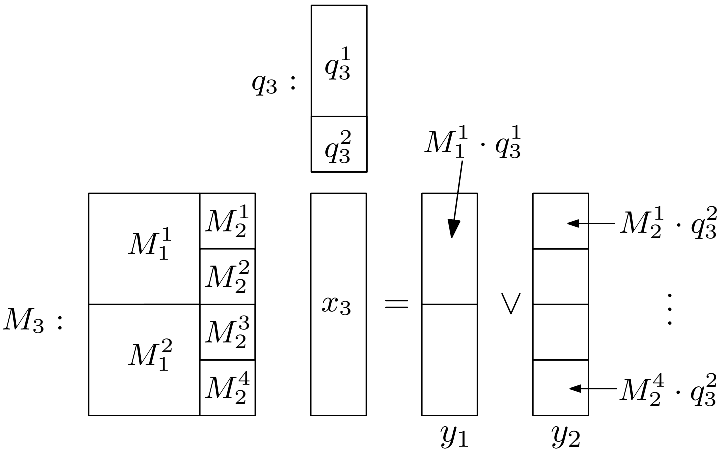


Figure 3.2: Example of calculating $x_3 = M_3 \otimes q_3$ based on the results from multiplication of square matrices M_i^z and vector q_3^i for $z \in [1, s/|q_3^i|]$ and $i \in \{1, 2\}$.

The correctness of the approach is immediate and now we need to calculate the total running time. While adding new columns to the considered matrix, we create a new interval of length 2^k for all j such that $j \equiv 2^k \pmod{2^{k+1}}$, so in total less than $s/2^k$ times. We divide every interval into $s/2^k$ matrices of size $2^k \times 2^k$ and for each of them we create an OMv data structure that answers at most 2^k queries. By Theorem 2.4.2 we can process online all the queries for a single matrix in $2^{3k}/2^{\Omega(\sqrt{k})}$ total time. This gives us the following total running time of processing all the queries:

$$\sum_{k=0}^{\log s} s/2^k \cdot s/2^k \cdot (2^k)^{3-\Omega(\sqrt{k})} = s^2 \cdot \sum_{k=0}^{\log s} 2^{k-\Omega(\sqrt{k})} = s^3/2^{\Omega(\sqrt{\log s})}$$

where the last step follows from Lemma 3.1.1. □

Finally, the total running time of our algorithm is $\tilde{\mathcal{O}}(s^\omega)$ for the preprocessing and $s^3/2^{\Omega(\sqrt{\log s})}$ for answering all the queries, which gives $s^3/2^{\Omega(\sqrt{\log s})}$ total time. This concludes the proof of Theorem 3.2.1.

Chapter 4

Equivalences between Non-trivial Variants of 3LDT and Conv3LDT

4.1 Preliminaries

In this work, we show a series of subquadratic reductions between different generalizations of 3SUM and Conv3SUM. A subquadratic reduction is formally defined as follows:

Definition 4.1.1 (cf. [WW18])**.** *Let A and B be computational problems with a common size measure m on inputs. We say that there is a* subquadratic reduction *from A to B if there is an algorithm $\mathcal{A}$ with oracle access to B, such that for every $\varepsilon > 0$ there is $\delta > 0$ satisfying three properties:*

1. *For every instance x of A, $\mathcal{A}(x)$ solves the problem A on x.*
2. *$\mathcal{A}$ runs in $\mathcal{O}(m^{2-\delta})$ time on instances of size m.*
3. *For every instance x of A of size m, let m_i be the size of the i-th oracle call to B in $\mathcal{A}(x)$. Then $\sum_i m_i^{2-\varepsilon} \leq m^{2-\delta}$.*

We use the notation $A \leq_2 B$ to denote the existence of a subquadratic reduction from A to B. If $A \leq_2 B$ and $B \leq_2 A$, we say that A and B are subquadratic-equivalent *and denote it $A \equiv_2 B$.*

Formal definitions of 3LDT, 3SUM, and Average. We work with the following formulations of 1- and 3-partite 3LDT, where $\bar{\alpha}$ denotes a triple $(\alpha_1, \alpha_2, \alpha_3)$:

1-partite $\mathsf{3LDT}_c(1, \bar{\alpha}, t)$
Parameters: Integer coefficients $\alpha_1, \alpha_2, \alpha_3$ and t, real $c \geq 2$.
Input: Number n, Set X of n numbers over the universe $[-n^c, n^c]$.
Output: Are there distinct $x_1, x_2, x_3 \in X$ such that $\sum_{i=1}^{3} \alpha_i x_i = t$?

3-partite $\mathsf{3LDT}_c(3, \bar{\alpha}, t)$

Parameters: Integer coefficients $\alpha_1, \alpha_2, \alpha_3$ and t, real $c \geq 2$.

Input: Number n, Sets S_1, S_2, S_3 of n numbers over the universe $[-n^c, n^c]$.

Output: Are there $x_1 \in S_1, x_2 \in S_2, x_3 \in S_3$ such that $\sum_{i=1}^{3} \alpha_i x_i = t$?

The $\mathsf{3SUM}_c$ problem is defined as $\mathsf{3LDT}_c(p, (1, 1, -1), 0)$, where $p = 1$ or $p = 3$ depending on the partity[1]. The $\mathsf{Average}_c$ problem introduced by Erickson [Eri95] is defined as $\mathsf{3LDT}_c(1, (1, 1, -2), 0)$.

Formal definitions of Conv3LDT, Conv3SUM, and ConvAverage. Let n be an odd number and $n' = \frac{n-1}{2}$. We consider the following formulation of Conv3LDT:

1-partite $\mathsf{Conv3LDT}\ _c(1, \bar{\alpha}, t)$

Parameters: Integer coefficients $\alpha_1, \alpha_2, \alpha_3$ and t, real $c \geq 1$.

Input: Number n, Array $A[-n', n']$ of n numbers over the universe $[-n^c, n^c]$.

Output: Are there distinct $j_1, j_2, j_3 \in [-n', n']$ such that $\sum_{i=1}^{3} \alpha_i j_i = t$ and $\sum_{i=1}^{3} \alpha_i A[j_i] = t$?

3-partite $\mathsf{Conv3LDT}\ _c(3, \bar{\alpha}, t)$

Parameters: Integer coefficients $\alpha_1, \alpha_2, \alpha_3$ and t, real $c \geq 1$.

Input: Number n, Arrays $A_1, A_2, A_3[-n', n']$ of n numbers over the universe $[-n^c, n^c]$.

Output: Are there $j_1, j_2, j_3 \in [-n', n']$ such that $\sum_{i=1}^{3} \alpha_i j_i = t$ and $\sum_{i=1}^{3} \alpha_i A_i[j_i] = t$?

Informally, in Conv3LDT we require the indices of elements to satisfy the same condition as their values. Analogously to above, the $\mathsf{Conv3SUM}_c$ problem is then defined as $\mathsf{Conv3LDT}_c(p, (1, 1, -1), 0)$, where $p = 1$ or $p = 3$ denotes whether the variant is 1-partite or 3-partite. The $\mathsf{ConvAverage}_c$ problem, first mentioned by Erickson[2], can be defined as $\mathsf{Conv3LDT}_c(1, (1, 1, -2), 0)$. We stress that the arrays consist of positive and negative indices. At the end of Section 4.4 we describe when and why this is necessary.

Finally, we require that $c \geq 2$ for 3LDT and $c \geq 1$ for Conv3LDT, as otherwise the instance can be solved in subquadratic time using the fast Fourier transform.

Our contribution. Consider an instance of 3LDT or Conv3LDT. Define the size of an instance to be the number n of elements in the considered sets or arrays.

Note that if any of the coefficients α_i is 0, then we need to find at most two numbers satisfying a linear relation, which can be done in $\mathcal{O}(n \log n)$ time for 3LDT (by first sorting and then for every candidate of x_3 scanning the sorted sequence with two pointers) and in $\mathcal{O}(n)$ time for Conv3LDT (by checking all pairs of indices satisfying the relation). Also, if $t \neq 0$ and $\gcd(\alpha_1, \alpha_2, \alpha_3) \nmid t$ then the instance is obviously a NO-instance, and we can return the answer in constant time. This motivates the following definition:

[1] While some works use this definition [Wil15; Cha18; BDP08], other [GO95; AC05; Eri99b; GS17; GP18] define the 3SUM problem as $\mathsf{3LDT}_c(p, (1, 1, 1), 0)$. It is well-known that the two variants are subquadratic-equivalent.

[2] See `https://cs.stackexchange.com/questions/10681/is-detecting-doubly-arithmetic-progressions-3sum-hard/10725#10725`).

Definition 4.1.2. *We call a variant of 3LDT$_c$ or Conv3LDT $_c$ problem with coefficients $\bar{\alpha}$ and t* trivial*, if either*

1. *Any of the coefficients α_i is zero, or*
2. *$t \neq 0$ and $\gcd(\alpha_1, \alpha_2, \alpha_3) \nmid t$*

and otherwise non-trivial*. If the above conditions hold, we call the coefficients $\bar{\alpha}$ and t* trivial*.*

We show that for all $c \geq 2$ all non-trivial variants of 3LDT$_c$ and Conv3LDT $_{c-1}$ are subquadratic-equivalent. Unless stated otherwise, we establish reductions between two variants of 3LDT (or Conv3LDT) with the same value of parameter c, that is over the same universe $U = [-n^c, n^c]$. To avoid clutter, in the reductions that work for all $c \geq 2$ we omit the parameter c. For example, one should read the notation 3LDT$(1, \overline{\alpha}, t) \leq_2$ 3LDT$(3, \overline{\alpha}, t)$ as: for all $c \geq 2$ it holds that 3LDT$_c(1, \overline{\alpha}, t) \leq_2$ 3LDT$_c(3, \overline{\alpha}, t)$. Similarly, for Conv3LDT $_c$ we read the statement Conv3LDT$(1, \bar{\alpha}, t) \leq_2$ Conv3LDT$(3, \bar{\alpha}, t)$ as: for all $c \geq 1$ it holds that Conv3LDT$_c(1, \bar{\alpha}, t) \leq_2$ Conv3LDT$_c(3, \bar{\alpha}, t)$. All numbers in the considered problems and reductions are integers. All reductions, unless said otherwise, are deterministic. We assume the standard w-bit word RAM model. That is, every word consists of w bits, and standard arithmetic and bitwise operations can be performed in constant time on such words. Internally, the words are unsigned integers from $[0, 2^w)$, however by using two's complement we can treat them as signed integers from $[-2^{w-1}, 2^{w-1})$. We assume that $w \geq \max\{\log n, \log U\}$.

We first establish equivalence for 3LDT:

Theorem 4.1.3. *For all $c \geq 2$, all non-trivial variants (1- and 3-partite) of 3LDT$_c$ are subquadratic-equivalent.*

In particular, this implies the following.

Corollary 4.1.4. *For all $c \geq 2$, Average$_c$ is subquadratic-equivalent to 3SUM$_c$.*

Thus, we completely resolve both Question 2.1.1 and Question 2.1.2. In order to design the most interesting of our reductions, from 3LDT$(3, \overline{\alpha}, 0)$ to 3LDT$(1, \overline{\alpha}, 0)$, we make use of progression-free sets. We call a set $S \subseteq \{1, 2, \ldots, n\}$ *progression-free* if it contains no non-trivial arithmetic progression, that is, three distinct elements a, b, c such that $a + b - 2c = 0$. Erdős and Turan [PP36] introduced the question of exhibiting a dense subset with such a property, and presented a construction with $\Omega(n^{\log_3 2})$ elements. This was improved by Salem and Spencer [SS42] to $n^{1-\mathcal{O}(1/\log\log n)}$, and then by Behrend [Beh46] to $\Omega(n/(2^{2\sqrt{2}\cdot\sqrt{\log n}} \cdot \log^{1/4} n))$. More recently, Elkin [Elk10] showed how to construct a set consisting of $\Omega(n \log^{1/4} n / 2^{2\sqrt{2}\cdot\sqrt{\log n}})$ elements. One could naturally ask for a dense subset that avoids a certain linear equation $\alpha_1 x_1 + \alpha_2 x_2 = (\alpha_1 + \alpha_2) x_3$, where α_1, α_2 are positive integers. Indeed, it turns out that Behrend's argument works with minor modifications also for such equations [Ruz93, Theorem 2.3]. We use an extension of this argument to partition an arbitrary set into a small number of progression-free sets. Average-free sets have been already successfully applied in various areas of theoretical computer science [CFL83; CW90;

HW03; WW13; DvM14; AFKS00; AB17; FGLS14; JKMS13]. In particular, Behrend-like constructions led to conditional lower bounds providing e.g. a reduction from k-Clique to k^2-SUM [ALW14], from k-SAT to Subset Sum [ABHS19] and for scheduling problems [ABHS22].

We then extend our techniques to show equivalences for Conv3LDT:

Theorem 4.1.5. *For all $c \geq 1$, all non-trivial variants (1- and 3-partite) of Conv3LDT $_c$ are subquadratic-equivalent.*

One can suspect that 3LDT is connected to Conv3LDT on a smaller universe as the indices of elements can also convey some information. We show that this intuition is in fact fully correct. By adjusting a folklore reduction from Conv3LDT to 3LDT and a modification of Pătraşcu's [Păt10] and Chan and He's [CH20] reduction from 3SUM to Conv3SUM we obtain that these problems are equivalent when the size of the universe of considered instances differs by a factor of n between convolution and non-convolution variants. Finally, by combining this with Theorems 4.1.3 and 4.1.5 we obtain equivalences between all non-trivial 1- and 3-partite variants of Conv3LDT and 3LDT, both convolution and non-convolution:

Theorem 4.1.6. *For every $c \geq 2$, all non-trivial 1- and 3-partite variants of 3LDT$_c$ and of Conv3LDT $_{c-1}$ are subquadratic-equivalent.*

In the above equivalences we only consider polynomial-size universes. Of course one can also consider problems with two parameters: number of elements n and the size of the universe U, possibly much bigger than n. However, it turns out that instances of LDT over a universe bigger than cubic can be deterministically reduced to instances over the cubic universe. This follows by extending the result of Fischer et al. [FKP24] as explained in detail in Section 4.5. Consequently, the only interesting variants of LDT are 3SUM$_c$ for $c \in [2, 3]$. Instances over larger universes for any non-trivial variant are equivalent to LDT$_3$, while instances over smaller universes or of any trivial variant can be solved in subquadratic time.

Theorem 4.1.7. *For every $U \geq n^3, p \in \{1, 3\}$ and non-trivial coefficients $\bar{\alpha}, t$, LDT$_?(p, \bar{\alpha}, t)$ over $[-U, U]$ is subquadratic-equivalent to LDT$_3(p, \bar{\alpha}, t)$.*

By combining this result with reductions between LDT and Conv3LDT, we can draw a similar conclusion for Conv3LDT, that the only interesting non-trivial variants are Conv3SUM$_c$ for $c \in [1, 2]$.

Theorem 4.1.8. *For every $U \geq n^3, p \in \{1, 3\}$ and non-trivial coefficients $\bar{\alpha}, t$, Conv3LDT$_?(p, \bar{\alpha}, t)$ over $[-U, U]$ is subquadratic-equivalent to Conv3LDT$_2(p, \bar{\alpha}, t)$.*

We stress that all reductions provided in this paper are deterministic.

4.2 Equivalences between different variants of 3LDT

In this section, we show Theorem 4.1.3 through a series of reductions depicted in Figure 4.1. In order to show a subquadratic reduction $A \leq_2 B$ we often present only a reduction from an instance of A to a number of instances of B. If the analysis of the sizes of the obtained instances (whether they satisfy Property 3 of Definition 4.1.1 or not) is immediate, we omit it. Here and below we use the following notation: when writing $\sum_i$ we mean the sum over all possible values of i; $[k] = \{1, 2, \dots, k\}$; $f[A] = \{f(a) : a \in A\}$ is the image of f over A; sumset $A + B$ is defined as $\{a + b : a \in A, b \in B\}$ and in particular we can add an element to a set: $A + x = A + \{x\} = \{a + x : a \in A\}$.

$$\begin{tikzcd}
\mathsf{3LDT}(3, \overline{\alpha_1}, t_1) \arrow[d, leftrightarrow, "\text{Lem. 4.2.3}"'] & \mathsf{3LDT}(1, \overline{\alpha_1}, t_1) \arrow[l, "\text{Lem. 4.2.4}"'] \\
\mathsf{3LDT}(3, \overline{\alpha_2}, t_2) \arrow[r, "\text{Lem. 4.2.8 \& 4.2.12}"'] & \mathsf{3LDT}(1, \overline{\alpha_2}, t_2)
\end{tikzcd}$$

Figure 4.1: Subquadratic reductions between different variants of 3LDT. Subscripts to coefficients α and t denote if the reduction allows changing the coefficients or not. In all the reductions the value of parameter c is preserved and hence not shown.

We start by showing technical lemmas that allow us to handle instances of 3LDT in which sets have at most n elements and their elements can exceed the $[-n^c, n^c]$ range, but are within the range $[-dn^c, dn^c]$ for some constant $d \geq 1$. We call such instances *semi-instances*. In such a case we add a number of elements that will never be a part of triple $\bar{x}$ of elements satisfying $\sum_i \alpha_i x_i = t$.

Proposition 4.2.1. *For any triple $\bar{\alpha}$ of nonzero coefficients we can find in constant time integer coefficients $\bar{\beta}$ such that $\forall_{\emptyset \neq S \subseteq [3]} \sum_{i \in S} \alpha_i \beta_i \neq 0$.*

Proof. We need to find coefficients $\bar{\beta}$ satisfying all seven conditions of the form $\sum_{i \in S} \alpha_i \beta_i \neq 0$ for $\emptyset \neq S \subseteq [3]$. First we set $\beta_1 = 1$. As $\alpha_i \neq 0$, we have $\alpha_1 \beta_1 \neq 0$. Next, by setting $\beta_2 = \max\{0, \lceil \frac{\alpha_1 \beta_1}{-\alpha_2} \rceil\} + 1$ we guarantee that $\alpha_2 \beta_2 \neq 0$ and $\alpha_1 \beta_1 + \alpha_2 \beta_2 \neq 0$. Finally, we set $\beta_3 = \max\{0, \lceil \frac{\alpha_1 \beta_1}{-\alpha_3} \rceil, \lceil \frac{\alpha_2 \beta_2}{-\alpha_3} \rceil, \lceil \frac{\alpha_1 \beta_1 + \alpha_2 \beta_2}{-\alpha_3} \rceil\} + 1$ and fulfill all the conditions with $3 \in S$. Hence the above choice of $\bar{\beta}$ fulfills the required seven conditions. □

Lemma 4.2.2. *Let $\bar{\alpha}$ be non-zero coefficients, $c \geq 2, d \geq 1$ and t be arbitrary. Given a semi-instance of $\mathsf{3LDT}_c(3, \overline{\alpha}, t)$ with sets S_1, S_2, S_3 of at most n elements from $[-n^c, n^c]$, we can construct in $\mathcal{O}(n)$ time an equivalent instance of $\mathsf{3LDT}_c(3, \overline{\alpha}, t)$, with sets S_1', S_2', S_3' of $n' = \mathcal{O}(n)$ elements from $[-\frac{1}{d}(n')^c, \frac{1}{d}(n')^c]$ such that there exists a triple $\bar{x} \in S_1 \times S_2 \times S_3$ of elements such that $\sum_i \alpha_i x_i = t$ iff there exists a triple $\bar{x}' \in S_1' \times S_2' \times S_3'$ such that $\sum_i \alpha_i x_i' = t$.*

Proof. By Proposition 4.2.1 we can find $\bar{\beta}$ such that $\forall_{\emptyset \neq S \subseteq [3]} \sum_{i \in S} \alpha_i \beta_i \neq 0$ and let $\beta^* = \max_i |\beta_i|$. Let $M = 2n^c \cdot \sum |\alpha_i|$ and n' satisfy $(n')^c \geq 2\beta^* M d$, so $n' = \lceil n(4d\beta^* \sum_i |\alpha_i|)^{1/c} \rceil$. Clearly $n' \geq n$ and for sufficiently big n we have $n' \leq n^c$. We set $S_i' = S_i \cup \{\beta_i M + j : j \in \{1, 2, \dots, n' - |S_i|\}\}$. Now we show that the sets S_i' satisfy the desired properties. Clearly, the above construction runs in $\mathcal{O}(n)$ time and sets S_i' have $n' = \mathcal{O}(n)$ elements each. Next, $S_i' \subseteq [-\frac{1}{d}(n')^c, \frac{1}{d}(n')^c]$ because $\frac{1}{d}(n')^c \geq 2\beta^* M \geq n^c$, so $S_i \subseteq [-n^c, n^c] \subseteq [-\frac{1}{d}(n')^c, \frac{1}{d}(n')^c]$ and

$S'_i \setminus S_i \subseteq [\beta_i M + 1, \beta_i M'_n] \subseteq [-2\beta^* M, 2\beta^* M]$. If there is a triple $\bar{x}$ such that $x_i \in S_i$ and $\sum \alpha_i x_i = t$ then it is also a solution for S'_1, S'_2, S'_3.

Now we show the opposite direction. Suppose there is a triple $\bar{x}'$ such that $x'_i \in S'_i$ and $\sum \alpha_i x'_i = t$. Define $\bar{\psi}$ as $\psi_i = \beta_i$ if $x'_i \in S'_i \setminus S_i$ and $\psi_i = 0$ if $x'_i \in S_i$. Then $|\sum \alpha_i x'_i - \sum \alpha_i \psi_i M| \leq \sum |\alpha_i| n^c = M/2$. By Proposition 4.2.1 $\sum \alpha_i \psi_i M = 0$ iff $\psi_1 = \psi_2 = \psi_3 = 0$. Otherwise $|\sum \alpha_i \psi_i M| \geq M$ so $|\sum \alpha_i x_i| \geq M/2$ which contradicts the assumption that $\sum \alpha_i x_i = t$. Hence for every triple $\bar{x}'$ such that $x'_i \in S'_i$ and $\sum \alpha_i x_i = t$ we have that $x'_i \in S_i$ which concludes the proof. □

Now we show equivalence of all non-trivial 3-partite variants of 3LDT and a reduction from 1-partite variant to 3-partite variant. To avoid clutter, till the end of this section we do not write the parameter c in the description of the considered variants as c is preserved in all the presented reductions and satisfies $c \geq 2$.

Lemma 4.2.3. *All non-trivial 3-partite variants of 3LDT are subquadratic-equivalent.*

Proof. We need to show a reduction between any two non-trivial 3-partite variants of 3LDT. To this end, we establish three reductions: 3LDT$(3, \overline{\alpha}, 0) \leq_2$ 3LDT$(3, \overline{\alpha}, t)$ and 3LDT$(3, \overline{\alpha}, t) \leq_2$ 3LDT$(3, \overline{\alpha}, 0)$ and finally 3LDT$(3, \overline{\alpha}, 0) \leq_2$ 3LDT$(3, \overline{\beta}, 0)$ for $\bar{\alpha}, \bar{\beta}$ and $t \neq 0$ such that all the considered variants are non-trivial. In each of those reductions we need to make sure that we obtain an instance of 3LDT and not a semi-instance. Each of them requires the same additional clean-up stage that we present at the end of this proof. Reductions between other variants can be obtained by combining at most three of the above.

1. 3LDT$(3, \overline{\alpha}, 0) \leq_2$ 3LDT$(3, \overline{\alpha}, t)$. We have $\gcd(\alpha_1, \alpha_2, \alpha_3) | t$ because $\bar{\alpha}$ and t are non-trivial coefficients, so by the Chinese remainder theorem there exists an integer triple $\bar{y}$ such that $\sum_i \alpha_i y_i = t$. Given the three sets S_1, S_2, S_3 we construct three sets S'_1, S'_2, S'_3 where $S'_i = \{x + y_i : x \in S_i\}$. Then there is $\bar{x} \in S_1 \times S_2 \times S_3$ satisfying $\sum_i \alpha_i x_i = 0$ iff there is $\bar{x}' \in S'_1 \times S'_2 \times S'_3$ satisfying $\sum_i \alpha_i x'_i = t$.

2. 3LDT$(3, \overline{\alpha}, t) \leq_2$ 3LDT$(3, \overline{\alpha}, 0)$. As above but by subtracting the y_i terms.

3. 3LDT$(3, \overline{\alpha}, 0) \leq_2$ 3LDT$(3, \overline{\beta}, 0)$. Define $q = \operatorname{lcm}(\beta_1, \beta_2, \beta_3)$ so that $\frac{\alpha_i q}{\beta_i}$ is an integer. Given S_1, S_2 and S_3 we construct S'_1, S'_2 and S'_3 by setting $S'_i = \{x \frac{\alpha_i q}{\beta_i} : x \in S_i\}$.

Recall that all the considered variants of 3LDT are over the same universe $[-n^c, n^c]$ for some $c \geq 2$. Now we need to handle the situation that sets S'_1, S'_2, S'_3 form a semi-instance of 3LDT, because after the above transformation the elements can be outside the range $[-n^c, n^c]$. In the first two cases, the universe increases at most by an additive factor $\max |y_i| = \mathcal{O}(1)$, so not more than by multiplicative factor 2. In the third case the universe increases at most by factor $q \cdot \max |\frac{\alpha_i}{\beta_i}|$. Let $w = \max\{2, q \cdot \max_i |\frac{\alpha_i}{\beta_i}|\}$. Before applying the reduction, we first apply Lemma 4.2.2 with $d = w$ obtaining sets of $n' = \Theta(n)$ elements from $[-\frac{1}{d}(n')^c, \frac{1}{d}(n')^c]$. Then after multiplying or adding a constant to an element, it still belongs to $[(n')^c, (n')^c]$. □

To show $\mathsf{3LDT}(1, \overline{\alpha}, t) \leq_2 \mathsf{3LDT}(3, \overline{\alpha}, t)$, we apply the folklore reduction from 1-partite 3-SUM to 3-partite 3-SUM based on the color-coding technique of Alon et al. [AYZ95]. For completeness we present the proof below.

Lemma 4.2.4. *For all $\bar{\alpha}$ and t, 3LDT$(1, \overline{\alpha}, t) \leq_2$ 3LDT$(3, \overline{\alpha}, t)$.*

Proof. In this reduction, given one set X we need to create a number of 3-partite instances of $\mathsf{3LDT}$ in such a way that there exist distinct $x_1, x_2, x_3 \in X$ satisfying the given equation iff at least one of the 3-partite instances is a YES-instance. The reduction will not change coefficients α_i and the parameter t. Note that simply creating a single 3-partite instance by making all three sets equal to X does not work, as we are not able to forbid taking the same element of X more than once.

We use the color-coding technique that was introduced by Alon et al. [AYZ95], in which we choose a number of colorings of the elements of X with k colors in such a way that, for every k-element subset of X, there is a coloring in which all elements from the subset have distinct colors. This can be achieved with high probability by simply choosing sufficiently many random colorings, but we will use the deterministic construction by Schmidt and Siegel [SS90].

Fact 4.2.5 (cf. [SS90])**.** *There exists a family F of $2^{\mathcal{O}(k)} \log^2 n$ functions $[n] \to [k]$ such that, for every k-element set $Y \subseteq [n]$, there exists a function $f \in F$ with $|f[Y]| = k$. Each function is described by a bit string of length $\mathcal{O}(k) + 2 \log \log n$ and, given constant-time read-only random access to the bit string describing $f \in F$ and any $x \in [n]$, we can compute $f(x)$ in constant time.*

We work with $k = 3$, so the above fact gives us a family F consisting of $\mathcal{O}(\log^2 n)$ functions. Given a set $X = \{x_1, x_2, \ldots, x_n\}$, for every function $f \in F$ and every permutation $\pi \in S_3$ we obtain a 3-partite semi-instance of $\mathsf{3LDT}$ by setting $S_{\pi(i)} = \{x_c : f(c) = i\}$ for $i \in [3]$. In every 3-partite semi-instance the sets S_i correspond to a partition of the original set X, and for any distinct $x_1, x_2, x_3 \in X$ there exists a 3-partite semi-instance such that $x_1 \in S_1$, $x_2 \in S_2$, and $x_3 \in S_3$. Thus, we showed how to reduce a 1-partite instance of $\mathsf{3LDT}$ to $\mathcal{O}(\log^2 n)$ semi-instances of 3-partite $\mathsf{3LDT}$ with the same coefficients $\bar{\alpha}$ and t. The reduction works in $\mathcal{O}(n \log^2 n)$ time. Finally, we reduce every semi-instance to an instance of 3-partite $\mathsf{3LDT}$ by Lemma 4.2.2 with $d = 1$. □

It remains to show how to reduce an arbitrary non-trivial 3-partite variant of $\mathsf{3LDT}$ to a 1-partite one with the same coefficients $\bar{\alpha}$ and t. Before proceeding to the reduction, we show a few preliminary lemmas. Let C be a sufficiently big constant to be fixed later. Recall that in definition of $\mathsf{3LDT}(3, \overline{\alpha}, t)$ we have three sets S_i, and in the definition of $\mathsf{3LDT}(1, \overline{\alpha}, t)$ one set X. We would like to construct the set X by setting $X = \bigcup_i \{Cx + \gamma_i : x \in S_i\}$, where $\bar{\gamma}$ are pairwise distinct coefficients chosen so as to ensure that all triples $\bar{x}$ consisting of distinct elements from X satisfying $\sum_i \alpha_i x_i = t$ also satisfy that x_i corresponds to an element of S_i, for every $i \in [3]$. For example, for $\mathsf{3SUM}$ we can set $X = \{3C + x : x \in S_1\} \cup \{C + x : x \in S_2\} \cup \{4C + x : x \in S_3\}$. Now we extend this approach to arbitrary coefficients.

For a triple $\bar{x}$ of elements from X, an *origin* is a function $f : [3] \to [3]$ such that, for every $i \in [3]$, x_i corresponds to an element of $S_{f(i)}$. Clearly f is a function, because γ's are pairwise distinct. For instance, consider $\bar{x} = (Cx + \gamma_2, Cy + \gamma_3, Cz + \gamma_3)$ where $x \in S_2$ and $y, z \in S_3$. Then we have $f(1) = 2$ and $f(2) = f(3) = 3$. Intuitively, now we would like to find coefficients $\bar{\gamma}$ such that for every triple $\bar{x}$ of elements from X such that $\sum_i \alpha_i x_i = t$ their origin is the identity function ($f(i) = i$ for $i \in [3]$), so in particular we need to forbid using more than one number from the same set ($|\{f(i) : i \in [3]\}| < 3$). However, some coefficients from $\bar{\alpha}$ might be equal, so we also need to allow origins in which we permute the elements with the same values of α_i. For example, if all coefficients α are equal, we allow every origin that is a permutation of [3], and when $\alpha_1 = \alpha_3 \neq \alpha_2$, we have two allowed origins: $(1, 2, 3)$ and $(3, 2, 1)$. This is formalized in the following definition.

Definition 4.2.6. *For any coefficients $\bar{\alpha}$, we call an origin f* allowed *if $\forall i \in [3] \{f(x) : x \in [3], \alpha_x = \alpha_i\} = \{x : x \in [3], \alpha_x = \alpha_i\}$, and otherwise we call it* forbidden. *In addition, if $f(1) = f(2) = f(3)$ we call the origin* constant.

We show that it is always possible to find a triple $\bar{\gamma}$ which excludes solutions from most of the forbidden origins. In other words, we present how to find $\bar{\gamma}$ such that for every triple $\bar{x}$ of elements from the constructed set X such that $\sum_i \alpha_i x_i = t$, we have that origin of $\bar{x}$ is either allowed or constant. Additionally, we need to ensure that in the allowed origin the summands not multiplied by C cancel out, so we require that $\sum_i \alpha_i \gamma_i = 0$.

Lemma 4.2.7. *For any triple $\bar{\alpha}$ of nonzero coefficients there exists a triple $\bar{\gamma}$ of nonzero, pairwise distinct coefficients such that $\sum_i \alpha_i \gamma_i = 0$ and for every non-constant forbidden origin f we have $\sum_i \alpha_i \gamma_{f(i)} \neq 0$.*

Proof. Consider the 3-dimensional space $\mathbb{Q}^3$. Clearly, the set of all triples $\bar{\gamma}$ such that $\sum_i \alpha_i \gamma_i = 0$ spans a plane there, we denote it Γ_{id}. There are less than $3^3 = \mathcal{O}(1)$ non-constant forbidden origins f and each of them corresponds to an equation $\sum_i \alpha_i \gamma_{f(i)} = 0$ that must be avoided, which also corresponds to a forbidden plane Γ_f. By the definition of a forbidden origin f, we have $\Gamma_f \neq \Gamma_{id}$. Moreover, even if $\sum_i \alpha_i = 0$, Γ_f is not the whole space $\mathbb{Q}^3$, as f is a non-constant configuration. Next, as we need all the coefficients γ_i to be nonzero, we add forbidden planes $\Gamma_i = \{\bar{\gamma} : \gamma_i = 0\}$, for $i \in [3]$. Similarly, as we need all the coefficients γ_i to be pairwise distinct, we add forbidden planes $\Gamma'_i = \{\bar{\gamma} : \gamma_i = \gamma_{(i+1) \bmod 3+1}\}$, for $i \in [3]$. Clearly, $\Gamma_i \neq \Gamma_{id}$ and $\Gamma'_i \neq \Gamma_{id}$ because the coefficients $\bar{\alpha}$ are nonzero. Then let $\mathcal{F} = \{\Gamma_f : f \text{ is non-constant and forbidden}\} \cup \{\Gamma_i : i \in [3]\} \cup \{\Gamma'_i : i \in [3]\}$ be the set of all forbidden planes. Now we need to show that $\Gamma_{id} \setminus \bigcup_{F \in \mathcal{F}} F \neq \emptyset$, using the assumption that $\forall_{F \in \mathcal{F}} F \neq \Gamma_{id}$.

Clearly both Γ_{id} and all planes $f \in \mathcal{F}$ contain the origin $o = (0, 0, 0)$. Consider an arbitrary line $\ell \subset \Gamma_{id}$ that does not pass through the origin o and contains infinitely many points with all coordinates rational. For example, we can take the line passing through $(1, 0, -\alpha_1/\alpha_3)$ and $(0, 1, -\alpha_2/\alpha_3)$. Observe that for any $F \in \mathcal{F}$, if $|\ell \cap F| \geq 2$ there would be three non-collinear points (two from ℓ and o) belonging to two distinct planes Γ_{id} and F, so contradiction. Hence $\ell \cap F$ is either empty or a point. Recall that there is a constant number of planes in $\mathcal{F}$. Then

$\Gamma_{id} \setminus \bigcup_{F\in\mathcal{F}} F \supseteq \ell \setminus \bigcup_{F\in\mathcal{F}} F$ contains some point with rational coordinates, because there are infinitely many such points on ℓ. This gives us a point in $\mathbb{Q}^3$ that belongs to Γ_{id} and does not belong to any $F \in \mathcal{F}$. By scaling its coordinates to integers, we obtain $\bar{\gamma}$. □

Now we are ready to show the reduction from $\mathsf{3LDT}(3, \overline{\alpha}, t)$ to $\mathsf{3LDT}(1, \overline{\alpha}, t)$ for $t \neq 0$.

Lemma 4.2.8. *Assume t and $\bar{\alpha}$ are non-zero. For any $\bar{\alpha}$, every 3-partite instance of 3LDT$(3, \overline{\alpha}, t)$ reduces in linear time to a 1-partite instance of 3LDT$(1, \overline{\alpha}, t)$.*

Proof. If the considered variant of 3LDT is trivial, we can solve it in $\mathcal{O}(n)$ time and terminate. Otherwise, by the extended Euclidean algorithm, we can choose an integer triple $\bar{y}$ such that $\sum_i \alpha_i y_i = t$ and let $\mathcal{Y}(\bar{\alpha}, t) = \max_i |y_i|$. We apply Lemma 4.2.7 on $\bar{\alpha}$ to obtain $\bar{\gamma}$ and construct the set X as follows:

$$X = \bigcup_i \{C^2(x - y_i) + C\gamma_i + y_i : x \in S_i\},$$

where C is a sufficiently big constant such that the absolute value of any linear combination of γ's or y's with coefficients α_i is smaller than C (for example, we can take $C = 1 + (\max_i \max\{|\gamma_i|, |y_i|\}) \cdot \sum_i |\alpha_i|$). If there is a triple $\bar{x}$ such that $x_i \in S_i$ and $\sum_i \alpha_i x_i = t$, then by the choice of $\bar{\gamma}$ and $\bar{y}$ we have $\sum_i \alpha_i z_i = t$, where $z_i = C^2(x_i - y_i) + C\gamma_i + y_i$. As coefficients γ_i are pairwise distinct, elements z_i are pairwise distinct as well. Recall that, for a given element $z \in X$, the function f returns the index i such that z comes from an element of S_i. Now consider a triple $\bar{z}$ such that $z_i \in X$ and $\sum \alpha_i z_i = t$. Let $z_i = C^2(x_{f(i)} - y_{f(i)}) + C\gamma_{f(i)} + y_{f(i)}$, where $x_{f(i)} \in S_{f(i)}$. By the definition of C and the fact that $\sum_i \alpha_i z_i = t$, it holds that $\sum \alpha_i(x_{f(i)} - y_{f(i)}) = 0$, $\sum \alpha_i \gamma_{f(i)} = 0$ and $\sum \alpha_i y_{f(i)} = t$. We will show that f is an allowed origin which guarantees that $x_{f(1)}, x_{f(2)}, x_{f(3)}$ is a valid solution of $\mathsf{3LDT}(3, \overline{\alpha}, t)$.

By Lemma 4.2.7, $\sum_i \alpha_i \gamma_{f(i)} = 0$ implies that the origin f is either constant or allowed. If f is constant, there exists $j \in [3]$ such that $f(i) = j$ for all $i \in [3]$, so from the fact that $\sum_i \alpha_i \gamma_{f(i)} = 0$ we have $\sum_i \alpha_i \gamma_j = 0$ and therefore $\sum_i \alpha_i = 0$ as $\gamma_j \neq 0$. It implies $\sum_i \alpha_i y_{f(i)} = \sum_i \alpha_i y_j = 0 \neq t$, hence f cannot be constant and is allowed.

Recall that the considered numbers from sets S_i are from the universe $[-U, U]$ where $U = n^c$ and c is a parameter of the considered variant. For n large enough, $U \geq \mathcal{Y}(\bar{\alpha}, t) = \max_i |y_i|$. Consequently, $|x - y_i| \leq 2U$, so the absolute value of the elements of X is at most $2C^2U + C^2 + C \leq 3C^2U$, so we obtain a semi-instance of $\mathsf{3LDT}(1, \overline{\alpha}, t)$. To avoid that, we first apply Lemma 4.2.2 on sets S_1, S_2, S_3 with $d = 3C^2$ and then create set X from the obtained sets S'_1, S'_2, S'_3, which concludes the reduction to an instance of $\mathsf{3LDT}(1, \overline{\alpha}, t)$. □

Surprisingly, the case $t = 0$ is more difficult. We would like to proceed as in Lemma 4.2.8, which is enough to exclude all non-constant forbidden origins and, if $\sum_i \alpha_i \neq 0$, also the constant origins. However, if $\sum_i \alpha_i = 0$, then we cannot exclude the constant origins. In other words, no matter what the chosen γ's are we are not able to exclude the solutions that use three distinct elements corresponding to the elements of the same set S_j. This suggests that we should partition each of the sets S_j into a few sets that contain no triple $\bar{x}$ of distinct elements such that $\sum_i \alpha_i x_i = 0$. To this end, we introduce the following definition:

Definition 4.2.9. *For any $\gamma, \delta > 0$, a set X is* (γ, δ)-free *if no three distinct elements $a, b, c \in X$ satisfy $\gamma a + \delta b = (\gamma + \delta)c$.*

Now we show that we can always partition an arbitrary subset of $[N]$ into $2^{\mathcal{O}(\sqrt{\log N})}$ (γ, δ)-free sets[3].

Theorem 4.2.10 (cf. [Beh46; Ruz93])**.** *For any $\gamma, \delta > 0$ and a set $X \subseteq [N]$, it is possible to construct $e = 2^{\mathcal{O}(\sqrt{\log N})}$ sets $X_1, X_2, \ldots, X_e$ such that every X_j is (γ, δ)-free and $\bigcup_j X_j = X$. The construction is deterministic and runs in $\mathcal{O}(|X|)$ time.*

Proof. Let $p = 2(\gamma + \delta) + 1, q = 2^{\sqrt{\log N}}$ and $r = \lceil \frac{q}{p} \rceil$. We represent every element $x \in X$ in base q: $x = \sum_{i=0}^{d-1} x_i q^i$ where $d = \lceil \log_q N \rceil = \lceil \sqrt{\log N} \rceil$ and $x_i \in \{0, \ldots, q-1\}$. For every i, define $\tilde{x}_i = \lfloor x_i / r \rfloor \in \{0, \ldots p-1\}$ and $x'_i = x_i \bmod r \in \{0, \ldots, r-1\}$. We put an element x to a set X_τ, where the index τ is a tuple $(\tilde{x}_0, \ldots, \tilde{x}_{d-1}; ||x'||_2^2)$. As $||x||_2^2 < dr^2$, there will be at most $e = p^d dr^2 \le p^d dq^2 = 2^{\mathcal{O}(\sqrt{\log N})}$ distinct sets.

We claim that the sets X_τ are (γ, δ)-free. Suppose the contrary, that there are three distinct elements $a, b, c \in X_\tau$ such that $\gamma a + \delta b = (\gamma + \delta)c$. By combining that with representations of a, b, c in base q we obtain:

$$\gamma \cdot \sum_i q^i (a'_i + r \cdot \tilde{a}_i) + \delta \cdot \sum_i q^i (b'_i + r \cdot \tilde{b}_i) = (\gamma + \delta) \cdot \sum_i q^i (c'_i + r \cdot \tilde{c}_i)$$

By the definition of X_τ we have $\tilde{a}_i = \tilde{b}_i = \tilde{c}_i$ for $0 \le i < d$ so the above equation simplifies to

$$\sum_i q^i (\gamma \cdot a'_i + \delta \cdot b'_i - (\gamma + \delta) \cdot c'_i) = 0$$

Recall that $x'_i \le r - 1 < \frac{q}{2(\gamma+\delta)+1}$, so there is no carry between different base-q digits of left-hand side of the expression, and for every $0 \le i < d$ we have $\gamma a'_i + \delta b'_i = (\gamma + \delta) c'_i$. Considering vectors a', b', c', this gives us $\gamma a' + \delta b' = (\gamma + \delta)c'$. We combine it with the triangle inequality and the fact that $||a'||_2 = ||b'||_2 = ||c'||_2$ from the definition of X_τ:

$$(\gamma + \delta)||c'||_2 = \|(\gamma + \delta)c'\|_2 = \|\gamma a' + \delta b'\|_2 \le \|\gamma a'\|_2 + \|\delta b'\|_2 = (\gamma + \delta)||c'||_2$$

which becomes an equality if and only if a' and b' are collinear. Their norms are equal because $a, b \in X_\tau$, so we finally obtain $a' = b' = c'$ which contradicts the distinctness of a, b and c. □

Corollary 4.2.11. *For any $\gamma, \delta, s > 0$, in $s \cdot 2^{\mathcal{O}(\sqrt{\log s})}$-time we can construct a (γ, δ)-free set of s elements from $[s \cdot 2^{\mathcal{O}(\sqrt{\log s})}]$.*

[3] The idea of this proof is borrowed from [JX23], who suggested how to improve the earlier version of the proof that appeared in the conference version of this work [DGS20]. Before we used probabilistic argument to obtain the partition from an arbitrary construction of dense (γ, δ)-free sets (possibly more dense than the Behrend's one) that provides only oracle membership access, but in fact this is not necessary in our reduction from 3-partite to 1-partite instances of LDT.

Proof. Recall that $[n] = \{1, \ldots, n\}$. Let d satisfy that the value $e = 2^{\mathcal{O}(\sqrt{\log N})}$ from Theorem 4.2.10 is bounded by $e \leq 2^{d\sqrt{\log N}}$ and let $X = [s \cdot 2^{d\sqrt{2\log s}}]$. By Theorem 4.2.10 for $N = s^2$, we can partition X into at most $2^{d\sqrt{2\log s}}$ sets, so at least one of them contains at least s elements. □

Lemma 4.2.12. *For every $\bar{\alpha}$ it holds 3LDT$(3, \overline{\alpha}, 0) \leq_2$ 3LDT$(1, \overline{\alpha}, 0)$.*

Proof. We show that every 3-partite instance of 3LDT$(3, \overline{\alpha}, 0)$ can be reduced to a number of instances of 1-partite 3LDT$(1, \overline{\alpha}, 0)$. If the considered variant of 3LDT is trivial, we can solve it in $\mathcal{O}(n)$ time and terminate. If $\sum_i \alpha_i \neq 0$, we apply Lemma 4.2.7 on $\bar{\alpha}$ to obtain $\bar{\gamma}$ and construct the set $X = \bigcup_i \{Cx + \gamma_i : x \in S_i\}$ where C is a sufficiently big constant chosen as in Lemma 4.2.8. All the coefficients γ_i are pairwise distinct, so there is no element added to X more than once. Similarly as in Lemma 4.2.8, this definition of set X is enough to exclude all non-constant forbidden origins and, if $\sum_i \alpha_i \neq 0$, also the constant origins.

Consider now the case $\sum_i \alpha_i = 0$. Without loss of generality, there is a permutation i_1, i_2, i_3 of $[3]$ such that $\alpha_{i_1}, \alpha_{i_2}$ are positive and α_{i_3} is negative. (If this is not the case, we first multiply all the coefficients by -1, apply our reduction to 1-partite variant and in the end we again multiply the coefficients of the obtained instances by -1.) We apply Theorem 4.2.10 to partition every set S_i into $e = 2^{\mathcal{O}(\sqrt{\log N})}$ $(\alpha_{i_1}, \alpha_{i_2})$-free subsets $S_{i,j}$, where $i \in [3], j \in [e]$. (We shift S_i by U to apply the theorem, and then shift the resulting sets $S_{i,j}$ by $-U$.) We reduce the instance of 3LDT$(3, \overline{\alpha}, 0)$ to e^3 semi-instances of 3LDT$(1, \overline{\alpha}, 0)$ by considering all possible combinations of the subsets and applying the construction for the case $\sum_i \alpha_i \neq 0$, which excludes all non-constant forbidden origins by the choice of $\bar{\gamma}$ and all constant origins by the partition. To show that the reduction is subquadratic, we must analyze the sizes of the obtained instances. As $2^{\mathcal{O}(\sqrt{\log n})} < n^{\varepsilon'}$ for any $\varepsilon' > 0$, we have $2^{\mathcal{O}(\sqrt{\log n})} \cdot n^{2-\varepsilon} < n^{2-\delta}$ for all $0 < \delta < \varepsilon$.

Finally, we note that we obtain semi-instances of 3LDT$(1, \overline{\alpha}, 0)$, because after the partition we have less than n elements and the constructed set $X \subseteq [-2Cn^c, 2Cn^c]$. Similarly as in Lemma 4.2.8, before constructing X we apply Lemma 4.2.2 with $d = 2C$, which gives us instances of 3LDT$(1, \overline{\alpha}, 0)$. □

4.3 Equivalences between different variants of Conv3LDT

In this section, we show Theorem 4.1.5 that claims that for all $c \geq 1$ all non-trivial variants of Conv3LDT$_c$ are subquadratic-equivalent. A scheme of reductions is shown in Figure 4.2. To avoid clutter, till the end of this section we do not write the parameter c in the description of the considered variants as c is preserved in all the presented reductions and satisfies $c \geq 1$.

The arguments are similar to those that we used in Section 4.2, but we provide them all for completeness. A subtle difference is that for arrays we cannot take a subset of elements (which was possible for sets) and we need to replace the elements to be removed with some dummy values that can never be part of a solution. Similarly as in Lemma 4.2.2, we show that we can remove a subset of elements from the arrays by replacing them with some dummy values that

$$\begin{tikzcd}\mathsf{Conv3LDT}(3, \overline{\alpha_1}, t_1) \arrow[r, leftrightarrow, "\text{Lem. 4.3.2}"'] & \mathsf{Conv3LDT}(3, \overline{\alpha_2}, t_2) \arrow[r, leftrightarrow, "\text{Thm. 4.3.3}"'] & \mathsf{Conv3LDT}(1, \overline{\alpha_2}, t_2)\end{tikzcd}$$

Figure 4.2: Subquadratic reductions between different variants of Conv3LDT. Subscripts to coefficients α and t denote if the reduction allows changing the coefficients or not. All the reductions preserve the parameter $c \geq 1$ describing the size of the universe.

can never be part of a solution to Conv3LDT. Additionally, we take into account possible increase of the size of the universe and extension of the arrays, measured by d and e respectively.

Lemma 4.3.1. *Let $\bar{\alpha}$ be non-zero, $c, d, e \geq 1$, t be arbitrary constants. For every n, we can calculate in constant time values $n'' = \mathcal{O}(n)$ and $\bar{\perp}$ such that $\perp_1, \perp_2, \perp_3 \in [-(n'')^c, (n'')^c]$, $dn^c \leq (n'')^c$ and $en \leq n''$ and for all $\emptyset \neq S \subseteq [3]$ and $\bar{z} \in [-dn^c, dn^c]^3$ we have: $\sum_{i \in S} \alpha_i \perp_i + \sum_{i \in [3] \setminus S} \alpha_i z_i \neq t$.*

Proof. By Proposition 4.2.1, for given non-zero coefficients $\bar{\alpha}$ we can find coefficients $\bar{\beta}$ such that $\forall_{\emptyset \neq S \subseteq [3]} \sum_{i \in S} \alpha_i \beta_i \neq 0$. Let $\beta^* = \max_i |\beta_i|$ and $M = (\sum_i |\alpha_i|) n^c d$. We set $\perp_i = \beta_i M$ and $(n'')^c \geq \max\{2\beta^* M, (en)^c\}$. Clearly the above choice satisfies all the lower bounds for n'', so now we need to show the last required property. Suppose there exists set $\emptyset \neq S \subseteq [3]$ and elements $z_j \in [-dn^c, dn^c]$ for $j \in T = [3] \setminus S$ such that $\sum_{i \in S} \alpha_i \perp_i + \sum_{i \in T} \alpha_i z_i = t$. By the properties of $\bar{\beta}$, $\sum_{i \in S} \alpha_i \perp_i = \sum_{i \in S} \alpha_i \beta_i M \neq 0$, so $|\sum_{i \in S} \alpha_i \perp_i| \geq M$. As $|\sum_{i \in T} \alpha_i z_i| \leq \sum_{i \in T} |\alpha_i||z_i| \leq \sum_i |\alpha_i| n^c d \leq M/2$, we obtain $\sum_{i \in S} \alpha_i \perp_i + \sum_{i \in T} \alpha_i z_i \geq M/2 > t$, hence contradiction. □

Intuitively, this lemma allows us to transform an instance of Conv3LDT to a larger instance (at least by a factor of e) in which we can use the dummy elements $\perp$ that can never be part of a solution and the original non-$\perp$ elements can be multiplied by d and still fit within the universe $[-(n'')^c, (n'')^c]$. With all the properties at hand, we are ready to state the equivalence between 1- and 3-partite variants of Conv. We start with the adaptation of Lemma 4.2.3.

Lemma 4.3.2. *For every $c \geq 1$ all non-trivial 3-partite variants of Conv3LDT $_c$ are subquadratic-equivalent.*

Proof. We can modify the coefficients of the instance as in Lemma 4.2.3 but by updating both the values and positions of elements simultaneously in the same way. For instance, in the reduction $\mathsf{Conv3LDT}(3, \overline{\alpha}, 0) \leq_2 \mathsf{Conv3LDT}(3, \overline{\alpha}, t)$ we create the arrays $A_i'[k + y_i] = A_i[k] + y_i$, where y_i are from the extended Euclidean algorithm and satisfy that $\sum_i \alpha_i y_i = t$. Similarly, to show $\mathsf{Conv3LDT}(3, \overline{\alpha}, 0) \leq_2 \mathsf{Conv3LDT}(3, \overline{\beta}, 0)$ we define $q = \operatorname{lcm}(\beta_1, \beta_2, \beta_3)$ and set $A_i'[k\frac{\alpha_i q}{\beta_i}] = A_i[k]\frac{\alpha_i q}{\beta_i}$. As argued in Lemma 4.2.3, in the reductions the elements and their indices increase at most by the constant factor w, so in order to obtain an instance of Conv3LDT $_c$ we first apply Lemma 4.3.1 with $e = d = w$ and then apply the reduction. □

Theorem 4.3.3. *For all $c \geq 1$ and non-trivial coefficients $\bar{\alpha}$ and t we have: Conv3LDT $_c(1, \bar{\alpha}, t) \equiv_2$ Conv3LDT $_c(3, \bar{\alpha}, t)$.*

Proof. Recall that $n' = \frac{n-1}{2}$ and the array is indexed with $[-n', n']$. Assume that α and t specify non-trivial instances of $\mathsf{Conv3LDT}$, i.e. $\alpha_i \neq 0$ for all $i \in [3]$ and $\gcd(\alpha_1, \alpha_2, \alpha_3) \mid t$. As all the reductions preserve the parameter c for the size of the universe, we omit it while writing e.g. $\mathsf{Conv3LDT}(1, \overline{\alpha}, t)$. First, we show $\mathsf{Conv3LDT}(1, \overline{\alpha}, t) \leq_2 \mathsf{Conv3LDT}(3, \overline{\alpha}, t)$. As in Lemma 4.2.4, we use color-coding from Fact 4.2.5, with the domain of functions shifted by $n' + 1$, to determine to which array we should add the k-th element from the input array and then use $\perp$ from Lemma 4.3.1 (with $d = e = 1$) for all the remaining empty fields in the arrays.

In the reduction $\mathsf{Conv3LDT}(3, \overline{\alpha}, t) \leq_2 \mathsf{Conv3LDT}(1, \overline{\alpha}, t)$ we first seek solutions containing a repeated index (solutions with $|\{j_1, j_2, j_3\}| < 3$), in $\mathcal{O}(n)$ time. Now we need to find solutions with all the indices distinct. Again we use color-coding from Fact 4.2.5 obtaining $\mathcal{O}(\log^2 n)$ functions $[-n', n'] \to [3]$ and we consider each of them separately. For a single coloring function χ we set $A[k] = A_{\chi(k)}[k]$ for each $k \in [-n', n']$. Then we use the approach of Lemmas 4.2.8 and 4.2.12 to modify the entries of the array in such a way that every triple of elements forming a solution consists of elements from three distinct arrays. We consider two cases:

Case of $t \neq 0$. By the extended Euclidean algorithm, we can choose an integer triple $\bar{y}$ such that $\sum_i \alpha_i y_i = t$ and apply Lemma 4.2.7 on $\bar{\alpha}$ to obtain a triple $\bar{\gamma}$ of nonzero coefficients such that $\sum_i \alpha_i \gamma_i = 0$ and for every non-constant forbidden origin f we have $\sum_i \alpha_i \gamma_{f(i)} \neq 0$. Let $C = 1 + (\max_i \max\{|\gamma_i|, |y_i|\}) \cdot \sum_i |\alpha_i|$. Given the arrays A_1, A_2, A_3, we construct the array A by setting for each $k \in [-n', n']$:

$$A[k] = C^2(A_{\chi(k)}[k] - y_{\chi(k)}) + C\gamma_{\chi(k)} + y_{\chi(k)}$$

Similarly to Lemma 4.2.8, it follows that every valid solution of $\mathsf{Conv3LDT}(1, \overline{\alpha}, t)$ in A is a valid solution of $\mathsf{Conv3LDT}(3, \overline{\alpha}, t)$ on A_1, A_2, A_3 and vice versa. As the elements can increase at most by factor $2C^2$, in order to obtain an instance of $\mathsf{Conv3LDT}(1, \overline{\alpha}, t)$ with elements within the universe, we first run Lemma 4.3.1 with $e = 1, d = 2C^2$.

Case of $t = 0$. Suppose first that $\sum_i \alpha_i \neq 0$. As $t = 0$, in the above construction we do not need the part corresponding to the values y_i from extended Euclidean algorithm, so we can construct the array A by setting for each $k \in [-n', n']$:

$$A[k] = CA_{\chi(k)}[k] + \gamma_{\chi(k)}. \tag{4.1}$$

By the properties of $\bar{\gamma}$, this is enough for the case when $\sum_i \alpha_i \neq 0$, as every valid solution of $\mathsf{Conv3LDT}(1, \overline{\alpha}, t)$ in A is a valid solution of $\mathsf{Conv3LDT}(3, \overline{\alpha}, t)$ on A_1, A_2, A_3 and vice versa. The case of $\sum_i \alpha_i = t = 0$ is more involved and we consider it in two separate lemmas.

Lemma 4.3.4. *For all non-trivial coefficients $\bar{\alpha}$ such that $\sum_i \alpha_i = 0$ and every $n > 0$ we can calculate in $\mathcal{O}(n \cdot 2^{\mathcal{O}(\sqrt{\log n})})$ time an array $\mathcal{B}$ of n elements from $[n]$, indexed by $P = [-\frac{n-1}{2}, \frac{n-1}{2}]$, such that for every three pairwise distinct indices $\bar{j} \in P^3$ such that $\sum_i \alpha_i j_i = 0$ we have $\sum_i \alpha_i \mathcal{B}[j_i] \neq 0$.*

Proof. Without loss of generality, assume $\alpha_1, \alpha_2 > 0$. Let $P = [-n', n']$ be the set of all indices of the arrays. By Theorem 4.2.10, we partition P into $e = 2^{\mathcal{O}(\sqrt{\log n})}$ pairwise disjoint (α_1, α_2)-free sets $P_1, P_2, \ldots, P_e$ such that $P = \bigcup_j P_j$ in $\mathcal{O}(n \cdot 2^{\mathcal{O}(\sqrt{\log n})})$ time. Let position $p \in P$ belong to set P_{x_p}, $\textsc{Bin}(x)$ consists of all positions containing 1 in binary representation of x and $q = \sum_i |\alpha_i|+1$. Then we set $\mathcal{B}[p] = \sum_{k \in \textsc{Bin}(x_p)} q^k$.

Clearly the elements from $\mathcal{B}$ are positive and upper bounded by $q^{\lfloor \log e \rfloor + 1} = q^{\mathcal{O}(\sqrt{\log n})} = 2^{\mathcal{O}(\sqrt{\log n})} \leq n$. Now we need to show that for all triples $\bar{j} \in P^3$ of pairwise distinct elements such that $\sum_i \alpha_i j_i = 0$ it holds that $\sum_i \alpha_i \mathcal{B}[j_i] \neq 0$. By the choice of sufficiently big q: $\sum_i \alpha_i \mathcal{B}[j_i] = 0$ if and only if $\forall_{0 \leq k \leq \lfloor \log e \rfloor} \sum_{i : k \in \textsc{Bin}(x_{j_i})} \alpha_i = 0$. As $\sum_i \alpha_i = 0$ and $\alpha_i \neq 0$ for $i \in [3]$, for every $S \subseteq [3]$ we have that $\sum_{i \in S} \alpha_i = 0$ if and only if $S = \emptyset \vee S = [3]$ which translates to the fact that positions $x_{j_1}, x_{j_2}, x_{j_3}$ either all have 2^k in their binary representation or none of them has. Hence $\sum_i \alpha_i \mathcal{B}[j_i] = 0$ iff $x_{j_1}, x_{j_2}, x_{j_3}$ have exactly the same binary representation, so $x = x_{j_i}$ for all $i \in [3]$. This means that positions j_1, j_2, j_3 satisfying that $\sum_i \alpha_i j_i = 0$ belong to the same (α_1, α_2)-free set P_x, which is a contradiction. □

Lemma 4.3.5. *For all $c \geq 1$ and non-trivial coefficients $\bar{\alpha}, t$ such that $\sum_i \alpha_i = t = 0$ it holds $\mathsf{Conv3LDT}_c(3, \overline{\alpha}, 0) \leq_2 \mathsf{Conv3LDT}_c(1, \overline{\alpha}, 0)$.*

Proof. Similarly to Lemma 4.2.12, we filter out elements from each array using Behrend's set, but now by indices of the values. More precisely, without loss of generality, assume $\alpha_1, \alpha_2 > 0$ and we consider (α_1, α_2)-free sets. Let $P = [-n', n']$ be the set of all indices of the arrays. By Theorem 4.2.10, we partition P into $e = 2^{\mathcal{O}(\sqrt{\log n})}$ pairwise disjoint (α_1, α_2)-free sets $P_1, P_2, \ldots, P_e$ such that $P = \bigcup_j P_j$ in $\mathcal{O}(n \cdot 2^{\mathcal{O}(\sqrt{\log n})})$ time. For each $\bar{u} \in [e]^3$, we restrict each array A_i only to elements from positions P_{u_i} and construct an instance of $\mathsf{Conv3LDT}(1, \overline{\alpha}, t)$, similarly as in Equation (4.1).

However, now we cannot directly use the dummy symbols $\perp_i$ from Lemma 4.3.1 as we might consider three positions j_1, j_2, j_3 of the array A such that their elements originate from the same array A_μ, where $\mu = \chi(j_1) = \chi(j_2) = \chi(j_3)$ and they were all discarded from the array A_μ, that is $j_i \notin P_{u_\mu}$ for $i \in [3]$. Then using $\perp_\mu$ for each of them will result in a triple of elements satisfying $\sum_i \alpha_i A[j_i] = \sum_i \alpha_i \perp_\mu = 0$ that does not correspond to a valid solution of $\mathsf{Conv3LDT}(3, \overline{\alpha}, t)$. To overcome this issue, we will replace the dummy elements with elements of array $\mathcal{B}$ from Lemma 4.3.4.

Let $M = \sum_i |\alpha_i| n^c$, n'' satisfy $(n'')^c \geq 3CM$ and $\mathcal{B}$ be an array defined in Lemma 4.3.4 for coefficients $\bar{\alpha}$, consisting of n'' elements indexed by $[-\frac{n''-1}{2}, \frac{n''-1}{2}]$. As $c \geq 1$, all elements from $\mathcal{B}$ are smaller than M. While applying Equation (4.1), we replace the filtered out elements with a transformation of the corresponding elements from $\mathcal{B}[k]$:

$$A[k] = \begin{cases} C A_{\chi(k)}[k] + \gamma_{\chi(k)} & k \in P_{u_{\chi(k)}} \wedge k \in [-\frac{n-1}{2}, \frac{n-1}{2}] \\ C(2M + \mathcal{B}[k]) + \gamma_{\chi(k)} & \text{otherwise} \end{cases} \tag{4.2}$$

We say that an element $A[k]$ is discarded if $k \notin P_{u_{\chi(k)}}$ or $k \notin [-\frac{n-1}{2}, \frac{n-1}{2}]$. It is easy to see that any valid solution of $\mathsf{Conv3LDT}(3, \overline{\alpha}, t)$ on A_1, A_2, A_3 gives a valid solution of $\mathsf{Conv3LDT}(1, \overline{\alpha}, t)$ in A for some choice of $\bar{u} \in [e]^3$. Now we show the opposite direction. Consider a triple $\bar{j}$ of positions that form a solution of $\mathsf{Conv3LDT}(1, \overline{\alpha}, 0)$ for a particular choice of $\bar{u} \in [e]^3$. This means that j_i are pairwise distinct, $\sum_i \alpha_i j_i = 0$, $\sum_i \alpha_i A[j_i] = 0$. Let $S = \{i : j_i \in P_{u_i} \wedge i \in [3]\}$ be the set of indices of elements originating from a non-discarded element, and $T = [3] \setminus S$. Then:

$$\begin{aligned} 0 = \sum_i \alpha_i A[j_i] &= \sum_{i \in S} \alpha_i (C A_{\chi(j_i)}[j_i] + \gamma_{\chi(j_i)}) + \sum_{i \in T} \alpha_i (C(2M + \mathcal{B}[j_i]) + \gamma_{\chi(j_i)}) \\ &= \sum_i \alpha_i \gamma_{\chi(j_i)} + C \left(\sum_{i \in S} \alpha_i A_{\chi(j_i)}[j_i] + \sum_{i \in T} \alpha_i \left(2M + \mathcal{B}[j_i]\right) \right) \end{aligned}$$

As the above expression equals 0 we have $\sum_i \alpha_i \gamma_{\chi(j_i)} = 0$, so by Lemma 4.2.7, $(\chi(j_1), \chi(j_2), \chi(j_3))$ is either a constant ($\mu = \chi(j_1) = \chi(j_2) = \chi(j_3)$) or not forbidden origin. Next, the coefficient by C also equals 0, so let $(\star) = \sum_{i \in S} \alpha_i A_{\chi(j_i)}[j_i] + \sum_{i \in T} \alpha_i \left(2M + \mathcal{B}[j_i]\right) = 0$. There are three cases to consider depending on the number of indices corresponding to non-discarded elements:

$|S| = 3$: We have $0 = (\star) = \sum_i \alpha_i A_{\chi(j_i)}[j_i] = 0$. If all $\chi(j_i)$ are equal to some μ, all elements originate from array A_μ. However, we restricted A_μ only to elements on positions from (α_1, α_2)-free set P_μ, which contradicts the fact that $\sum_i \alpha_i j_i = 0$. Otherwise, $(\chi(j_1), \chi(j_2), \chi(j_3))$ is a non-forbidden origin, so we obtain a valid solution to the original instance of $\mathsf{Conv3LDT}(3, \overline{\alpha}, t)$.

$|S| = 0$: We have $0 = (\star) = \sum_i \alpha_i (2M + \mathcal{B}[j_i]) = \sum_i \alpha_i \mathcal{B}[j_i]$. which contradicts the fact that this sum is non-zero, by Lemma 4.3.4.

else: For $|S| \in \{1, 2\}$ we have $|T| \in \{1, 2\}$. Let $(\star_1) = \sum_{i \in T} 2M\alpha_i = 2M \sum_{i \in T} \alpha_i$ so $|(\star_1)| \geq 2M$ because $\alpha_i \neq 0$ (as we consider non-trivial coefficients $\bar{\alpha}$) and $\sum \alpha_i = 0$ (by the assumption). Next, let $(\star_2) = (\star) - (\star_1) = \sum_{i \in S} \alpha_i A_{\chi(j_i)}[j_i] + \sum_{i \in T} \alpha_i \mathcal{B}[j_i]$ and then $|(\star_2)| \leq \sum_i |\alpha_i| n^c \leq M$ because elements from A and $\mathcal{B}$ are from $[-n^c, n^c]$. Thus $|(\star)| = |(\star_1) + (\star_2)| \geq M$ which contradicts that $(\star) = 0$.

Finally, all the elements from A are from $[-(n'')^c, (n'')^c]$, because $A_{\chi(k)}[k] \in [-n^c, n^c]$, $2M + \mathcal{B}[k] < 3M$ and $(n'')^c \geq 3CM = 3C \sum_i |\alpha_i| n^c$, so the obtained array is an instance of $\mathsf{Conv3LDT}(1, \overline{\alpha}, 0)$. To summarize, we created e^3 such instances of $\mathsf{Conv3LDT}(1, \overline{\alpha}, 0)$ with arrays of $n'' = \mathcal{O}(n)$ elements, where $e = 2^{\mathcal{O}(\sqrt{\log n})}$. If one could solve $\mathsf{Conv3LDT}(1, \overline{\alpha}, 0)$ in $\mathcal{O}((n'')^{2-\varepsilon})$ time, combining that with our reduction will give an algorithm solving an instance of $\mathsf{Conv3LDT}(3, \overline{\alpha}, 0)$ in time $\mathcal{O}(e^3 (n'')^{2-\varepsilon}) = \mathcal{O}(n^{2-\delta})$ for every $0 < \delta < \varepsilon$. □

This concludes the subquadratic reduction from an instance of $\mathsf{Conv3LDT}_c(3, \bar{\alpha}, 0)$ to $2^{\mathcal{O}(\sqrt{\log n})}$ instances of $\mathsf{Conv3LDT}_c(1, \bar{\alpha}, 0)$. □

4.4 Equivalences between 3LDT and Conv3LDT

In this section, we show reductions between 3-partite variants of 3LDT and Conv3LDT (see Figure 4.3 for an overview). We start with a folklore reduction $\mathsf{Conv3LDT}_c(3, \overline{\alpha}, 0) \leq_2 \mathsf{3LDT}_{c+1}(3, \bar{\alpha}, 0)$ that increases the size of the universe by n.

$$\begin{tikzcd}
 & \mathsf{Conv3LDT}(3,\overline{\alpha_2},0) \arrow[r, "c\to c+1", "\text{Lem. 4.4.1}"'] & \mathsf{3LDT}(3,\overline{\alpha_2},0) \arrow[d, "\text{Lem. 4.2.3}"'] & \\
\mathsf{Conv3LDT}(1,\overline{\alpha_1},t_1) \arrow[r, leftrightarrow, "\text{Thm. 4.3.3}"'] & \mathsf{Conv3LDT}(3,\overline{\alpha_1},t_1) \arrow[u, "\text{Lem. 4.3.2}"] & \mathsf{3LDT}(3,\overline{\alpha_3},t_3) \arrow[d, "\text{Lem. 4.2.3}"'] \arrow[r, leftrightarrow, "\text{Thm. 4.1.3}"'] & \mathsf{3LDT}(1,\overline{\alpha_4},t_4) \\
 & \mathsf{Conv3SUM} \arrow[u, "\text{Lem. 4.3.2}"] & \mathsf{3SUM} \arrow[l, "c-1\leftarrow c"', "\text{Thm. 4.4.2}"] &
\end{tikzcd}$$

Figure 4.3: Subquadratic reductions between different variants of Conv3LDT and 3LDT. Subscripts to coefficients α and t denote if the reduction allows changing the coefficients or not. Only two reductions change the value of parameter c, all other reductions preserve it.

Lemma 4.4.1. *For all $c \geq 1$ and $\bar{\alpha}$, an instance of Conv3LDT$_c(3, \overline{\alpha}, 0)$ reduces in linear time to an instance of* 3LDT$_{c+1}(3, \bar{\alpha}, 0)$.

Proof. We create sets $S_i = \{A_i[j] \cdot W + j : j \in [-n', n']\}$ where $W = (\sum_i |\alpha_i| + 1) \cdot n$. By the choice of W, a solution $\bar{x}$ where $x_i \in S_i$ satisfies $\sum_i \alpha_i x_i = 0$ iff both $\sum_i \alpha_i (x_i \bmod W) = 0$ and $\sum_i \alpha_i \lfloor \frac{x_i}{W} \rfloor = 0$, which exactly corresponds to the condition on both the values and indices of elements from the instance of Conv3LDT$(3, \overline{\alpha}, 0)$. The elements from S_i are bounded by $W \cdot n^c + \frac{n}{2} \leq n^c \cdot n \cdot (\sum_i |\alpha_i| + 2)$ so we let $n'' = n\lceil (\sum_i |\alpha_i| + 3)^{1/(c+1)} \rceil$. Then we have an instance of 3LDT$_{c+1}$ with sets of at most n'' elements from $[-(n'')^{c+1}, (n'')^{c+1}]$. By Lemma 4.2.2 we can reduce it to an instance of 3LDT$_{c+1}(3, \bar{\alpha}, 0)$ on sets with $n''' = \mathcal{O}(n'') = \mathcal{O}(n)$ elements. □

Next, we show how to reduce LDT to Conv3LDT while controlling the exponent in the size of the universe. The starting point is the celebrated proof by Pătraşcu who showed a randomized reduction from 3SUM to Conv3SUM (without controlling the size of the universe). We follow a later deterministic approach of Chan and He [CH20], except that we need to tweak it to control the size of the universe and extend it to arbitrary variants of LDT.

Recall that in the 3-partite variant of 3SUM we are given three sets S_1, S_2, S_3 and need to check if there are three numbers x_1, x_2, x_3 such that $x_1 + x_2 = x_3$ and $x_i \in S_i$ for $i \in [3]$. Earlier we defined 3SUM for universe $[-n^3, n^3]$, but in this section we consider arbitrary universe $[n^c, n^c]$ for $c \geq 2$, that is variants 3LDT$_c(3, (1, 1, -1), 0)$ for any $c \geq 2$. Similarly, Conv3SUM denotes a family of variants Conv3LDT$_c(3, (1, 1, -1), 0)$ for any $c \geq 1$.

Theorem 4.4.2. *For all $c \geq 2$, an instance of 3SUM$_c$ on 3 sets with n numbers reduces in n* polylog *n time to* polylog *n instances of Conv3SUM$_{c-1}$ on arrays of $\mathcal{O}(n)$ elements.*

Proof. We define a semi-instance of 3SUM on three sets S_i of at most n elements from $[-n^c, n^c]$ to be *m-spread* if for all $i \in [3]$ we have $|\{x \bmod m : x \in S_i\}| = |S_i|$, that is all elements of each set have distinct remainders modulo m. Note that this implies that the sets have at most m elements each. An important tool in our reduction is the deterministic recursive algorithm for reducing 3SUM to a number of instances of Conv3SUM described by Chan and He [CH20]. We slightly reformulate their algorithm:

Lemma 4.4.3 (Section 5 [CH20])**.** *There exists a deterministic reduction $\mathcal{R}$ such that for every instance $\mathcal{I}$ of 3SUM$_c$ on three sets of n elements and parameter t satisfying that $t \geq \operatorname{polylog} n$, $\mathcal{R}$ reduces $\mathcal{I}$ to $k = t^3 \operatorname{polylog} n$ instances $\mathcal{I}_1, \ldots, \mathcal{I}_k$ of 3SUM on numbers from $[-n^c, n^c]$ where the j-th instance $\mathcal{I}_j$ is accompanied with a number $m_j \in [\frac{n}{4}, n)$ such that $\mathcal{I}_j$ is m_j-spread. The reduction runs in $\mathcal{O}(n^{1.5} \operatorname{polylog} n)$ time.*

Proof. We provide an informal overview of the algorithm of Chan and He. Given an instance of 3SUM with three sets S_i, we find a number m and hash elements of every set into buckets modulo m. Let t be a parameter. We call an element *bad* if its bucket contains more than t elements, otherwise *good*. The number $m \in \Theta(n)$ will be chosen in $n^{1.5} \operatorname{polylog} n$ time in such a way that for every $i \in [3]$ the total number of bad elements in S_i is at most $d_1 \frac{|S_i|^2}{mt} \log^{d_2} n$ for some constants $d_1, d_2 > 0$.[4] In order to find a triple $\bar{x}$ such that $x_i \in S_i$ for $i \in [3]$ and $x_1 + x_2 = x_3$ we proceed as follows:

- First we find solutions in which all elements x_i are good. We consider t^3 triples $\bar{p} \in [t]^3$ and for each of them we restrict the problem as follows. For each $i \in [3]$ we only keep the p_i-th element from each of the buckets of set S_i. This gives us t^3 m-spread instances of 3SUM.

- In order to find a solution with at least one bad element x_i, e.g. x_3, we call the procedure recursively to solve 3SUM for the following sets: S_1, S_2 and the third set consisting of only the bad elements from S_3. We proceed similarly for the case when x_1 or x_2 is bad.

We require that $t \geq 8d_1 \log^{d_2} n = \operatorname{polylog} n$ and choosing $m \in [\frac{n}{4}, n)$ will give us the number of bad elements in set S_i at most $d_1 \frac{|S_i|^2}{mt} \log^{d_2} n \leq \frac{|S_i|^2}{2n}$. Suppose that a set was repeatedly replaced with the set of its bad elements and after the j-th recursive step its size is $\frac{n}{k_j}$. As $(\frac{n}{k_j})^2 \frac{1}{2n} = \frac{n}{2{k_j}^2}$, we have $k_{j+1} \geq 2k_j^2$. By setting $k_0 = 1$, we therefore obtain $k_j \geq 2^{2^j - 1}$. Thus the depth of the recursion tree is $h = \mathcal{O}(\log \log n)$ and each node makes at most 3 recursive calls by restricting only to bad elements from S_i for $i \in [3]$. Hence in total there are at most $t^3 3^h = t^3 \operatorname{polylog} n$ instances of m-spread 3SUM, where the values of m may differ between different nodes of the recursion tree. □

Now we show how to reduce a m-spread instance of 3SUM to an instance of Conv3SUM, decreasing the size of the universe by a factor of m.

Lemma 4.4.4. *Let $m \in [\frac{n}{4}, n)$. We can reduce in linear time a m-spread instance of 3SUM with numbers from $[-n^c, n^c]$ to an instance of Conv3SUM with arrays on $n'' = \mathcal{O}(n)$ elements from $[-(n'')^{c-1}, (n'')^{c-1}]$.*

[4] The choice of m with desired properties is described in detail in Section 5 of [CH20]. Our only modification is to set $m_1 = m_2 = \sqrt{n}$ which gives an algorithm for finding m in $n^{1.5} \operatorname{polylog} n$ time.

Proof. Let $\mathcal{I}^m$ be the m-spread instance of 3SUM, $h(x) = x \bmod m$ and $v(x) = \lfloor \frac{x}{m} \rfloor$. We create the instance $\mathcal{I}$ of Conv3SUM with arrays A_i indexed with $[-2m+1, 2m-1]$ where the i-th of them is initially filled with $\bot_i$ defined in Lemma 4.3.1. For $i \in [3]$, for all $x \in S_i$ we set $A_i[h(x)] = v(x)$. Additionally, for all $z \in S_3$ we set $A_3[h(z)+m] = v(z) - 1$. Note that no two elements were put in the same cell as the sets S_i are m-spread.

Now we show that this reduction is correct. Suppose there is a triple $\bar{x} : x_i \in S_i$ for $i \in [3]$ and $x_1 + x_2 = x_3$. There are two cases two consider:

- if $h(x_1) + h(x_2) < m$, we have $v(x_1) + v(x_2) = v(x_3)$ and $h(x_1) + h(x_2) = h(x_3)$, so the triple $(h(x_1), h(x_2), h(x_3))$ is a solution for $\mathcal{I}$,
- if $h(x_1) + h(x_2) \geq m$, we have $v(x_1) + v(x_2) = v(x_3) - 1$ and $h(x_1) + h(x_2) = h(x_3) + m$, so the triple $(h(x_1), h(x_2), h(x_3) + m)$ is a solution for $\mathcal{I}$.

In the other direction, suppose there is a solution $\bar{y}$ for $\mathcal{I}$ and we show that it gives a solution for $\mathcal{I}^m$. Indeed, by definition we have $y_1 + y_2 = y_3$ and $A_1[y_1] + A_2[y_2] = A_3[y_3]$ so $y_1 + y_2 + m(A_1[y_1] + A_2[y_2]) = y_3 + mA_3[y_3]$. From the properties of $\bot_i$, every solution of $\mathcal{I}$ contains only the non-$\bot$ values, so every y_i must be obtained from a value $x_i \in S_i$. We set $x_i = y_i + mA_i[y_i]$ for $i \in [3]$ and claim that it is a solution for $\mathcal{I}^m$. Clearly, for $i = 1, 2$ it holds that $x_i \in S_i$. For $i = 3$, $x_3 = y_3 + mA_3[y_3] = (y_3 - m) + m(A_3[y_3] + 1)$, so $x_3 \in S_3$ both for $y_3 < m$ and $y_3 \geq m$.

Recall that the m-spread instance $\mathcal{I}^m$ consists of sets of at most $m \leq n$ elements from $[-n^c, n^c]$. Then $v(x) \in [-\frac{n^c}{m}, \frac{n^c}{m}]$ for $x \in S_i$ and $i \in [3]$ and the created arrays have elements on the positions from $[-(2m-1), 2m-1]$. As we need $\bot_i$ for the non-occupied elements of the arrays, we apply Lemma 4.3.1 for n, $d = 4$ (as we need $\frac{n^c}{m} \leq dn^{c-1}$) and $e = 4$ (as the obtained array has $4m - 1 \leq 4n$ elements). This gives us $n'' = \mathcal{O}(n)$ and $\bar{\bot}$ such tat we can have an array of n'' entries, unoccupied elements from the i-th array are replaced with $\bot_i$ and all entries (occupied or not) are from $[-(n'')^{c-1}, (n'')^{c-1}]$. Hence the claim follows. □

We set $t = \operatorname{polylog} n$ such that it satisfies the requirement from Lemma 4.4.3. Combining the two above lemmas we obtain a reduction from 3SUM on 3 sets with n numbers from $[-n^c, n^c]$ to $k = t^3 \operatorname{polylog} n = \operatorname{polylog} n$ instances of Conv3SUM on 3 arrays on $n'' = \mathcal{O}(n)$ elements from $[-n''^{(c-1)}, n''^{(c-1)}]$, which concludes the proof. □

By Theorem 4.4.2 and Lemma 4.4.1, we immediately obtain the following corollary:

Corollary 4.4.5. *For all $c \geq 2$ it holds $3LDT_c(3, (1,1,-1), 0) \equiv_2$ Conv3LDT$_{c-1}(3, (1,1,-1), 0)$.*

Now we can combine Lemma 4.4.1, Theorem 4.1.3, Theorem 4.4.2, Lemma 4.3.2 and Theorem 4.3.3 as presented in Figure 4.3 to obtain the following theorem:

Theorem 4.1.6. *For every $c \geq 2$, all non-trivial 1- and 3-partite variants of 3LDT$_c$ and of Conv3LDT $_{c-1}$ are subquadratic-equivalent.*

Finally, we discuss the subtlety of the negative indices in the arrays that we introduced in the definition of Conv3LDT. A natural question is whether we can achieve a similar result with only positive indices in Conv3LDT. Consider a non-trivial variant of Conv3LDT with all $\alpha_i > 0$. There is a constant number of triples of indices $j_1, j_2, j_3 \in [n]$ such that $\sum_i \alpha_i j_i = t$ which we can all check in linear-time. Similarly for the variants with all $\alpha_i < 0$. Therefore, in those cases there is a fast algorithm for 3SUM which makes such an equivalence unlikely.

For that reason, in the definition of Conv3LDT we allow negative indices of arrays. However, if not all coefficients α have the same sign, we can work with instances of arrays with only positive indices:

Lemma 4.4.6. *For all $c \geq 1$, coefficients $\bar{\alpha}$ not all having the same sign, and arbitrary t, every non-trivial variant of Conv3LDT$_c(3, \bar{\alpha}, t)$ is equivalent to the same variant of Conv3LDT$_c(3, \bar{\alpha}, t)$, but on arrays with only positive indices.*

Proof. Suppose the original instance operates on arrays $A_i[-n', n']$. Let $\delta = n' + 1$ and we show how to choose $\bar{\beta}$ that satisfy $\sum_i \alpha_i \beta_i = 0$ and $\beta_i > 0$ for all $i \in [3]$. As not all coefficients $\bar{\alpha}$ have the same sign, without loss of generality we can assume that $\alpha_1 > 0$ and $\alpha_2, \alpha_3 < 0$. Then it suffices to set $\beta_1 = -\alpha_2 - \alpha_3$ and $\beta_2 = \beta_3 = \alpha_1$ that satisfy the above properties.

We shift the i-th array by $\delta\beta_i$ elements, that is we set $A'_i[k] = A_i[k - \delta\beta_i]$. Then any triple $\bar{j}'$ of indices in A' corresponds to indices $\bar{j} - \delta\bar{\beta}$ in A and we have $\sum_i \alpha_i j'_i = \sum_i \alpha_i (j_i - \delta\beta_i) = \sum_i \alpha_i j_i - \delta(\sum_i \alpha_i \beta_i) = \sum_i \alpha_i j_i$. All the elements from arrays A_i are moved to entries of A'_i with indices from $[1, n'']$ where $n'' = n' + \delta \max_i \beta_i = \mathcal{O}(n')$ so there is a solution of Conv3LDT$(3, \overline{\alpha}, t)$ for arrays A iff there is a solution for tables A'. Depending on the value of β_i, we have to fill either the first $\delta\beta_i - n' - 1$ elements of A'_i with $\perp_i$, the last $n'' - (n' + \delta\beta_i)$ elements or both at the beginning and end of A'_i. To this end we apply Lemma 4.3.1 with $d = 1$ and $e = \lceil n''/n \rceil$ to obtain n''' and $\bar{\perp}$ such that $\perp_1, \perp_2, \perp_3$ have the desired properties. Then we use $\bar{\perp}$ to fill the non-existent elements in the arrays and finally obtain arrays on n''' elements from $[-(n''')^c, (n''')^c]$ on positions $[1, n''']$. □

Corollary 4.4.7. *For all $c \geq 2$, all non-trivial 1- and 3-partite variants of 3LDT$_c$ and Conv3LDT$_{c-1}$ with non-trivial coefficients such that $\bar{\alpha}$ not all having the same sign are subquadratic-equivalent, even if we operate on arrays with only positive indices.*

In particular, in Average we have $\bar{\alpha} = (1, 1, -2)$, so not all the coefficients have the same sign and we can obtain a similar result for Average, which was considered by Erickson:

Corollary 4.4.8. *For all $c \geq 2$, Average$_c$ is subquadratic-equivalent to ConvAverage$_{c-1}$, even if we operate on arrays with only positive indices.*

4.5 Reducing the size of the universe

Fischer et al. [FKP24] showed a deterministic reduction of the size of the universe for 3SUM from arbitrarily big to cubic. In this section we show how to extend their result to all 3-partite variants of 3LDT. Next, we show how to use reductions from the previous sections to show universe reduction for 1-partite 3LDT and variants of Conv3LDT.

At the input we are given n numbers of length $\log U$. For $U > 2^{n^{\Omega(1)}}$, the trivial $\mathcal{O}(n^2 \log U)$ algorithm is already subquadratic in the size of the input, so we can focus only on the cases when $U < 2^{n^{o(1)}}$. We will operate on numbers of length $\mathcal{O}(\log U)$: add them, subtract, multiply by a number at most $\mathcal{O}(n)$ or calculate modulo a small number. All these operations take $\mathcal{O}(\log U) = n^{o(1)}$ time which is irrelevant from the perspective of subquadratic reductions. For this reason we will not explicitly mention this factor in the time complexity of the presented reductions.

Theorem 4.5.1. *For all non-trivial coefficients $\bar{\alpha}$ and t, if an instance of $\mathsf{LDT}_3(3, \bar{\alpha}, t)$ of sets of n numbers over $[-n^3, n^3]$ can be solved in deterministic time $\mathcal{O}(n^{2-\varepsilon})$ for some $\varepsilon > 0$, then $\mathsf{LDT}_?(3, \bar{\alpha}, t)$ on sets of n numbers from $[-U, U]$ can be solved in deterministic time $\mathcal{O}(n^{2-\varepsilon'} \log^c U)$ for some constants $\varepsilon', c > 0$.*

Proof. We show how to extend the proof of Theorem 2 from Fischer et al. [FKP24] from 3SUM to 3LDT. For that purpose we restate the relevant lemmas adjusted to the general, 3-partite case. We do not restate the full proofs, we only highlight what needs to be changed.

Following [FKP24], we use the notation $\tilde{\mathcal{O}}(T) = T(\log T)^{\mathcal{O}(1)}$. To avoid clutter, in this section we implicitly allow instances of 3LDT to have at most n numbers, instead of exactly n numbers. Then, by applying Lemma 4.2.2 we can transform each such instance to contain exactly n elements from $[-n^3, n^3]$, as in the original definition.

Lemma 4.5.2 (Lemma 11 [FKP24])**.** *Let $\bar{\alpha}$ and t be non-trivial coefficients and $0 \leq \mu < 3, \delta > 0$. There is a deterministic algorithm that, given a $3\mathsf{LDT}_?(3, \bar{\alpha}, t)$ instance A on sets A_1, A_2, A_3 of n numbers over $[-U, U]$, either finds a positive modulus $m \in [n^\mu, 2n^\mu)$ such that*

$$|\{(x_1, x_2, x_3) \in A_1 \times A_2 \times A_3 : \sum_i \alpha_i x_i \equiv t \pmod{m}\}| \leq n^{3-\mu+\delta} (\log U)^{\mathcal{O}(1/\delta)}$$

or decides that A is a yes-instance. The algorithm runs in $\tilde{\mathcal{O}}(n^{\max(\mu,1)+\delta})$ time.

Proof. In order to handle the more general case, there are only slight differences with respect to the original proof. First, in our applications the number of instances g always equals 1 so we can skip it. Second, while computing $S(m \cdot p)$ with FFT we also need to take into account the coefficients $\bar{\alpha}$.

Finally, we need to alter the analysis of the probability that a random prime p divides $q = \sum_i \alpha_i x_i - t$ assuming that $q \neq 0$. Let $r = 1 + \sum_i |\alpha_i|$. As t is constant, we have $q \in [-rU, rU]$. Following the analysis from the original proof, the aforementioned probability is at most

$$\frac{\log_{n^\delta}(rU)}{\Omega(n^\delta / \log n)} = \mathcal{O}(n^{-\delta} \log U)$$

and the rest of the analysis of the correctness follows. □

Lemma 4.5.3 (Lemma 12 [FKP24])**.** *Let $A_1, A_2, A_3 \subseteq [-U, U]$ be sets of size n, and let $g \geq 1$. We can deterministically construct a partition of sets A_1, A_2, A_3 into subsets $A_i^1, \dots, A_i^g$ for $i \in [3]$ where $A_i = \bigcup_{j \in [g]} A_i^j$, and a set of triples $R \subseteq [g]^3$ of size $\mathcal{O}(g^2)$ such that:*

- *Each set $A_i^{j_i}$ has size $\mathcal{O}(n/g)$ can be covered by an interval of length $\mathcal{O}(U/g)$.*
- *For all triples $x_1, x_2, x_3 \in A_1 \times A_2 \times A_3$ with $\sum_i \alpha_i x_i = t$, there is some $(j_1, j_2, j_3) \in R$ with $x_i \in A_i^{j_i}$ for $i \in [3]$.*

The algorithm runs in $\tilde{\mathcal{O}}(n + g^2)$.

Proof. Our analysis is a slight generalization of the proof from [FKP24]. We only carefully describe how to alter the definitions in the original proof to handle all possible signs of coefficients in $\bar{\alpha}$. First, similarly as in [FKP24], for ever $i \in [3]$ we partition set A_i into subsets $A_i^1, \dots, A_i^g$ such that for each $j \in [g]$:

- $|A_i^j| \leq 2n/g$, and
- $\max(A_i^j) - \min(A_i^j) \leq 2U/g$.

We can find such a partition greedily, by iterating over the sorted elements of A_i until either of the two required conditions is violated. As every condition separately generates at most $g/2$ sets, in total we obtain at most g sets. We assume that the sets are in the natural order, that is $\max(A_i^j) < \min(A_i^{j+1})$ for all $j \in [g-1]$.

Now we show how to construct the set $R \subseteq [g]^3$ of $\mathcal{O}(g^2)$ triples such that for all triples $x_1, x_2, x_3 \in A_1 \times A_2 \times A_3$ with $\sum_i \alpha_i x_i = t$, there is some $(j_1, j_2, j_3) \in R$ with $x_i \in A_i^{j_i}$ for all $i \in [3]$. First, we add to R all triples containing at most two distinct elements, there are $g + g \cdot (g-1) \cdot 3 = \mathcal{O}(g^2)$ of them. Now we need to add triples of three distinct elements.

Let xA be the set $\{x \cdot a : a \in A\}$. We construct the set R as follows. Suppose that α_3 is positive. For every pair $(i, j) \in [g]^2$, we binary search the smallest $k_1 \in [g]$ such that $\max(\alpha_1 A_1^i) + \max(\alpha_2 A_2^j) + \max(\alpha_3 A_3^{k_3}) \geq t$ and the largest $k_2 \in [k_1, g]$ such that $\min(\alpha_1 A_1^i) + \min(\alpha_2 A_2^j) + \min(\alpha_3 A_3^{k_3}) \leq t$. Then for all $k_1 \leq k \leq k_2$ we add to R the triple (i, j, k). The case of negative α_3 is symmetric, we swap roles of k_1 and k_2.

Observe that the above approach is correct (we do not miss any relevant triple in R), it runs in $\tilde{\mathcal{O}}(g^2 + |R|)$ time and the size of R is $\mathcal{O}(g^3)$. By the above construction, we do not add to R triples (i, j, k) such that either:

1. $\min(\alpha_1 A_1^i) + \min(\alpha_2 A_2^j) + \min(\alpha_3 A_3^k) > t$, or
2. $\max(\alpha_1 A_1^i) + \max(\alpha_2 A_2^j) + \max(\alpha_3 A_3^k) < t$.

Clearly, for such triples there is no triple $(x_1, x_2, x_3) \in A_1^i \times A_2^j \times A_3^k$ such that $\sum_i \alpha_i x_i = t$. We call such a triple (i, j, k) *immediate*. Now we show that the size of R is actually $\mathcal{O}(g^2)$ as desired.

Similarly as in [FKP24], consider the partially ordered set $P = ([g]^3, \prec)$ where $(i, j, k) \prec (i', j', k')$ if and only if $\max(\alpha_1 A_1^i) < \min(\alpha_1 A_1^{i'})$, $\max(\alpha_2 A_2^j) < \min(\alpha_2 A_2^{j'})$ and $\max(\alpha_3 A_3^k) < \min(\alpha_3 A_3^{k'})$. Note that this is not the standard lexicographical order, as some α_i might be negative. Observe that for two triples $(i, j, k), (i', j', k')$ such that $(i, j, k) \prec (i', j', k')$, at least one of them is immediate. Indeed, suppose the contrary, that they both falsify conditions (1) and (2). Then:

$$\begin{aligned} t &\leq \max(\alpha_1 A_1^i) + \max(\alpha_2 A_2^j) + \max(\alpha_3 A_3^k) && \text{as } (i,j,k) \text{ falsifies (2)} \\ &< \min(\alpha_1 A_1^{i'}) + \min(\alpha_2 A_2^{j'}) + \min(\alpha_3 A_3^{k'}) && \text{as } (i,j,k) \prec (i',j',k') \\ &\leq t && \text{as } (i',j',k') \text{ falsifies (1)} \end{aligned}$$

which is a contradiction. Thus, on any chain (totally ordered set of elements) in P there is at most one non-immediate triple in R. Hence it suffices to show that we can cover P with $\mathcal{O}(g^2)$ chains. Let $\operatorname{sgn}(x) = \frac{x}{|x|}$[5]. For all $a, b \in [-(g-1), \ldots, (g-1)]$ we take the chain $C_{a,b} = \{(f(t, \operatorname{sgn}(\alpha_1)), f(t, \operatorname{sgn}(\alpha_2)) + a, f(t, \operatorname{sgn}(\alpha_3)) + b : t \in [g]\} \cap [g]^3$ where $f(t, x) = t$ if $x = 1$ and $g + 1 - t$ otherwise. Intuitively, function f ensures that the specific coefficient in the chain is either increasing or decreasing, depending on the sign of the corresponding coefficient of α. Then any triple $(i, j, k) \in [g]^3$ is covered by the chain $C_{a,b}$ for $a = j - i$ if $\operatorname{sgn}(\alpha_1) = \operatorname{sgn}(\alpha_2)$ or $a = j + i - (g + 1)$ if $\operatorname{sgn}(\alpha_1) \neq \operatorname{sgn}(\alpha_2)$ and b calculated in the same way.

This concludes the proof that $|R| = \mathcal{O}(g^2)$ and hence the running time is $\tilde{\mathcal{O}}(n + g^2)$. □

This immediately implies:

Corollary 4.5.4 (Corollary 13 [FKP24]). *For any $g \geq 1$, a given $3LDT_?(3, \bar{\alpha}, t)$ instance on sets of n numbers over the universe $[-U, U]$ can be deterministically reduced to $\mathcal{O}(g^2)$ instances on sets of at most $\mathcal{O}(n/g)$ numbers over the universe $[-U', U']$ where $U' = \mathcal{O}(U/g)$. The running time of the reduction is $\tilde{\mathcal{O}}(ng)$.*

Observation 4.5.5 (Observation 15). *Let $U \geq U'$. An instance of $3LDT_?(3, \bar{\alpha}, t)$ on sets of n numbers over $[-U, U]$ can be reduced to $\mathcal{O}((\frac{U}{U'})^3)$ many instances of $3LDT_?(3, \bar{\alpha}, t)$ over $[-U', U']$ on sets of at most n numbers each.*

Proof. Exactly the same as the original one. □

Now we show how to extend the proof of Theorem 2 from [FKP24] from 3SUM to 3LDT. Let $\delta = \varepsilon/32, \mu = 2 - 2\delta$ and $\alpha = \frac{1}{2} + 2\delta$. At a high level, the authors of [FKP24]:

1. apply Lemma 4.5.2 to find a modulus m that gives few pseudo-solutions (triples $x_1, x_2, x_3 \in A_1 \times A_2 \times A_3$ such that $\sum_i \alpha_i x_i \equiv t \pmod{m}$),

2. reduce listing pseudo-solutions to listing solutions,

[5] In this paragraph we only consider the sign of coefficients α. They are always non-zero as the variant of 3LDT that we consider is non-trivial.

3. apply self-reduction from Corollary 4.5.4 to obtain instances on sets of at most $n^{1-\alpha}$ numbers over $[-U', U']$ for $U' = \mathcal{O}(m/n^\alpha) = \mathcal{O}(n^{\mu-\alpha})$,

4. apply reduction from Observation 4.5.5 to further reduce the universe size to $\mathcal{O}(n^{3-3\alpha})$.

We already showed how to adapt steps 1,3 and 4 to the reduction for 3LDT. For step 2, the authors use the 1-to-$\mathcal{O}(1)$ correspondence between pseudo-solutions within set A and solutions within set $A' = \{a \bmod m, (a \bmod m) + m : a \in A\}$. In the case of 3LDT this is not sufficient, because $\sum_i \alpha_i x_i \equiv t \pmod{m}$ if and only if $\sum_i \alpha_i (x_i \bmod m) - t = m \cdot k$ for $k \in \{-r, \ldots, r\}$ where $r = 1 + \sum_i |\alpha_i|$.

Let $d = \gcd(x_1, x_2, x_3)$. We apply the Chinese remainder theorem to calculate integer triple $\bar{y}$ such that $\sum_i \alpha_i y_i = \gcd(x_1, x_2, x_3) = d$. As coefficients $\bar{\alpha}$ and t are non-trivial, we have $d|t$. For every $k \in \{-r, \ldots, r\}$ we have $d|mk$, so we can write $\sum_i \alpha_i (x_i \bmod m) - t = \frac{mk}{d}(\sum_i \alpha_i y_i)$ which is equivalent to

$$\sum_i \alpha_i \left(x_i \bmod m - \frac{mk}{d} y_i \right) = t$$

This gives us a constant number of instances of 3LDT over $[-cm, cm]$ for $c = \mathcal{O}(1)$ to consider. Then the original instance A is a yes-instance if and only if at least on of the created instances has a valid solution or step 1 searching for the good modulus m returned that A is a yes-instance. The constant c is subsumed during the step 3 and the analysis of the remaining steps follows. □

Recall Definition 4.1.1 of subquadratic reduction that we denote with $\leq_2$. In our case the size of input is $\mathcal{O}(n \log U)$, and we consider universes of size at most $2^{n^{o(1)}}$. Then $\mathcal{O}(n^{2-\varepsilon'} \log^c U)$ is subquadratic in $\mathcal{O}(n \log U)$, so the above theorem gives us a subquadratic reduction:

Corollary 4.5.6. *For all non-trivial coefficients $\bar{\alpha}$ and t:*

$$\mathsf{LDT}_?(3, \bar{\alpha}, t) \leq_2 \mathsf{LDT}_3(3, \bar{\alpha}, t).$$

Now we can combine this result with the existing reductions from Section 4.2 to obtain a similar claim for 1-partite 3LDT:

Lemma 4.5.7. *For all non-trivial coefficients $\bar{\alpha}$ and t:*

$$\mathsf{LDT}_?(1, \bar{\alpha}, t) \leq_2 \mathsf{LDT}_3(1, \bar{\alpha}, t)$$

Proof. The reduction combines three existing reductions:

$$\mathsf{LDT}_?(1, \bar{\alpha}, t) \overset{\text{L 4.2.4}}{\leq_2} \mathsf{LDT}_?(3, \bar{\alpha}, t) \overset{\text{C 4.5.6}}{\leq_2} \mathsf{LDT}_3(3, \bar{\alpha}, t) \overset{\text{T 4.1.3}}{\leq_2} \mathsf{LDT}_3(1, \bar{\alpha}, t)$$

In the first the step we can use Lemma 4.2.4, as it does not depend on the values of the elements. □

Clearly, $\mathsf{LDT}_3(p, \bar{\alpha}, t) \leq_2 \mathsf{LDT}_?(p, \bar{\alpha}, t)$, so we obtain the main result of this section:

Theorem 4.1.7. *For every $U \geq n^3, p \in \{1,3\}$ and non-trivial coefficients $\bar{\alpha}, t$, $\mathsf{LDT}_?(p, \bar{\alpha}, t)$ over $[-U, U]$ is subquadratic-equivalent to $\mathsf{LDT}_3(p, \bar{\alpha}, t)$.*

Finally, we can obtain similar universe reductions for Convolution variants of LDT:

Theorem 4.1.8. *For every $U \geq n^3, p \in \{1,3\}$ and non-trivial coefficients $\bar{\alpha}, t$, $\mathsf{Conv3LDT}_?(p, \bar{\alpha}, t)$ over $[-U, U]$ is subquadratic-equivalent to $\mathsf{Conv3LDT}_2(p, \bar{\alpha}, t)$.*

Proof. It suffices to show that or all $p \in \{1,3\}$ and non-trivial coefficients $\bar{\alpha}$ and t:

$$\mathsf{Conv3LDT}_?(p, \bar{\alpha}, t) \leq_2 \mathsf{Conv3LDT}_2(p, \bar{\alpha}, t)$$

The cases of $p = 1$ and $p = 3$ are established by the following chain of inequalities that we justify in detail below:

$$\begin{aligned}
\mathsf{Conv3LDT}_?(1, \bar{\alpha}, t) &\overset{\text{T 4.3.3}}{\leq_2} \mathsf{Conv3LDT}_?(3, \bar{\alpha}, t) \overset{\text{L 4.3.2}}{\leq_2} \mathsf{Conv3LDT}_?(3, \bar{\alpha}, 0) \\
&\overset{\text{L 4.4.1}}{\leq_2} \mathsf{LDT}_?(3, \bar{\alpha}, 0) \overset{\text{C 4.5.6}}{\leq_2} \mathsf{LDT}_3(3, \bar{\alpha}, 0) \\
&\overset{\text{T 4.1.6}}{\equiv_2} \mathsf{Conv3LDT}_2(3, \bar{\alpha}, t) \overset{\text{T 4.1.6}}{\equiv_2} \mathsf{Conv3LDT}_2(1, \bar{\alpha}, t)
\end{aligned}$$

In the first inequality, we can use Theorem 4.3.3, because it only uses color-coding (Fact 4.2.5), which does not depend on the values of the elements and Lemma 4.3.1. The latter operates on the values which are only a slight, linear modification of the original ones, so the claim can be adjusted.

In the second inequality, we again adapt Lemma 4.3.2 to bigger values. Finally, in the third inequality we can use Lemma 4.4.1 in which we only apply a linear function to each of the elements, so it also applies for numbers from $[-U, U]$. The claims used in the following steps apply directly, with no further modifications. Hence the claim follows. □

Chapter 5

Slowing Down Top Trees for Better Worst-Case Compression

5.1 Preliminaries (following [BFG17; BGLW15])

In this section, we briefly restate the top tree construction algorithm of Bille et al. [BGLW15]. To construct trees that can be used to show the lower bound and present our modification of the original algorithm we need to work with exactly the same definitions. Consequently, the following description closely follows the condensed presentation from Bille et al. [BFG17] and can be omitted if the reader is already familiar with the approach.

Let T be a (rooted) tree on n nodes. Every node has a label and node v has $c(v)$ children $v_1, \ldots, v_{c(v)}$ ordered from left to right. Let $T(v)$ denote the subtree of v, including v and $N(T)$ denote the set of nodes from a tree T. We distinguish some subtrees of tree T as *clusters*. Every cluster has a top boundary node and possibly a bottom boundary node and there are two types of clusters.

First, for a node v and $1 \leq s \leq r \leq c(v)$, we define a cluster $T(v, s, r)$ induced by a node v and a contiguous range of subtrees of children of v starting from the s-th and ending at the r-th, that is: $N(T(v, s, r)) = \{v\} \cup \bigcup_{i \in [s,r]} N(T(v_i))$. Such a cluster has top boundary node v and no bottom boundary node.

Second, for a node v, $1 \leq s \leq r \leq c(v)$ and a node $u \neq v$ from $N(T(v, s, r))$ we define a cluster $T(v, s, r) \setminus T(u) \cup \{u\}$ to be the subtree induced by nodes from $N(T(v, s, r)) \setminus N(T(u)) \cup \{u\}$, which has edges from $T(v, s, r)$ excluding the edges from $T(u)$. Such a cluster has top boundary node v and bottom boundary node u.

If edge-disjoint clusters A and B have exactly one common boundary node and $C = A \cup B$ is a cluster, then A and B can be *merged* into C. Then one of the top boundary nodes of A and B becomes the top boundary node of C and there are various ways of choosing the bottom boundary node of C. See Figure 5.1 with all five possible ways of merging two clusters.

A *top tree* $\mathcal{T}$ of T is an ordered and labeled binary tree describing a hierarchical decomposition of T into clusters which satisfies the following properties:

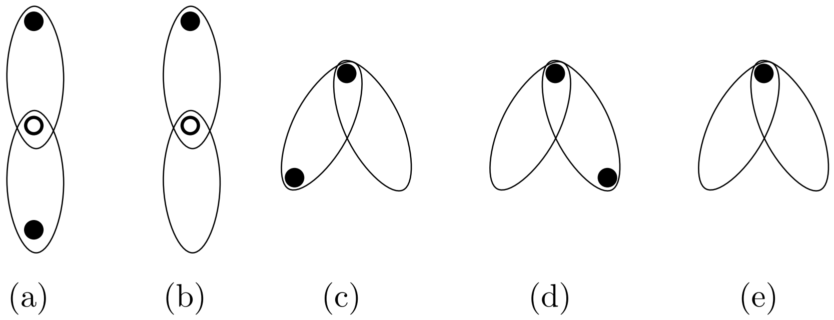


Figure 5.1: (cf. Figure 2 in [BGLW15]) all five ways of merging two clusters. Full circles denote boundary nodes of the resulting cluster and empty circle denotes the common boundary node of the merged clusters which is not boundary for the resulting cluster. Variants (a) and (b) appear during the vertical step and variants (c)-(e) during the horizontal step.

- The nodes of $\mathcal{T}$ correspond to the clusters of T.
- The root of $\mathcal{T}$ corresponds to the whole T.
- The leaves of $\mathcal{T}$ correspond to the edges of T. The label of each leaf is the pair of labels of the endpoints of its corresponding edge (u, v) in T. The two labels are ordered so that the label of the parent appears before the label of the child.
- Each internal node of $\mathcal{T}$ corresponds to the merged cluster of the clusters corresponding to its two children. The label of each internal node is the type of merge it represents (out of the five merging options). The children are ordered so that the left child is the child cluster visited first in the preorder traversal of T.

The top tree $\mathcal{T}$ is constructed bottom-up in iterations, starting with the edges of T as the leaves of $\mathcal{T}$. During the process, we maintain an auxiliary ordered tree $\widetilde{T}$, initially set to T. The edges of $\widetilde{T}$ correspond to the nodes of $\mathcal{T}$, which in turn correspond to the clusters of T. The internal nodes of $\widetilde{T}$ correspond to the boundary nodes of these clusters and the leaves of $\widetilde{T}$ correspond to a subset of leaves of T.

On a high level, the iterations are designed in such a way that each of them merges a constant fraction of edges of $\widetilde{T}$. This is proved in Lemma 1 of [BGLW15], and we describe a slightly stronger property in Lemma 5.3.1. This guarantees that the height of the resulting top tree is $\mathcal{O}(\log n)$. Every iteration consists of two steps:

Horizontal merges For each node $v \in \widetilde{T}$ with $k \geq 2$ children $v_1, \ldots, v_k$, for $i = 1$ to $\lfloor \frac{k}{2} \rfloor$, merge the edges (v, v_{2i-1}) and (v, v_{2i}) if v_{2i-1} or v_{2i} is a leaf. If k is odd and v_k is a leaf and both v_{k-2} and v_{k-1} are non-leaves then also merge (v, v_{k-1}) and (v, v_k).

Vertical merges For each maximal path $v_1, \ldots, v_p$ of nodes in $\widetilde{T}$ such that v_{i+1} is the parent of v_i and $v_2, \ldots, v_{p-1}$ have a single child: If p is even merge the following pairs of edges

$\{(v_1, v_2), (v_2, v_3)\}, \{(v_3, v_4), (v_4, v_5)\}, \dots, \{(v_{p-3}, v_{p-2}), (v_{p-2}, v_{p-1})\}$. If p is odd merge the following pairs of edges $\{(v_1, v_2), (v_2, v_3)\}, \{(v_3, v_4), (v_4, v_5)\}, \dots, \{(v_{p-4}, v_{p-3}), (v_{p-3}, v_{p-2})\}$, and if (v_{p-1}, v_p) was not merged in the previous step (horizontal merge) then also merge $\{(v_{p-2}, v_{p-1}), (v_{p-1}, v_p)\}$.

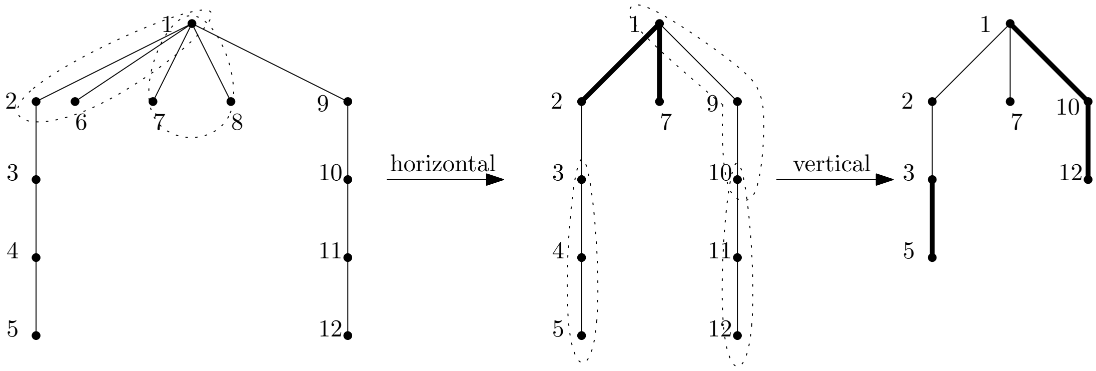


Figure 5.2: Tree $\widetilde{T}$ after two steps of a single iteration. Dotted lines denote the merged edges (clusters) and thick edges denote results of the merge. Note that edge $(1, 2)$ does not participate in the vertical merge as it was obtained as a result of the horizontal merge in this iteration.

See an example of a single iteration in Figure 5.2. Finally, the compressed representation of T is the so-called top DAG $\mathcal{TD}$, which is the minimal DAG representation of $\mathcal{T}$ obtained by identifying identical subtrees of $\mathcal{T}$. As every iteration shrinks $\widetilde{T}$ by a constant factor, $\mathcal{T}$ can be computed in $\mathcal{O}(n)$ time, and then $\mathcal{TD}$ can be computed in $\mathcal{O}(|\mathcal{T}|)$ time [DST80]. Thus, the entire compression takes $\mathcal{O}(n)$ time. See Algorithm 2.

Algorithm 2 Original top tree construction algorithm of Bille et al.[BGLW15] for a tree T.

```
$\widetilde{T} := T$
initialize leaves of $\mathcal{T}$ with edges of $T$
**for** $t = 1, \dots, \Theta(\log n)$, as long as $\widetilde{T}$ is not a single edge **do**
    perform horizontal merges and update $\widetilde{T}$ and $\mathcal{T}$
    perform vertical merges and update $\widetilde{T}$ and $\mathcal{T}$
construct DAG $\mathcal{TD}$ of $\mathcal{T}$                    ▷ $\mathcal{TD}$ is the top DAG of $T$
```

5.2 A lower bound for the approach of Bille et al.

In this section, we prove Theorem 2.3.1 and show that the $\mathcal{O}(\frac{n}{\log_\sigma n} \log\log_\sigma n)$ bound from [HR15] on the number of distinct clusters created by the algorithm described in Section 5.1 is tight. We first consider labeled trees for which $|\Sigma| > 1$ and $\sigma = |\Sigma|$. Then we show how to modify our construction and apply it to unlabeled trees.

For every $k \in \mathbb{N}$, we set $t = 8^k$ and will construct a tree T_k with $n = \Theta(\sigma^t)$ nodes for which the corresponding top DAG is of size $\Theta(\frac{n}{\log_\sigma n} \log\log_\sigma n)$. In the beginning, we describe a gadget G_k that is the main building block of T_k. It consists of $\Theta(t)$ nodes: a path of $t + 1$ nodes and $2^k - 1 < t^{1/3}$ full ternary trees of size $\mathcal{O}(t^\varepsilon)$ connected to the top node of the path (root), where $\varepsilon + \frac{1}{3} < 1$, see Figure 5.3. The main intuition behind the construction is that the full ternary trees are significantly smaller than the path, but they need the same number of iterations to get compressed.

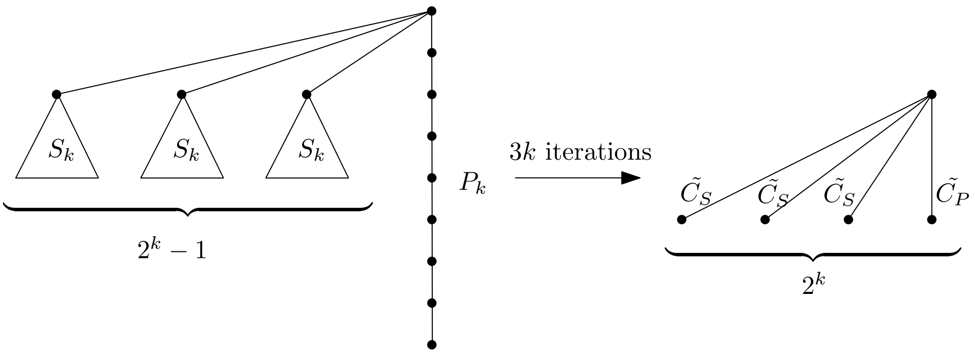


Figure 5.3: Gadget G_k consists of $2^k - 1 < t^{1/3}$ trees S_k and one path P_k. After $3k$ iterations it gets compressed to a tree with 2^k nodes connected to the root.

More precisely, let P_k be the path of length $t = 8^k$, with $t+1$ nodes. Clearly, for $k \geq 1$ after 3 iterations P_k gets compressed to P_{k-1}, and so after $3k$ iterations becomes P_0 of length 1, a single cluster that we call $\tilde{C}_P$. Similarly, let S_k be the full ternary tree of height k with 3^k leaves, so it has $\frac{3^{k+1}-1}{2} = \mathcal{O}(3^k) = \mathcal{O}(t^{0.53})$ nodes in total. Observe that for $k > 1$, after 3 iterations S_k becomes S_{k-1}, S_1 gets compressed to a single cluster in 2 iterations so after $3k-1$ iterations S_k becomes a single cluster, we call it C_S. Finally, in the next iteration it gets merged with the edge connecting the roots of S_k and G_k, resulting in a single cluster attached to the root of G_k, we call it $\tilde{C}_S$. To sum up, the gadget G_k consists of a path P_k of $t+1$ nodes and $2^k - 1 < t^{0.34}$ trees of size $\mathcal{O}(t^{0.53})$ each, so in total $\Theta(t)$ nodes. After $3k$ iterations G_k consists of $2^k - 1$ clusters $\tilde{C}_S$ corresponding to S_k together with the edge between the roots of S_k and G_k, and one cluster $\tilde{C}_P$ corresponding to P_k, as shown in Figure 5.3. In each of the subsequent k iterations, the remaining 2^k clusters are merged in pairs.

Recall that the final top DAG contains a node for every distinct subtree of the top tree, and every node of the top tree corresponds to a cluster obtained during the compression process. Now we create many almost identical gadgets connected to the common root. In order to ensure that they are distinct, we assign labels from Σ to nodes on paths P_k so that no two paths are equal. Formally, the tree T_k consists of $r = \sigma^t/t$ gadgets connected to the common root as in Figure 5.4. The i-th gadget $G_k^{(i)}$ is a copy of G_k with the labels of the bottom t nodes of $P_k^{(i)}$ chosen as to spell out the i-th (in the lexicographical order) word of length t over Σ. For all the remaining nodes it is enough to choose the same label, e.g. the smallest in Σ. Clearly $\sigma^t > r = \sigma^t/t$, so there are more possible words of length t than the number of gadgets that we create. Note that in total the tree T_k has $n = 1 + r \cdot |G_k| = \sigma^t/t \cdot \Theta(t) = \Theta(\sigma^t)$ nodes.

After the first $3k$ iterations, every path $P_k^{(i)}$ gets compressed to a cluster $\tilde{C}_P^{(i)}$, as presented in Figure 5.3. Note that clusters $\tilde{C}_P^{(i)}$ and $\tilde{C}_P^{(j)}$ correspond to distinct subtrees of the top tree $\mathcal{T}$ for all $0 \leq i \neq j < r$. Consequently, so do the clusters obtained from them in each of the subsequent k iterations that apply only horizontal merge. Thus all the $r \cdot k$ clusters correspond

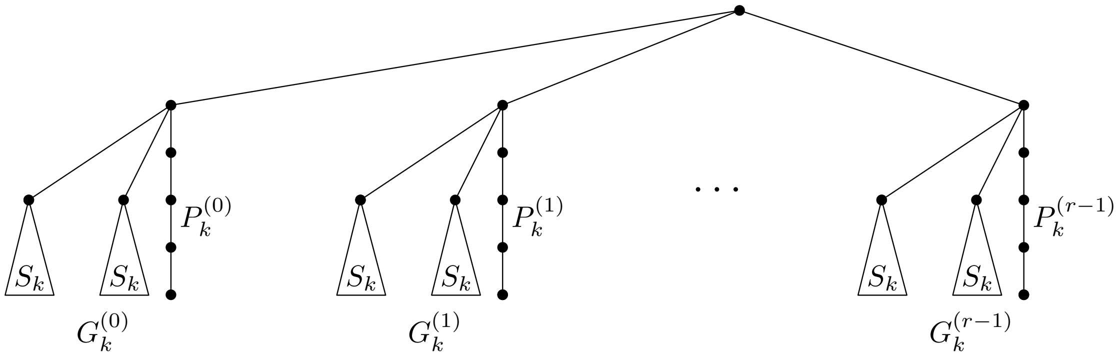


Figure 5.4: T_k consists of $r = \sigma^t/t$ gadgets $G_k^{(i)}$, where the i-th of them contains a unique path $P_k^{(i)}$.

to distinct subtrees of $\mathcal{T}$, which gives us the following lower bound on the size of the top DAG: $\Omega(r \cdot k) = \Omega(\sigma^t/t \cdot \log t) = \Omega(n/\log_\sigma n \cdot \log\log_\sigma n)$, as $t = 8^k = \Theta(\log_\sigma n)$. This concludes the proof of Theorem 2.3.1 for labeled trees with non-unary alphabet.

Unlabeled trees Now we modify the above construction so that it works for unary alphabets. Recall that for every $k \in \mathbb{N}$ we set $t = 8^k$ and $\sigma = \max\{2, |\Sigma|\} = 2$. We cannot use the earlier approach directly, as we cannot distinguish the gadgets $G_k^{(i)}$ by modifying labels on the path $P_k^{(i)}$. To address this issue we replace every path $P_k^{(i)}$ with the tree $B_k^{(i)}$ that encodes binary representation of $0 \le i < 2^t$ in the following way. $B_k^{(i)}$ consists of t nodes connected to the root. For every $0 \le j < t$, under the j-th child of the root there is connected a path of 2 edges if j-th bit of i equals 0, or two edges otherwise. Then after 2 iterations the tree is compressed to t clusters connected to the root, each of them is either $\tilde{C}_0$ or $\tilde{C}_1$. See Figure 5.5 for the example and description of clusters $C_0, C_1, \tilde{C}_0$ and $\tilde{C}_1$. Note that after the next $\log t = 3k$ iterations the tree gets compressed to a single cluster $C_B^{(i)}$.

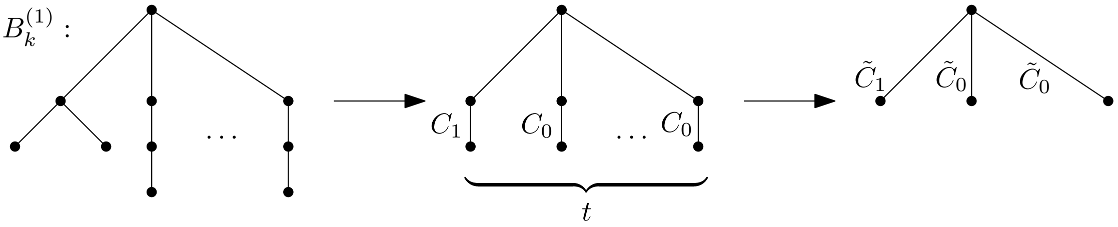


Figure 5.5: $B_k^{(i)}$ encodes binary representation of $0 \le i < 2^t$. After 2 steps it gets compressed to a tree of t clusters connected to the root. In the example of $B_k^{(1)}$ we have $k = 1, t = 8$ and $i = 1 \cdot 2^0$, so the first child of the root has two edges attached to it and then there $t - 1$ paths of length 2 connected to the root.

Now we describe the tree T'_k for which the size of top DAG is $\Omega(\frac{n}{\log_\sigma n} \log\log_\sigma n)$. T'_k consists of $r = 2^t/t$ gadgets $G_k'^{(i)}$ where the i-th gadget $G_k'^{(i)}$ consists of $2^k - 1$ trees S_{k+1} defined for the case of labeled trees and tree $B_k^{(i)}$ connected to the root, see Figure 5.6. After $3k + 3$ steps it gets compressed to a tree with 2^k clusters connected to the root, where the last one is $\tilde{C}_B^{(i)}$, as in the labeled case presented in Figure 5.3. Similarly as in the analysis of the labeled case, each of the r gadgets introduces at least $\log 2^k = k$ new nodes of the top DAG, as all clusters containing $\tilde{C}_B^{(i)}$ are distinct, so in total the top DAG has size $\Omega(r \cdot k)$.

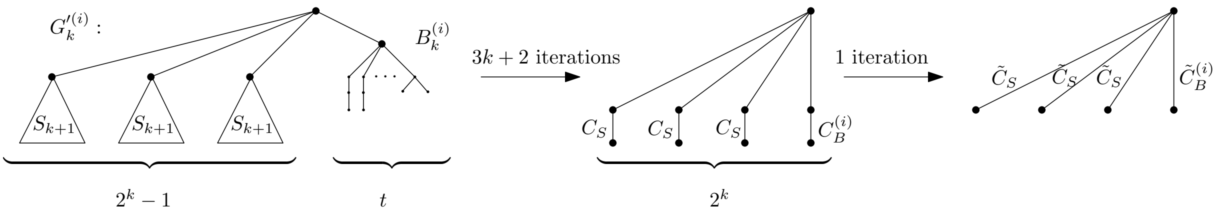


Figure 5.6: The modified gadget $G_k'^{(i)}$ for unlabeled trees. After $3k+3$ iterations it gets compressed to a tree with the same properties as in the labeled case.

Each of the trees $B_k^{(i)}$ consists of $3t + 1 = \Theta(t)$ nodes. Gadget $G_k^{(i)}$ has $2^k - 1 < t^{0.34}$ trees S_{k+1} of size $\mathcal{O}(3^k) = \mathcal{O}(t^{0.53})$ so in total, including tree $B_k^{(i)}$, $G_k^{(i)}$ has $\Theta(t)$ nodes. Then the tree T_k' has $n = 1 + r \cdot \Theta(t) = \Theta(\frac{2^t}{t} \cdot t) = \Theta(2^t)$ nodes, so $t = \Theta(\log n)$. By the above analysis, its top DAG has size $\Omega(r \cdot k) = \Omega(\frac{2^t}{t} \log_8 t) = \Omega(\frac{n}{t} \log t) = \Omega(\frac{n}{\log_\sigma n} \log \log_\sigma n)$ which concludes the proof of Theorem 2.3.1.

5.3 An optimal tree compression algorithm

Let α be a constant greater than 1 and consider the following modification of Algorithm 2 [BGLW15]. Our algorithm works in the same way for both labeled and unlabeled trees. Intuitively, we would like to proceed exactly as the original algorithm, except that in the t-th iteration we do not perform a merge if one of the participating clusters is of size larger than α^t. However, this would require a slight modification of the original charging argument showing that that after every iteration the tree $\widetilde{T}$ shrinks by a constant factor. To avoid adapting the whole proof of [BGLW15] to our new approach, we proceed slightly differently. In each iteration we first generate and list all the merges that would have been performed in both steps of a single iteration of the original Algorithm 2. Then we apply only the merges in which both clusters have at most α^t nodes. See Algorithm 3.

Algorithm 3 A modified top tree construction algorithm of Bille et al. [BGLW15] for a tree T.

```
$\widetilde{T} := T$
initialize leaves of $\mathcal{T}$ with edges of $T$
**for** $t = 1, \ldots, \Theta(\log n)$, as long as $\widetilde{T}$ is not a single cluster **do**
    list all the merges that would have been made by one iteration of Algorithm 2
    filter out the merges that use a cluster of size bigger than $\alpha^t$
    modify $\widetilde{T}$ and $\mathcal{T}$ by applying the remaining merges
construct DAG $\mathcal{TD}$ of $\mathcal{T}$                    ▷ $\mathcal{TD}$ is the top DAG of $T$
```

We run the algorithm until the tree $\widetilde{T}$ becomes a single cluster. Clearly, there are $\mathcal{O}(\log n)$ iterations, because after $\log_\alpha n$ iterations the algorithm is no longer constrained and can behave not worse than the original one. Thus the depth of the obtained DAG is $\mathcal{O}(\log n)$ as before. Till the end of this section we prove that the obtained DAG has $\mathcal{O}(\frac{n}{\log_\sigma n})$ nodes by showing different properties of the tree $\widetilde{T}$ of clusters. In the following we assume that $\alpha = 10/9$, but do not substitute it to avoid clutter. First we show that even if there are some clusters that cannot be merged in one step, the tree still shrinks by roughly a constant factor.

Lemma 5.3.1. *Suppose that there are $m = p+q$ clusters in $\widetilde{T}$ after $t-1$ iterations of Algorithm 3, where q is the number of clusters of size larger than α^t. Then, after t iterations there are at most $7/8m + q$ clusters.*

Proof. The proof is a generalization of Lemma 1 from [BGLW15]. Clearly $m > 1$, otherwise Algorithm 3 would terminate after $(t-1)$-th iteration. There are $m + 1 \geq 3$ nodes in $\widetilde{T}$, so at least $m/2 + 1$ of them have less than 2 children in $\widetilde{T}$. Indeed, otherwise there would be more than $m + 1 - (\frac{m}{2} + 1) = \frac{m}{2}$ nodes with at least 2 children so there would be more than m edges in a tree of $m + 1$ nodes. Next, as the root can have only one child, there are at least $\frac{m}{2}$ nodes that have a parent and at most one child. Let M be the set of at least $\frac{m}{2}$ edges connecting these nodes with their parent.

Fact 5.3.2 (Lemma 1 in [BGLW15])**.** *Let $\widetilde{T}$ have at least 3 nodes, $p(v)$ be the parent of v in $\widetilde{T}$ and M be the set of edges $(v_i, p(v_i))$ in $\widetilde{T}$ such that v_i has less than 2 children in $\widetilde{T}$. During a single iteration of Algorithm 2, at least $|M|/2$ edges of M are merged.*

Proof. For completeness we restate the original proof from [BGLW15]. We will provide an injective mapping from not merged to merged edges of M. Consider an edge $(v, p) \in M$ that was not merged. There are three possibilities for the node v:

1. v is a leaf and has at least one sibling.

   Observe that v has a left sibling as otherwise (v, p) would have been merged with the right sibling of v. Let w be the left sibling of v. As (v, p) was not merged with (w, p) during the horizontal step, (w, p) must have been merged with (u, p), where u is the left sibling of w. Let $x = w$ if w is a leaf or $x = u$ otherwise. Then we map (v, p) to (x, p). Note that in this case we mapped (v, p) to the edge (x, p) where x is a leaf as either w or u is a leaf.

2. v is a leaf and has no siblings.

   $\widetilde{T}$ has at least 3 nodes, so there exists parent r of p. As (v, p) did not take part in the vertical step, (p, r) took part in the horizontal merge with (q, r), where q is a sibling of p. Then we map (v, p) to (p, r). Note that in this case we mapped (v, p) to the edge (p, r) where p is a not a leaf and has a sibling.

3. v has exactly one child c.

   The only possibility that (v, p) was not merged is that (c, v) was merged vertically with (d, c) where d is a child of c. Then we map (v, p) to the edge (c, v) where c is not a leaf but has no siblings.

Observe that the above mapping is injective, that is no two not merged edges are mapped to the same merged edge. Indeed, in all the three cases we can describe an inverse mapping and every edge can be mapped to in at most one of the cases. To sum up, from the above charging argument we have that there are at least as many merged edges in M as not merged edges, so the claim follows. □

From the above property we obtain that at least half of the edges of M would be merged in a single iteration of the original algorithm. In the worst case, they would be all merged in pairs, so there would be at least $|M|/4 \geq m/8$ merges performed. Now, q clusters (edges) are too large to participate in a merge and in the worst case each of them would have participated in a different merge of the original algorithm. Thus, Algorithm 3 performs at least $m/8 - q$ merges, so after a single iteration there are at most $m - (m/8 - q) = 7/8m + q$ clusters. □

Fact 5.3.3. *After t iterations of Algorithm 3 all clusters in $\widetilde{T}$ have at most $2\alpha^t$ nodes.*

Proof. By induction on t. Before the first iteration, for $t = 0$ all clusters of $\widetilde{T}$ have two nodes as they correspond to edges of T. In t-th iteration the largest cluster that can take part in a merge has at most α^t nodes. Two merged clusters always share a node so the size of the resulting cluster is upper bounded by $2\alpha^t - 1$. The largest cluster can be formed either in t-th or earlier iteration, so the claim follows. □

Fact 5.3.4. *Every cluster on c nodes created by Algorithm 3 is represented by a subtree of $\mathcal{T}$ on at most $2c - 3$ nodes.*

Proof. Induction by the size of cluster. In the beginning the clusters have 2 nodes and are represented by a leaf of $\mathcal{T}$. Consider a merge of two clusters on c_1 and c_2 nodes which are represented by subtrees of $\mathcal{T}$ on t_1 and t_2 nodes respectively. The merged cluster has $c' = c_1 + c_2 - 1$ nodes (the merged clusters share a node) and is represented by a subtree of $\mathcal{T}$ with $t' = t_1 + t_2 + 1$ nodes (we additionally have a root connected to the two subtrees). By induction we have $t_i \leq 2c_i - 3$ for $i \in \{1, 2\}$ so $t' \leq 2c_1 - 3 + 2c_2 - 3 + 1 = 2c' - 3$ and the claim follows. □

Lemma 5.3.5. *After t iterations of Algorithm 3 there are $\mathcal{O}(n/\alpha^t)$ clusters in $\widetilde{T}$.*

Proof. We prove by induction on t that after t iterations, $\widetilde{T}$ contains at most cn/α^t clusters, where $c = 103$. The case $t = 0$ is immediate. For $t > 0$ consider tree $\widetilde{T}$ after $t - 1$ iterations of the algorithm. Let p be the number of clusters having at most α^t nodes (call them small) and q having size larger than α^t (call them big). From the induction hypothesis, $p + q \leq cn/\alpha^{t-1}$, so in particular $p \leq cn/\alpha^{t-1}$.

Consider two arbitrary distinct clusters of $\widetilde{T}$. Observe that they are almost disjoint, that is they can share only one node that is boundary for both the clusters and every non-boundary node of a cluster of $\widetilde{T}$ cannot belong to any other cluster. Then $q \leq \frac{n}{\alpha^t - 2}$ as the big clusters have at least $\alpha^t - 2$ non-boundary nodes, the sets of non-boundary nodes for all clusters are pairwise disjoint and their total size is upper bounded by n.

Now we show that the total number of clusters after t iterations is at most cn/α^t. Recall that $\alpha = 10/9$. There are two cases to consider:

- $q \leq \frac{1}{100}p$: We apply Lemma 5.3.1 and conclude that the total number of clusters after the t-th iteration is at most $7/8(p+q)+q = 7/8p + 15/8q < 9/10p \leq cn/\alpha^t$.
- $p < 100q$: In the worst case no pair of clusters was merged and the total number of clusters after the t-th iteration is $p+q < 101q \leq 101\frac{n}{\alpha^{t-2}} \leq cn/\alpha^t$ for $\alpha^t \geq 103$. For $\alpha^t < 103$, the claim also holds as $p+q < n \leq cn/\alpha^t$. □

Finally we are ready to prove the main theorem of this section, that the top DAG returned by Algorithm 3 has optimal size.

Theorem 5.3.6. *Let T be a tree on n nodes labeled from an alphabet of size σ. Then the size of the corresponding top DAG obtained by Algorithm 3 with $\alpha = 10/9$ is $\mathcal{O}(\frac{n}{\log_\sigma n})$.*

Proof. The proof follows the reasoning from Section 3.1 in [BGLW15]. Clusters are represented with binary trees labeled either with pairs of labels from the original alphabet or one of the 5 labels representing the type of merge, so in total there are $|\Sigma|^2+5 \leq \sigma^2+5$ possible labels of nodes in $\mathcal{T}$. From the properties of Catalan numbers, it follows that the number of different binary trees of size x is bounded by 4^x. Thus there are at most $\sum_{i=1}^{x}(4(\sigma^2+5))^i \leq \sum_{i=1}^{x}(12\sigma^2)^i \leq (12\sigma^2)^{x+1}$ distinct labeled trees of size at most x, since $\sigma \geq 2$. Even if some of them appear many times in $\widetilde{T}$, they will be represented only once in $\mathcal{TD}$.

Let $t = \lfloor \log_\alpha \frac{3}{16}\log_{12\sigma^2} n \rfloor$ and consider the situation after t iterations of Algorithm 3. By combining Facts 5.3.3 and 5.3.4, every cluster obtained so far is represented by a tree on at most $4\alpha^t - 3 \leq 4\alpha^t - 1$ nodes. By the discussion in the previous paragraph, there are at most $(12\sigma^2)^{4\alpha^t} \leq (12\sigma^2)^{3/4\log_{12\sigma^2} n} = n^{3/4}$ distinct trees describing the obtained clusters. As identical subtrees of $\mathcal{T}$ are identified by the same node in the top DAG, all clusters processed during the first t iterations of the algorithm are represented by at most $n^{3/4}$ nodes in $\mathcal{TD}$.

Next, by Lemma 5.3.5 there are at most $\mathcal{O}(n/\alpha^t)$ clusters in $\widetilde{T}$ and in the subsequent merges we can obtain at most $\mathcal{O}(n/\alpha^t)$ new clusters, each of which corresponds to a new node in $\mathcal{TD}$. Finally, the size of the DAG obtained by Algorithm 3 on a tree T of size n is upper bounded by $n^{3/4} + \mathcal{O}(n/\alpha^t) = \mathcal{O}(n/\log_{12\sigma^2} n)$, which is $\mathcal{O}(n/\log_\sigma n)$ as $\log_{12\sigma^2} n \geq \frac{1}{6}\log_\sigma n$ for $\sigma \geq 2$. □

Chapter 6

Problems Equivalent to Counting 4-Cycles in Graphs

6.1 Counting 4-Cycles in Different Graphs

Whenever we talk about counting 4-cycles in a graph we mean simple cycles (with all nodes distinct) of length 4, but not necessarily induced. For counting 4-cycles self-loops and isolated nodes are irrelevant, but there might be multiple edges, and then we count the cycle multiple times, the product of the multiplicities of the edges forming the cycle. Following the naming convention from [LWW18], we define a *4-circle-layered graph* to be a 4-partite directed graph with four disjoint groups of nodes $V_0, \ldots, V_3$ such that every edge in the graph leads to the next group modulo 4, that is $E \subseteq \sum_{0 \leq i \leq 3} V_i \times V_{(i+1) \bmod 4}$. *Multigraph* is a triple $(V, E, \textsc{mult})$, where E is a set of m edges and the function $\textsc{mult} : E \to \{1, \ldots, U\}$ denotes multiplicity of an edge. For simple graphs it holds that $\textsc{mult}(e) = 1$ for all edges $e \in E$ and the function is omitted. To simplify notation we write $\textsc{mult}(u, v)$ instead of $\textsc{mult}((u, v))$.

In this section we show reductions between problems of counting 4-cycles in different graphs. The graphs are always simple but they differ in the structure: they are either arbitrary, bipartite or 4-cycle-layered and either weighted or unweighted. As a warm-up we show how to reduce counting 4-cycles in simple graphs to counting 4-cycles in bipartite graphs.

Lemma 6.1.1. *Counting 4-cycles in simple graphs can be reduced in linear time to counting 4-cycles in simple bipartite graphs.*

Proof. For a given simple graph $G = (V, E)$ we construct a bipartite graph $G' = (V_1 \cup V_2, E')$ in the following way. For every $v \in V$ we create two nodes $v_1 \in V_1$ and $v_2 \in V_2$ and for every edge $\{u, v\} \in E$ we create two edges $\{u_1, v_2\}$ and $\{u_2, v_1\}$ in G'. Then every cycle (a, b, c, d) in G corresponds to two cycles (a_1, b_2, c_1, d_2) and (a_2, b_1, c_2, d_1) in G' and there are no other 4-cycles in G'. □

In the following theorem we show that for any two types of graphs among: undirected graph, undirected multigraph, directed multigraph, 4-circle-layered multigraph, 4-circle-layered graph, we can reduce counting 4-cycles in a graph of one type with m edges with multiplicities bounded by U to $\mathcal{O}(\log^4 U)$ instances of counting 4-cycles in graphs of the other type with $\mathcal{O}(m)$ edges.

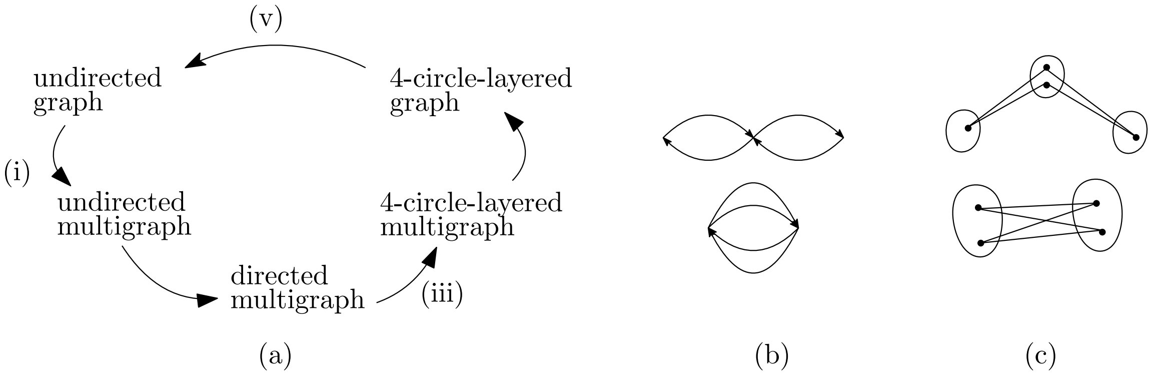


Figure 6.1: (a) Sequence of reductions showing equivalence between counting 4-cycles in different types of graphs. (b) Non-simple cycles from G' to subtract in reduction (iii). (c) Cycles to subtract (top) and add (bottom) in reduction (v).

Theorem 2.2.6. *Counting 4-cycles in undirected multigraphs on m edges with multiplicities bounded by U can be reduced to $\mathcal{O}(\log^4 U)$ instances of counting 4-cycles in undirected simple graphs on $\mathcal{O}(m)$ edges.*

Proof. We consider five types of graphs: first undirected graphs, then undirected multigraphs, directed multigraphs, 4-circle layered multigraphs and finally 4-circle-layered graphs. For each of them we show that counting 4-cycles in graphs of this type can be reduced in $\mathcal{O}(m)$ time to a number of instances of counting 4-cycles in the graphs of the next type, as presented in Figure 6.1(a). We describe each of the reductions separately:

(i) **undirected graph → undirected multigraph.** Simple graph is a special case of multigraph but with edge multiplicities bounded by $U = 1$.

(ii) **undirected multigraph → directed multigraph.** Given an undirected multigraph G we construct a directed multigraph G' replacing every undirected edge with two directed edges with the same multiplicity as the original edge. Then the number of 4-cycles in G' is twice the number of 4-cycles in G, as every cycle can be traversed in both directions and we count only simple 4-cycles, that is we cannot cannot use one node more than once on a single cycle. Then we have: $\#_{C_4}(G) = \frac{1}{2}\#_{C_4}(G')$.

(iii) **directed multigraph → 4-circle-layered multigraph.** Given a directed multigraph $G = (V, E)$ we construct a 4-circle-layered graph G' by copying nodes of G four times and adding edges between corresponding nodes from two consecutive groups preserving the multiplicities. More precisely, for every node $v \in V$ and $i \in \{0, 1, 2, 3\}$ we create node $v'_i \in V'_i$, the copy of v in the i-th group of nodes of G'. For every directed edge $(u, v) \in E$ we add the edge $(u'_i, v'_{(i+1) \bmod 4})$ to G' for all $0 \leq i \leq 3$, setting $\textsc{mult}_{G'}(u'_i, v'_{(i+1) \bmod 4}) := \textsc{mult}_G(u, v)$. Then the number of 4-cycles in G' is 4 times the number of 4-cycles in G

plus some additional cycles which do not correspond to simple cycles in G. Indeed, all the additional 4-cycles in G' correspond to non-simple (on 2 or 3 distinct nodes) 4-cycles in G, which are shown in Figure 6.1(b) and can be counted in linear time. Formally, let $N(u) = \{v \in V : (u,v) \in E \wedge (v,u) \in E\}$ be the set of neighbors of a node u connected with u in both directions and we define the values $a(u), b(u)$ and $c(u)$ as follows:

$$a(u) = \sum_{v \in N(u)} \text{MULT}_G(u,v)\text{MULT}_G(v,u) \qquad \text{counts paths } u \to v \to u$$

$$b(u) = \sum_{v \in N(u)} (\text{MULT}_G(u,v)\text{MULT}_G(v,u))^2 \qquad \text{counts paths } u \to v \to u \to v \to u$$

$$c(u) = (a(u))^2 - b(u) \qquad \text{counts paths } u \to v \to u \to w \to u, (v \neq w)$$

Sets $N(u)$ can be obtained by bucket sorting the adjacency lists in linear time and in the same time we compute values $a(u), b(u)$ and $c(u)$. We use values $b(u)$ and $c(u)$ to subtract the additional cycles, as presented in Figure 6.1(b): $\#_{C_4}(G) = \frac{1}{4}\left(\#_{C_4}(G') - \sum_{u \in V}(2c(u) + b(u))\right)$.

(iv) **4-circle-layered multigraph → 4-circle-layered graph.** We show that counting 4-cycles in a 4-circle-layered multigraph with edge multiplicities bounded by U can be reduced to $\log^4 U$ instances of counting 4-cycles in 4-circle-layered simple graphs of the same size as the original graph. Informally, we split every edge of the graph into edges with multiplicities being powers of two and iterate over all possible combinations of powers of two forming the cycle.

More precisely, we iterate over all quadruples $(p_0, p_1, p_2, p_3) \in \{0, \ldots, \lfloor \log U \rfloor\}^4$ and for each of them create a simple, unweighted 4-circle-layered graph on the same set of nodes as the original graph and a subset of its edges. For all $0 \leq i \leq 3$ we keep only the edges between groups V_i and $V_{(i+1) \bmod 4}$ such that their multiplicity contains 2^{p_i} in its binary representation. Then we count 4-cycles in the obtained graph and multiply it by $2^{\sum_i p_i}$. Finally, the total number of 4-cycles in the original multigraph is the sum of results obtained for all the considered quadruples.

(v) **4-circle-layered → undirected.** Given a 4-circle-layered graph G with groups of nodes V_i we create an undirected graph G' by undirecting all edges from G. Then we can no longer ensure that the 4-cycles pass through 4 different groups of nodes, so we need to subtract 4-cycles fully contained in three groups of nodes and add 4-cycles fully contained in two groups, as shown in Figure 6.1(c). The number of such cycles can be obtained by counting 4-cycles in the graph G' restricted only to the particular groups of nodes. Formally, let $G'[W]$ be the subgraph of G' restricted to the nodes from W and edges between them. Then we have: $\#_{C_4}(G) = \#_{C_4}(G') - \sum_{0 \leq i \leq 3} \#_{C_4}(G'[V_i \cup V_{i+1} \cup V_{i+2}]) + \sum \#_{C_4}(G'[V_i \cup V_{i+1}])$ where $0 \leq i \leq 3$ and the indices $i+1$ and $i+2$ are taken modulo 4. □

6.2 Counting Patterns

6.2.1 Notation and Definitions

Permutation π of length n is a bijective mapping $\pi : [n] \to [n]$, where $[n] = \{1, \dots, n\}$ and a k-pattern is a permutation of length k. A permutation π contains a k-pattern σ if there exist indices $1 \leq i_1 < i_2 < \ldots < i_k \leq n$ such that $\sigma(j) < \sigma(j')$ iff $\pi(i_j) < \pi(i_{j'})$ for $j, j' \in [k]$. A sequence of k increasing indices with the above property is called an *occurrence* of σ in π. For example, in permutation $524\underline{6}\underline{1}7\underline{3}$ the underlined positions $4, 5$ and 7 with elements respectively 6, 1 and 3 form an occurrence of pattern 312. By counting a k-pattern in a permutation we mean counting occurrences of the pattern. We generalize the problem of counting occurrences of a pattern by adding weights to the elements. Formally, we introduce k weight functions $\lambda_j : [n] \to U$ for $j \in [k]$ and define $\#^{\lambda}_{\sigma}(\pi)$, the number of multi-weighted occurrences of a k-pattern σ in permutation π, with weight functions $\lambda_{[k]}$ to be:

$$\#^{\lambda}_{\sigma}(\pi) = \sum_{\substack{\text{sequences } i_1 < i_2 < \dots < i_k \\ \text{forming an occurrence of } \sigma \text{ in } \pi}} \prod_{j=1}^{k} \lambda_j(i_j)$$

In some cases we work with the special case when all the weights for a particular element x are equal, that is for all $x \in [n]$ we have $\lambda_j(x) = \lambda(x)$ for all $j \in [k]$. Then $\lambda : [n] \to U$ is the weight function. In order to distinguish the naming between the cases, we say that we count a pattern, weighted pattern or multi-weighted pattern when, respectively, there is either no weight function, a single weight function or k weight functions $\lambda_{[k]}$. Clearly, the number of occurrences of a pattern σ equals to the number of weighted occurrences of this pattern where $\lambda(i) = 1$ for all $i \in [n]$ and multi-weighted case is the generalization of the weighted one. When the weight function(s) λ is clear from the context we simply write $\#_{\sigma}(\pi)$ instead of $\#^{\lambda}_{\sigma}(\pi)$.

Shapes. We represent permutation π as a set of points in the plane: $S_\pi = \{(i, \pi(i)) : i \in [n]\}$ and we interchangeably use points and their corresponding elements from the permutation. For instance, four points $\{(i_j, \pi(i_j)) : j \in [k]\}$ form an occurrence of k-pattern σ iff positions $i_1 < \ldots < i_k$ form an occurrence of σ in π. The points in S_π are also accompanied with the weight function(s) λ. We say that a horizontal line divides a plane into top and bottom part and a vertical line divides into left and right part. Division of a plane with both a horizontal and a vertical line splits the points from S_π into four *regions* and we abbreviately denote each of them by capital letters denoting horizontal and vertical location of the region: TL, TR, BL or BR, where e.g. TL is the top-left region. Slightly abusing the notation, by a region we mean either the region or the set of points from S_π that belong to the region, with the appropriate order between them. Translating it to the correspondence between the elements of π and S_π, notice that the division of the plane with horizontal line $y = h$ and vertical line $x = v$ partitions elements from π into four groups, for instance $(i, \pi(i)) \in$ TL iff $i < v \wedge \pi(i) > h$. We will only consider such divisions of the plane that the dividing lines never pass through a point from S_π, by choosing $h, v \notin [n]$.

Given a division of the plane, we say that an occurrence of k-pattern σ forms *shape* $\frac{a|b}{c|d}$ if $a+b+c+d=k$ and among the k points there are respectively a, b, c and d points in top-left, top-right, bottom-left and bottom-right region of the plane. By counting a particular shape for a division, denoted $\#^{\lambda}_{\sigma_{\frac{a|b}{c|d}}}\left(\frac{TL|TR}{BL|BR}\right)$, we mean counting sequences of $a+b+c+d$ points forming the shape with appropriate number of points in each of the regions, taking into account the weight function(s) λ. Note that one pattern may form multiple shapes, e.g. 4123 may form $\frac{1|1}{0|2}, \frac{1|2}{0|1}, \frac{2|0}{2|0}$ or $\frac{1|1}{1|1}$, depending on the location of points with respect to the position of the dividing lines. However, some shapes cannot be formed by all patterns, no matter how we divide the plane, e.g. $\frac{1|1}{1|1}$ can be formed by 2314, but not by 2134. On the other hand, $\frac{0|2}{2|0}$ cannot be formed by 2314 but can be formed by 2134. As the shape $\frac{1|1}{1|1}$ will be central in our considerations, to simplify notation we write $\#_{\sigma_4}$ as the abbreviation of $\#_{\sigma_{\frac{1|1}{1|1}}}$ and we call the occurrences of pattern forming the shape $\frac{1|1}{1|1}$ *4-partite.*

Among all the possible shapes of a 4-pattern, we distinguish the following shapes that we call canonical: $\frac{4|0}{0|0}, \frac{3|0}{0|1}, \frac{3|0}{1|0}, \frac{2|0}{0|2}, \frac{2|0}{2|0}, \frac{1|1}{0|2}, \frac{1|2}{0|1}, \frac{1|1}{1|1}$. All other possible shapes on 4 elements are obtained by a rotation or symmetrical reflection of a canonical shape (the latter is important e.g. for the shape $\frac{1|1}{0|2}$). We present algorithms only for the canonical shapes as all other shapes can be counted with almost the same approach, only appropriately adjusted to the other location of the points. For instance $\frac{1|1}{0|2}, \frac{0|1}{2|1}, \frac{2|0}{1|1}$ and $\frac{0|2}{1|1}$ are all transformations of the same shape, but $\frac{1|2}{0|1}$ is not.

We call shapes $\frac{4|0}{0|0}, \frac{3|1}{0|0}, \frac{2|0}{2|0}$ and their transformations *non-proper*, because the division of the plane does not split the pattern both horizontally and vertically. All other shapes are called *proper.* Formally, a shape $\frac{a|b}{c|d}$ is proper iff $\min\{a+b, a+c, b+d, c+d\} \neq 0$. Now we are ready to state the crucial property that distinguishes the two main groups of 4-patterns:

Definition 6.2.1. *A 4-pattern that can form the shape $\frac{1|1}{1|1}$ is called non-trivial, and all other 4-patterns are called trivial.*

Note that there are 8 trivial 4-patterns: $1234, 1243, 2134, 2143, 4321, 4312, 3421, 3412$, all other 4-patterns are *non-trivial.* All trivial 4-patterns can form $\frac{0|2}{2|0}$ (or its reflection $\frac{2|0}{0|2}$), which cannot be formed by non-trivial patterns. Clearly, only non-trivial 4-patterns can be 4-partite.

Observe that there is no point in considering multi-weighted 4-partite patterns as, given a pattern, we know which element of the pattern appears in each of the regions: e.g. in region TL we have the first element of the pattern if $\sigma(1) > \sigma(2)$ and the second element in BL or vice versa if $\sigma(1) < \sigma(2)$, so for every element only one of its weights is relevant. For this reason we will consider only the case of counting weighted 4-partite 4-patterns, but not multi-weighted.

Minimum Base Ranges. For simplicity we assume that n is a power of 2. Let $\mathcal{T}_n$ be a full binary tree with n leaves numbered from 1 to n with internal nodes corresponding to the range of indices of leaves from their subtrees. We call the ranges corresponding to a node in the tree *base ranges.* Observe that the length of every range is a power of two and every $i \in [n]$ is contained in $\log n$ base ranges that are ancestors of i in $\mathcal{T}_n$. For a subset $S \subseteq [n]$, we define its *minimum base range* $\mathrm{MBR}(S)$ as the smallest base range from $\mathcal{T}_n$ containing all elements from S. Note that the node $v \in V(\mathcal{T}_n)$ corresponding to that range is the lowest common ancestor (LCA) of all leaves corresponding to the elements from S. If $|S| > 1$, v is not a leaf and we denote the two children of the node v as v^L and v^R which satisfy that, unifying the nodes with their base range, $S \subseteq v = v^L \cup v^R$ and $S \not\subseteq v^L$ and $S \not\subseteq v^R$.

We construct two full binary trees $\mathcal{T}_n$, one for x- and one for y-coordinates of points from S_π. Consider the Cartesian product $\mathcal{T}_n \times \mathcal{T}_n$ of the trees. For every pair $(R_x, R_y) \in \mathcal{T}_n \times \mathcal{T}_n$ of ranges, let $P_{R_x,R_y} = \{(i, \pi(i)) \in S_\pi : i \in R_x \wedge \pi(i) \in R_y\} = S_\pi \cap (R_x \times R_y)$ consist of all points from S_π that have their x-coordinate in R_x and y-coordinate in R_y. We call a pair (R_x, R_y) *relevant* if its set P_{R_x,R_y} is non-empty. As every number $i \in [n]$ belongs to $\mathcal{O}(\log n)$ base ranges, every point $(i, \pi(i))$ belongs to $\mathcal{O}(\log^2 n)$ sets P_{R_x,R_y} and hence we have:

Observation 6.2.2. *There are $\mathcal{O}(n \log^2 n)$ relevant pairs of ranges.*

General remarks. All the reductions we show in this work are split into several intermediate steps. Unless stated otherwise, each presented reduction runs in time almost linear in the total size of the input and the sum of sizes of the created instances of the other problem we reduce to.

In most of our approaches we will consider subsets W of points from S_π and count occurrences of a pattern among these points. Without loss of generality we can assume that points from W are a subset of $[|W|]^2$ and correspond to a permutation:

Lemma 6.2.3. *Counting occurrences of a pattern σ among a set W of points with pairwise distinct x- and y-coordinates can be reduced in $\tilde{\mathcal{O}}(|W|)$ time to counting σ in a permutation of $[|W|]$, that is $|W|$ points from $[|W|]^2$ with pairwise distinct coordinates. This also applies to the cases when both the pattern and the plane are divided horizontally, vertically or both horizontally and vertically.*

Proof. We sort each of the x-coordinates and replace them with their rank $\textsc{rank}(x) \in [|W|]$. If the plane was divided vertically with a line v, let $v_- = \max\{x : (x, y) \in W, x < v\}$ and we set the new dividing line $v' := \textsc{rank}(v_-) + 1/2$ which satisfies $x > v \iff \textsc{rank}(x) > v'$. After this transformation the horizontal order between pairs of points (and points and the vertical line, if it existed) is preserved and their x-coordinates are now in $[|W|]$. Clearly the chosen line does not pass through any of the new points as $v' \notin [|W|]$.

We proceed similarly for y coordinates. Then a subset of the transformed points forms an occurrence of the pattern σ if and only if the points before the transformation formed an occurrence of the pattern. □

With the above lemma, later we can assume that any instance of counting a pattern among t points considers a subset of t points from $[t]^2$ with pairwise distinct coordinates.

6.2.2 Range Queries and Short Patterns

Some of our algorithms use range queries for counting points in rectilinear (aligned with the x- and the y-axis) rectangles efficiently. Below we provide the precise interface for such queries.

Lemma 6.2.4 ([Cha88; JMS04])**.** *There exists a deterministic data structure that preprocesses a set of n weighted points in $\mathcal{O}(n \log n)$ time and answers queries about the number or the sum of weights of points inside rectilinear rectangles in $\mathcal{O}(\log n)$ time.*

For completeness, we explain the folklore algorithms for counting patterns shorter than 4. Although for $k = 3$ it is possible to design an algorithm using only range queries, we present an approach based on minimum base ranges that will be helpful for understanding our approach for patterns of length 4.

Theorem 6.2.5 (cf. [EL21, Corollary 1.1])**.** *For any pattern σ and weight functions $\lambda_{[|\sigma|]}$, where $|\sigma| < 4$ there exists an algorithm counting multi-weighted pattern σ in permutations of length n in $\tilde{\mathcal{O}}(n)$ time.*

Proof. Let $k = |\sigma|$. Clearly, if $k = 1$, we return $\sum_{x \in [n]} \lambda_1(x)$, the sum of weights of all elements. For $k = 2$ and the pattern 12 (respectively 21), for every element we count the sum of λ_2 weights of larger (smaller) elements to the right using a range query, and multiply it by the λ_1 weight of the considered element. The precise interface for range queries used in this proof is provided in Lemma 6.2.4. In the next paragraph, to avoid clutter sometimes we do not mention the weight functions $\lambda_{[|\sigma|]}$ when they are clear from the context.

Recall from the previous section the definition and properties of MBRs. Consider a 3-pattern σ. We build the tree $\mathcal{T}_n$ for x-coordinates and group all occurrences of σ by the MBR of their x-coordinates, that is we count the occurrence at positions $j_1 < j_2 < j_3$ while processing the minimum base range $v = \text{MBR}(\{j_1, j_2, j_3\}) \in V(\mathcal{T}_n)$. For a fixed range v, we write $\pi_{[v^L]}$ to denote the subsequence of π consisting of elements at positions from v^L and similarly for v^R. As $v = \text{MBR}(\{j_1, j_2, j_3\})$, there is at least one position from σ in v^L and one in v^R. Then the number of occurrences of σ that have their MBR in v is exactly $\#_{\sigma_{1|2}}(\pi_{[v^L]}|\pi_{[v^R]}) + \#_{\sigma_{2|1}}(\pi_{[v^L]}|\pi_{[v^R]})$, where $\#_{\sigma_{1|2}}(A|B)$ denotes the number of occurrences of 3-pattern σ such that first element is in the range A and the next two elements are in B, and similarly for $\#_{\sigma_{1|2}}$. Then we have:

$$\#_\sigma(\pi) = \sum_{v \in V(\mathcal{T}_n)} \#_{\sigma_{1|2}}(\pi_{[v^L]}|\pi_{[v^R]}) + \#_{\sigma_{2|1}}(\pi_{[v^L]}|\pi_{[v^R]})$$

In Lemma 6.2.6 we show how to compute $\#_{\sigma_{1|2}}(A|B)$ and $\#_{\sigma_{2|1}}(A|B)$ in $\tilde{\mathcal{O}}(|A| + |B|)$ time. As every element $i \in [n]$ belongs to $\mathcal{O}(\log n)$ minimum base ranges $\pi_{[v]}$, we have $\sum_{v \in V(\mathcal{T}_n)} |\pi_{[v]}| = \tilde{\mathcal{O}}(n)$ and hence the total complexity of counting weighted 3-patterns is $\tilde{\mathcal{O}}(n)$. □

Lemma 6.2.6. *For any 3-pattern σ, two disjoint ranges A, B of a permutation π and weight functions $\lambda_{[3]}$, we can compute $\#^\lambda_{\sigma_{1|2}}(A|B)$ and $\#^\lambda_{\sigma_{2|1}}(A|B)$ in $\tilde{\mathcal{O}}(|A| + |B|)$ time.*

Proof. Suppose first that $\sigma = 132$ and we show how to compute $\#^\lambda_{1|2}(A|B)$. We create an auxiliary function λ_{13} on B such that for every $b \in B$ we set $\lambda_{13}(b) := \lambda_3(b) \cdot \sum_{a \in A : \pi(a) < \pi(b)} \lambda_1(a)$, the weight of b multiplied by the sum of weights of elements in A smaller than $\pi(b)$. The sums can be retrieved in logarithmic time with orthogonal range query to the data structure from

Lemma 6.2.4 created on points from A with weights λ_1. Notice that λ_{13} counts length-2 sequences (a, b) such that $a \in A, b \in B$ and $\pi(a) < \pi(b)$ with weight $\lambda_1(a) \cdot \lambda_3(b)$ that constitute the first and last element of an occurrence of the pattern. Then the total number of weighted occurrences of σ is

$$\#_{132_{1|2}}(A|B) = \sum_{b \in B} \lambda_2(b) \cdot \sum_{\substack{b' \in B : b < b', \\ \pi(b) > \pi(b')}} \lambda_{13}(b')$$

For the case of $\sigma = 123$ we define $\lambda_{12}(b)$ similarly and then only the relation between b and b' changes: $\#_{123_{1|2}}(A|B) = \sum_{b \in B} \lambda_2(b) \cdot \sum_{\substack{b' \in B : b > b', \\ \pi(b) > \pi(b')}} \lambda_{12}(b')$. We compute the latter sums in logarithmic time using the range query data structure from Lemma 6.2.4 built on points from B with weights λ_{13} or λ_{12}. If $\sigma \in \{312, 321\}$ we proceed analogously but choose the next element to be smaller than the previous one, not larger.

Now we are left with patterns 213 and 231 that cannot be solved the same way. Observe that we have:

$$\#_1^{\lambda_1}(A) \cdot \#_{12}^{\lambda'}(B) = \#_{123_{1|2}}^{\lambda}(A|B) + \#_{213_{1|2}}^{\lambda}(A|B) + \#_{312_{1|2}}^{\lambda}(A|B)$$

where the weight functions $\lambda'_{[2]}$ are defined as follows: $\lambda'_1 = \lambda_2$ and $\lambda'_2 = \lambda_3$. As discussed in the first paragraph, in $\tilde{\mathcal{O}}(|A| + |B|)$ time we can calculate $\#_{123_{1|2}}^{\lambda}(A|B)$ and $\#_{312_{1|2}}^{\lambda}(A|B)$ and similarly $\#_1^{\lambda_1}(A)$ and $\#_{12}^{\lambda'}(B)$ by Theorem 6.2.5 and hence we can use the above property to calculate $\#_{213_{1|2}}^{\lambda}(A|B)$ in the same time. Analogously we compute $\#_{231_{1|2}}$ by considering $\#_1(A) \cdot \#_{21}(B)$.

Finally, we compute $\#_{\sigma_{2|1}}^{\lambda}(A|B)$ by reflecting the plane horizontally across the vertical line of division. Recall that by Lemma 6.2.3 we can assume that the points are from the set $S_\pi = \{(x, \pi(x)) : x \in [|\pi|]\}$ for some permutation π of $[|A| + |B|]$. Then we obtain an instance of counting $\textsc{rev}(\sigma)$, the pattern σ reversed, that is:

$$\#_{\sigma_{2|1}}^{\lambda}(A|B) = \#_{\textsc{rev}(\sigma)_{1|2}}^{\textsc{rev}(\lambda)}(\textsc{rev}(B)|\textsc{rev}(A))$$

where functions $\textsc{rev}(\lambda)_{[3]}$ are defined as $\textsc{rev}(\lambda)_j(x) := \lambda_{(4-j)}(|A| + |B| + 1 - x)$ for $j \in [3]$. □

Corollary 6.2.7. *For any division of a plane into parts A and B, given a multi-weighted 2- or 3-pattern σ and weight function(s) λ, in $\tilde{\mathcal{O}}(|A| + |B|)$ time we can compute $\#_{\sigma_{1|1}}(A|B)$ and $\#_{\sigma_{1|2}}(A|B), \#_{\sigma_{2|1}}(A|B)$, if the plane was divided vertically, and $\#_{\sigma_{\frac{1}{1}}}(\frac{A}{B})$ and $\#_{\sigma_{\frac{2}{1}}}(\frac{A}{B}), \#_{\sigma_{\frac{1}{2}}}(\frac{A}{B})$ if the plane was divided horizontally.*

Proof. Observe that $\#_{\sigma_{1|1}}(A|B)$ can be counted in $\tilde{\mathcal{O}}(|\pi|)$ time with the following formula:

$$\#_{\sigma_{1|1}}(A|B) = \sum_{(x, \pi(x)) \in A} \lambda_1(x) \cdot \sum_{\substack{(x', \pi(x')) \in B \\ [\pi(x) < \pi(x')] = [\sigma(1) < \sigma(2)]}} \lambda_2(x')$$

where the latter sum can be calculated using the orthogonal range query data structure from Lemma 6.2.4. By Lemma 6.2.6 we compute $\#_{\sigma_{1|2}}(A|B)$ and $\#_{\sigma_{2|1}}(A|B)$ in $\tilde{\mathcal{O}}(|\pi|)$ time.

In order to compute $\#_{\sigma_{\frac{1}{1}}}(\frac{A}{B})$ and $\#_{\sigma_{\frac{2}{1}}}(\frac{A}{B}), \#_{\sigma_{\frac{1}{2}}}(\frac{A}{B})$ we rotate the plane and the pattern by 90 degrees clockwise around the point $(0, 0)$ and normalize the points back to $[|\pi|]^2$, that is we transform the point $(x, \pi(x)) \in S_\pi$ into $(\pi(x), |\pi|+1-x)$ and similarly for the pattern σ obtaining pattern σ'. We create new weight functions $\lambda'_{[|\sigma|]}$ as follows: $\lambda'_j(x) := \lambda_{\sigma^{-1}(j)}(\pi^{-1}(x))$. Finally, the horizontal line $y = h$ is transformed into the line $x = h$ and division $\sigma_{\frac{a}{b}}$ is transformed to $\sigma'_{b|a}$. Observe that this gives us a complete reduction to the case of counting the pattern σ' in the plane divided vertically which can be done in $\tilde{\mathcal{O}}(|\pi|)$ time, as described in the previous paragraph and in Lemma 6.2.6. □

6.2.3 Extending and Shortening Patterns

In this subsection we show that counting a multi-weighted pattern σ can be reduced to counting a longer multi-weighted pattern σ' and, in some cases, to counting a shorter multi-weighted pattern σ''. This serves as a warm-up before the next sections and additionally will give us an infinite family of patterns for which counting is equivalent to counting non-trivial 4-patterns, which might be interesting on its own.

Lemma 6.2.8. *Counting a multi-weighted k-pattern σ can be reduced to counting multi-weighted $(k+1)$-pattern σ' where $\sigma'(j) = \sigma(j)$ for $j \in [k]$ and $\sigma'(k+1) = k+1$.*

Proof. Given a permutation π over $[n]$ and weight functions $\lambda_{[k]}$ we define permutation π' over $[n+1]$ as $\pi'(i) = \pi(i)$ for $i \in [n]$ and $\pi'(n+1) = n+1$ and construct weight functions $\lambda'_{[k+1]}$ as follows:

$$\lambda'_j(x) := \begin{cases} \lambda_j(x) & \text{for } j \in [k], x \in [n] \\ 1 & \text{for } j = k+1, x = n+1 \\ 0 & \text{otherwise} \end{cases}$$

Then we have $\#^{\lambda}_{\sigma}(\pi) = \#^{\lambda'}_{\sigma'}(\pi')$ as all the occurrences of σ' in π' with non-zero product of weights contain k elements from π forming pattern σ and the element $n+1$ at the end. □

Observe that we can obtain a similar reduction by adding $k+1$ at the beginning of the pattern ($\sigma'(1) = k+1, \sigma'(j+1) = \sigma(j)$ for $j \in [k]$) or by adding 1 at the beginning or end of the pattern. Then we say that σ can be extended to each of the four patterns σ'.

Definition 6.2.9. *A k-pattern σ can be extended to the following $(k+1)$-patterns:*

- $\sigma'_1 : \sigma'_1(j) = \sigma(j)$ *for* $j \in [k]$ *and* $\sigma'_1(k+1) = k+1$
- $\sigma'_2 : \sigma'_2(j) = \sigma(j) + 1$ *for* $j \in [k]$ *and* $\sigma'_2(k+1) = 1$
- $\sigma'_3 : \sigma'_3(j+1) = \sigma(j)$ *for* $j \in [k]$ *and* $\sigma'_3(1) = k+1$
- $\sigma'_4 : \sigma'_4(j+1) = \sigma(j) + 1$ *for* $j \in [k]$ *and* $\sigma'_4(1) = 1$

Lemma 6.2.10. *Counting a multi-weighted k-pattern σ satisfying that $\sigma(k) = k, \sigma(k-1) = k-1$ can be reduced in almost linear time to counting multi-weighted $(k-1)$-pattern σ'' where $\sigma''(j) = \sigma(j)$ for $j \in [k-1]$.*

Proof. Given weight functions $\lambda_{[k]}$ we define weight functions $\lambda''_{[k-1]}$ as follows:

$$\lambda''_j(x) := \lambda_j(x) \cdot \begin{cases} 1 & \text{for } j \in [k-2], x \in [n] \\ \sum_{\substack{x' : x < x' \\ \pi(x) < \pi(x')}} \lambda_k(x') & \text{for } j = k-1, x \in [n] \end{cases}$$

where all the sums are calculated in total $\tilde{\mathcal{O}}(|\pi|)$ time using the orthogonal range query data structure from Lemma 6.2.4 constructed on weights λ_k. Then we have $\#^{\lambda}_{\sigma}(\pi) = \#^{\lambda''}_{\sigma''}(\pi)$ as the λ_k weights of the last two elements from σ are aggregated in the λ''_{k-1} weight of the last element of σ''. □

Note that we can obtain a similar reduction when $\sigma(1) = k, \sigma(2) = k-1$ or when σ starts with $1, 2$ or ends with $2, 1$. If such an appropriate condition holds, we say that σ can be shortened to pattern σ''. For example, if $\sigma(k-1) = 2, \sigma(k) = 1$ we shorten σ to σ'' such that $\sigma''(j) = \sigma(j) - 1$ for $j \in [k-1]$.

Definition 6.2.11. *A k-pattern σ can be shortened to the following $(k-1)$-patterns:*

- $\sigma''_1 : \sigma''_1(j) = \sigma(j)$ *for* $j \in [k-1]$*, when* $\sigma(k-1) = k-1, \sigma(k) = k$
- $\sigma''_2 : \sigma''_2(j) = \sigma(j) - 1$ *for* $j \in [k-1]$*, when* $\sigma(k-1) = 2, \sigma(k) = 1$
- $\sigma''_3 : \sigma''_3(j) = \sigma(j+1)$ *for* $j \in [k-1]$*, when* $\sigma(1) = k, \sigma(2) = k-1$
- $\sigma''_4 : \sigma''_4(j) = \sigma(j+1) - 1$ *for* $j \in [k-1]$*, when* $\sigma(1) = 1, \sigma(2) = 2$

Corollary 6.2.12. *Counting a multi-weighted pattern σ can be reduced to counting patterns σ' that σ extends to and patterns σ'' that σ shortens to.*

Note that if a pattern σ starts or ends with 1 or $|\sigma|$, it extends to a pattern σ' and σ' shortens to σ. The above corollary gives us an infinite family of patterns equivalent to a single pattern. In particular we obtain that there are sixteen 5-patterns equivalent to non-trivial 4-patterns:

Corollary 6.2.13. *There are sixteen 5-patterns equivalent to some non-trivial 4-pattern:* 12435, 12453, 12534, 12543, 13245, 23145, 31245, 32145, 34521, 35421, 43521, 53421, 54123, 54132, 54213, 54231.

6.2.4 Reductions between 4-Patterns and 4-Partite 4-Patterns

In this section we show reductions between counting 4-partite 4-patterns and counting 4-patterns (recall Figure 2.2) and provide equivalence between all non-trivial 4-partite 4-patterns. The flavor of some of our arguments is similar to the ones used in [EL21], but we avoid the notion of corner tree formulas and explicitly state two technical lemmas that are required for our main result. First, using inclusion-exclusion principle, we show that counting 4-partite 4-patterns can be reduced to counting 4-patterns without the division of the plane. Recall that σ_4 denotes $\sigma_{\frac{1|1}{1|1}}$.

Lemma 6.2.14. *Counting 4-partite pattern σ_4 on n elements can be reduced to a constant number of instances of counting 4-pattern σ in permutations of total size $\mathcal{O}(n)$.*

Proof. When we omit the division of the plane and count the pattern σ in the plane, we additionally count also the quadruples of points forming the pattern but coming from not all of the 4 regions of the plane. To address this, we use inclusion-exclusion principle and add or subtract patterns on points from all possible subsets of regions. Then the number of 4-partite patterns is:

$$\#_{\sigma_4}\left(\frac{TL|TR}{BL|BR}\right) = \sum_{S\subseteq\{TL,TR,BL,BR\}} (-1)^{|S|}\cdot \#_\sigma\left(\bigcup_{Q\in S} Q\right)$$

where the union over regions consists of points from the regions preserving the relative order between them, as if there was no division of the plane into the regions. □

For the reduction in the other direction, first we need a technical lemma showing that all proper shapes but $\frac{1|1}{1|1}$ can be counted in $\tilde{\mathcal{O}}(n)$ time. At a high level, we will use orthogonal range queries to modify the weight functions and reduce the problem to counting 3-patterns. Recall that we do not have to consider rotations or symmetries of shapes separately, as the algorithm for them can be obtained analogously.

Lemma 6.2.15. *For any 4-pattern σ, weight functions $\lambda_{[4]}$ and division of the plane with n points, the shapes $\frac{3|0}{0|1}, \frac{2|0}{0|2}, \frac{1|1}{0|2}, \frac{1|2}{0|1}$ can be counted in $\tilde{\mathcal{O}}(n)$ time.*

Proof. First, in order to count shapes $\frac{3|0}{0|1}$ and $\frac{2|0}{0|2}$ it suffices to count appropriate 3-, 2- or 1-patterns on points in TL and BR and multiply the two numbers. By Theorem 6.2.5, this approach runs in $\tilde{\mathcal{O}}(n)$ time. For shapes $\frac{1|1}{0|2}$ and $\frac{1|2}{0|1}$, let σ' and $\lambda'_{[3]}$ be the pattern and weight functions obtained from σ and $\lambda_{[4]}$ after removing the first element from σ. Formally, σ' is a 3-pattern that satisfies $\sigma'(j) = \sigma(j+1) - [\sigma(j+1) > \sigma(1)]$ for $j \in [3]$ and we have weight functions $\lambda'_{[3]}$ such that $\lambda'_j(x) = \lambda_{j+1}(x)$ for $j \in [3]$. We will reduce counting the two shapes to counting σ' on some modifications of λ'.

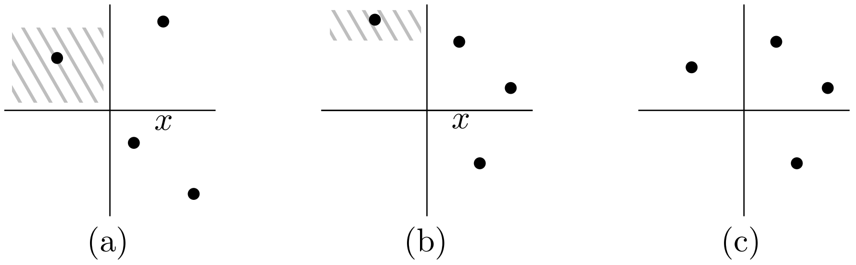


Figure 6.2: (a) A quadruple of points forming $\frac{1|1}{0|2}$ for $\sigma = 3241$. $\sigma' = 231, p = 2$ and $\lambda''_2(x)/\lambda'_2(x)$ is the sum of weights of points in TL, below $\pi(x)$ (the gray-dashed region). (b) A quadruple of points forming $\frac{1|2}{0|1}$ for $\sigma = 4312$. $\lambda_1''^{(t)}(x)$ calculates the sum of weights of points from TL above $\pi(x)$. (c) A quadruple of points forming $\frac{1|2}{0|1}$ for $\sigma = 3412$. For counting this shape we use both $\lambda''^{(t)}$ and $\lambda''^{(b)}$.

Consider the shape $\frac{1|1}{0|2}$. Observe that $(\sigma')^{-1}(3)$ is the index of the element from σ' that is in region TR and it satisfies that $1 + (\sigma')^{-1}(3) = \arg\max_{i\in\{2,3,4\}} \sigma(i)$. Informally, now we multiply the appropriate weight of each point $x \in TR$ by the sum of weights λ_1 of points in TL that are above or below $\pi(x)$ (depending on σ) in TL and focus on the 3-pattern formed by the elements in $TR \cup BR$. See Figure 6.2(a) for an illustration. We define auxiliary weight functions $\lambda''_{[3]}$ as follows. For $j \in [3]$ and $x \in TR \cup BR$ we set:

$$\lambda''_j(x) = \lambda'_j(x) \cdot \begin{cases} \sum_{\substack{(x',\pi(x')) \in TL \\ [\pi(x') < \pi(x)] = [\sigma(1) < \sigma(p+1)]}} \lambda_1(x'), & \text{if } j = (\sigma')^{-1}(3) \wedge (x, \pi(x)) \in TR \\ 1, & \text{otherwise} \end{cases}$$

We compute the sums in logarithmic time with orthogonal range query to the data structure from Lemma 6.2.4 for weights λ_1. Then we have:

$$\#^{\lambda}_{\sigma_{\frac{1|1}{0|2}}} \left(\frac{TL|TR}{BL|BR} \right) = \#^{\lambda''}_{\sigma'_{\frac{1}{2}}} \left(\frac{TR}{BR} \right)$$

We count $\sigma'_{\frac{1}{2}}$ in $\tilde{\mathcal{O}}(n)$ time by Corollary 6.2.7, so in the same time we can count the shape $\frac{1|1}{0|2}$.

For shape $\frac{1|2}{0|1}$ we define the "top" and "bottom" auxiliary functions $\lambda''^{(t)}_{[3]}, \lambda''^{(b)}_{[3]}$ as follows. For $j \in [3]$ and $x \in TR \cup BR$ we set:

$$\lambda''^{(t)}_j = \lambda'_j(x) \cdot \begin{cases} \sum_{\substack{(x',\pi(x')) \in TL \\ \pi(x') > \pi(x)}} \lambda_1(x'), & \text{if } j = (\sigma')^{-1}(3) \wedge (x, \pi(x)) \in TR \\ 1, & \text{otherwise} \end{cases}$$

informally we increased the $(\sigma')^{-1}(3)$-th weight of x by the sum of weights of points from TL that are above $(x, \pi(x))$. See Figure 6.2(b). Note that with these weights we can count the shape $\frac{1|2}{0|1}$ for the case when $\sigma(1) = 4$ in $\tilde{\mathcal{O}}(n)$ time by computing $\#^{\lambda''^{(t)}}_{\sigma'_{\frac{2}{1}}} \left(\frac{TR}{BR} \right)$. Similarly, for the case of $\sigma(1) = 2$ we use weights $\lambda''^{(b)}$:

$$\lambda''^{(b)}_j = \lambda'_j(x) \cdot \begin{cases} \sum_{\substack{(x',\pi(x')) \in TL \\ \pi(x') < \pi(x)}} \lambda_1(x'), & \text{if } j = (\sigma')^{-1}(2) \wedge (x, \pi(x)) \in TR \\ 1, & \text{otherwise} \end{cases}$$

However, this approach does not immediately work for the case when $\sigma(1) = 3$. Let $\sigma'^{(f)}$ be the 4-pattern such that its first element is f and after removing the first element (and normalizing) we obtain σ', as described in the beginning of the proof. For example, $312^{(2)} = 2413$. Notice that in the previous paragraph we showed how to count the shape $\frac{1|2}{0|1}$ for $\sigma \in \{\sigma'^{(2)}, \sigma'^{(4)}\}$ and we are left to cover the pattern $\sigma'^{(3)}$. Observe that we have:

$$\sum_{f \in \{2,3,4\}} \#^{\lambda}_{\sigma'^{(f)}_{\frac{1|2}{0|1}}} \left(\frac{TL|TR}{BL|BR} \right) = \#^{\lambda_1}_{1}(TL) \cdot \#^{\lambda'}_{\sigma'_{\frac{2}{1}}} \left(\frac{TR}{BR} \right)$$

as the right-hand side disregards the relative position of the element from TL to the elements from TR whereas at the left-hand side we consider all the three possibilities separately. Using the observations provided above, all but one terms of the above equation can be calculated in $\tilde{\mathcal{O}}(n)$ time. Now we can count the shape $\frac{1|2}{0|1}$ for patterns σ with $\sigma(1) = 3$ in $\tilde{\mathcal{O}}(n)$ time in the following way:

$$\#^{\lambda}_{\sigma_{\frac{1|2}{0|1}}} \left(\frac{TL|TR}{BL|BR} \right) = \#^{\lambda_1}_{1}(TL) \cdot \#^{\lambda'}_{\sigma'_{\frac{2}{1}}} \left(\frac{TR}{BR} \right) - \#^{\lambda''^{(t)}}_{\sigma'_{\frac{2}{1}}} \left(\frac{TR}{BR} \right) - \#^{\lambda''^{(b)}}_{\sigma'_{\frac{2}{1}}} \left(\frac{TR}{BR} \right) \quad \square$$

Recall that, given a division of the plane into 4 regions, an occurrence of a 4-pattern σ is 4-partite if all its elements are in pairwise distinct regions. In the following lemma we show that we can count 4-patterns by counting 4-partite 4-patterns. At a high level, every occurrence of the pattern is counted while considering the division of the plane aligned with the division of minimum base ranges containing all coordinates of the four points. We proceed similarly as in the proof of Theorem 6.2.5 but now consider minimum base ranges of both the coordinates x and y.

Lemma 6.2.16. *Counting a multi-weighted 4-pattern σ on n elements can be reduced in $\tilde{\mathcal{O}}(n)$ time to multiple instances of counting weighted 4-partite patterns σ_4 of total size $\tilde{\mathcal{O}}(n)$.*

Proof. Recall the definition and properties of MBRs from Section 6.2.1. $\mathrm{MBR}(S)$, the minimum base range of a set $S \subseteq [n]$ is the minimum base range containing all elements of S in the full binary tree $\mathcal{T}_n$ on $n' = 2^{\lceil \log n \rceil}$ leaves and every node $R \in \mathcal{T}_n$ corresponds to a range of consecutive elements from $[n]$. By Observation 6.2.2 we have that there are $\tilde{\mathcal{O}}(n)$ distinct pairs $(R_x, R_y) \in \mathcal{T}_n \times \mathcal{T}_n$ for which there exists an $i \in [n]$ such that $i \in R_x$ and $\pi(i) \in R_y$. We can retrieve all such pairs in $\tilde{\mathcal{O}}(n)$ time by iterating through all points from S_π and generating the set of all relevant pairs of ranges. Recall that $P_{R_x,R_y} = S_\pi \cap (R_x \times R_y)$. In terms of the permutation π, R_x corresponds to its substring and R_y restricts the values of elements to a particular interval.

For every relevant pair of ranges (R_x, R_y) with P_{R_x,R_y} of at least 4 points, we consider the plane restricted only to points from P_{R_x,R_y} and divided in the following way. As all points from S_π have distinct coordinates and $|P_{R_x,R_y}| \geq 4$, the range R_x contains at least 4 elements, so is not a leaf in $\mathcal{T}_x$ and has two children R_x^L, R_x^R in $\mathcal{T}_x$. The two ranges R_x^L and R_x^R are disjoint so we can find a vertical line that separates them. Notice that any such line does not pass through a point from S as R_x^L and R_x^R are two consecutive ranges in $\mathcal{T}_n$. We find a horizontal line separating the range R_y in the same way. For the set of points P_{R_x,R_y} and the above division of the plane, we count all shapes $\frac{3|0}{0|1}, \frac{2|0}{0|2}, \frac{1|1}{0|2}, \frac{1|2}{0|1}$ and all their possible rotations and symmetries in $\tilde{\mathcal{O}}(|P_{R_x,R_y}|)$ time, by Lemma 6.2.15. Finally, we need to count the shape $\frac{1|1}{1|1}$, the multi-weighted 4-partite pattern σ_4 on the set P_{R_x,R_y} and sum up all the obtained results.

Now we show that the above procedure counts every occurrence of the pattern σ exactly once, while considering the pair of minimum base ranges for both coordinates of the points from the occurrence. Formally, an occurrence g of σ on positions $i_1 < i_2 < i_3 < i_4$ is counted only for the pair of ranges (R_x, R_y) where $R_x = \mathrm{MBR}(\{i_1, i_2, i_3, i_4\})$ and $R_y = \mathrm{MBR}(\{\pi(i_1), \pi(i_2), \pi(i_3), \pi(i_4)\})$ and the appropriate shape, depending on the position of points from $\{(i_j, \pi(i_j)) : j \in [4]\}$ with respect to the division. Suppose the contrary, that g is counted for another pair of ranges (R'_x, R'_y) where $R'_x \neq R_x$, for $R'_y \neq R_y$ the reasoning is similar. If $\{i_1, i_2, i_3, i_4\} \not\subseteq R'_x$, for some j the point $(i_j, \pi(i_j))$ will not be present in the considered instance. Otherwise, from the structure of base ranges we have that $\mathrm{MBR}(\{i_1, i_2, i_3, i_4\})$ is fully contained in one half of R'_x. In this case g also will not be counted, because it forms a non-proper shape for the considered division ($\frac{2|0}{2|0}$, $\frac{3|1}{0|0}$ or $\frac{4|0}{0|0}$ or their rotations) and we do not count such shapes.

As every point from S_π is included in $\mathcal{O}(\log^2 n)$ sets P_{R_x,R_y}, the total size of all the considered sets is $\tilde{\mathcal{O}}(n)$ and hence counting shapes different than $\frac{1|1}{1|1}$ takes $\tilde{\mathcal{O}}(n)$ time. Finally, the total size of the instances of counting weighted 4-partite pattern σ_4 is also $\tilde{\mathcal{O}}(n)$. □

By definition, trivial patterns do not form the $\frac{1|1}{1|1}$ shape, so the reduced instances have always zero occurrences of the 4-partite pattern, which can be returned in constant time. Hence:

Corollary 6.2.17 (cf. [EL21, Corollary 1.2])**.** *All trivial 4-patterns (*$1234, 1243, 2134, 2143, 4321, 4312, 3421, 3412$*) in permutations of length n can be counted in* $\tilde{\mathcal{O}}(n)$ *time.*

Now we show that in fact all (non-trivial) 4-partite 4-patterns are equivalent by a linear-time transformation of the considered set of points. At a high level, we will show that reversing the order of points in any of the four parts of the plane (left, top, ...) allows us to slightly modify the pattern. Recall that by Lemma 6.2.3 we can assume that it holds $S_\pi \subseteq [|\pi|]^2$ for any considered subset of points S_π and that for 4-partite patterns it suffices to consider only weighted patterns, not multi-weighted.

Lemma 6.2.18. *Counting any non-trivial weighted 4-partite pattern σ_4 can be reduced to counting any other non-trivial weighted 4-partite pattern σ'_4.*

Proof. We start with showing that by reversing the points in the left part of the plane we can swap the first two elements of the pattern:

$$\#_{abcd_4}\left(\frac{TL|TR}{BL|BR}\right) = \#_{bacd_4}\left(\text{REV}\left(\frac{TL}{BL}\right)\frac{|TR}{|BR}\right).$$

Formally, suppose that we need to count the 4-partite pattern $abcd_4$ in the plane divided as follows: $\frac{TL|TR}{BL|BR}$ and the rightmost point from the left part $(TL \cup BL)$ has the x-coordinate z. We replace every point (x, y) from the left part with $(z + 1 - x, y)$ and translate the weight function to the new points, so we set: $\lambda'(x) = \lambda(z + 1 - x)$ for $x \in [z]$ and $\lambda'(x) = \lambda(x)$ for $x > z$. Then, only the horizontal order of points from the left part is reversed and any 4-partite occurrence of the pattern $abcd_4$ in the original instance corresponds to a 4-partite occurrence of the pattern $bacd_4$ in the transformed instance. Similarly, after reversing the right part we obtain the pattern $abdc_4$ from $abcd_4$. When we reverse the (vertical) order of the top or bottom part, we swap respectively elements 3 and 4 or 1 and 2 in the pattern. For example, by reversing the top part, from the pattern 1324_4 we obtain the pattern 1423_4.

Observe that operations in any two parts of the plane are independent, we can apply any subset of them and obtain either of the 16 possible non-trivial 4-partite patterns. See Figure 6.3 with the precise description of operations between the patterns. Thus, we can transform in linear time any instance of counting non-trivial 4-partite pattern σ_4 to an instance of counting any of the 16 possible non-trivial 4-partite patterns. □

In the next section we show how to count 4-partite weighted 4-patterns using efficient algorithms for counting 4-cycles in sparse graphs and vice versa: how to count 4-cycles in sparse graphs by counting 4-partite patterns.

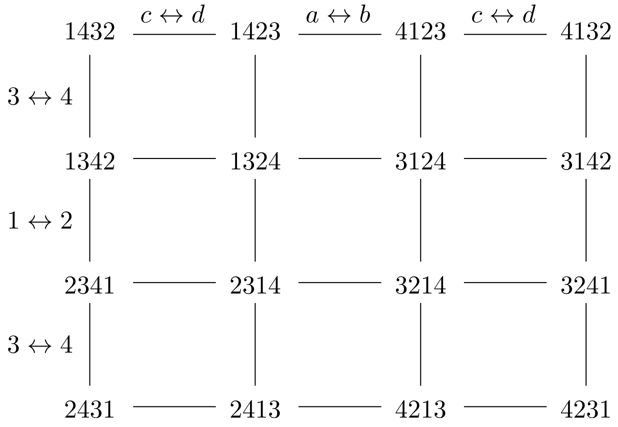


Figure 6.3: Reductions between non-trivial 4-partite patterns described in Lemma 6.2.18. For a pattern $abcd_4$, the operation $a \leftrightarrow b$ ($c \leftrightarrow d$) swaps the first (second) pair of its elements and corresponds to reversing left (right) part of the plane. Operation $1 \leftrightarrow 2$ ($3 \leftrightarrow 4$) swaps elements 1 and 2 (3 and 4) in the pattern and corresponds to reversing bottom (top) part of the plane.

6.3 Equivalence between Counting 4-Patterns and 4-Cycles

By Lemma 6.2.18, all non-trivial 4-partite patterns are equivalent so in the following claims it suffices to consider only one of them and we will focus on counting the pattern 1324_4. In the following lemma we show that counting non-trivial 4-partite patterns can be reduced to counting 4-cycles in 4-circle-layered multigraphs. At a high level, we will group all occurrences of the pattern by the minimum base ranges of coordinates of points in each of the parts of the plane.

Lemma 6.3.1. *Counting a non-trivial weighted 4-partite pattern on n points can be reduced to an instance of counting 4-cycles in a 4-circle-layered multigraph on $\tilde{\mathcal{O}}(n)$ edges with multiplicities bounded by $U \cdot n$, for U being the bound on the original weight function λ.*

Proof. For a permutation π and division of the plane with points S_π we need to construct a 4-circle-layered multigraph in such a way that the number of 4-cycles in the graph gives us the number of occurrences of the pattern. We consider four full binary trees $\mathcal{T}_n^L, \mathcal{T}_n^R, \mathcal{T}_n^B, \mathcal{T}_n^T$ for each part of the plane separately. For each base range in the trees we create a separate node in the new 4-partite graph.

Now we process all points from S_π grouped by their region. Suppose we process a point $(x, y) \in S_\pi$ from the top-right region. We iterate over all pairs $(R_R, R_T) \in \mathcal{T}_n^R \times \mathcal{T}_n^T$ of base ranges such that $x \in R_R$ and $y \in R_T$ and the ranges are not singletons (leaves in $\mathcal{T}_n$), so contain at least two elements. Recall that we focus on the pattern 1324, because the choice of the particular pattern is crucial in the following step. We add edge (R_T, R_R) with multiplicity $\lambda(x)$ to the 4-circle-layered multigraph if x is in the right half of R_R and y is in the top half of R_T. This means that the point (x, y) can be a part of an occurrence of pattern 1324 in which R_T is the MBR of y-coordinates of the two top points and R_R is the MBR of x-coordinates of the two right points. See Figure 6.4. We proceed similarly for the remaining three regions, modifying

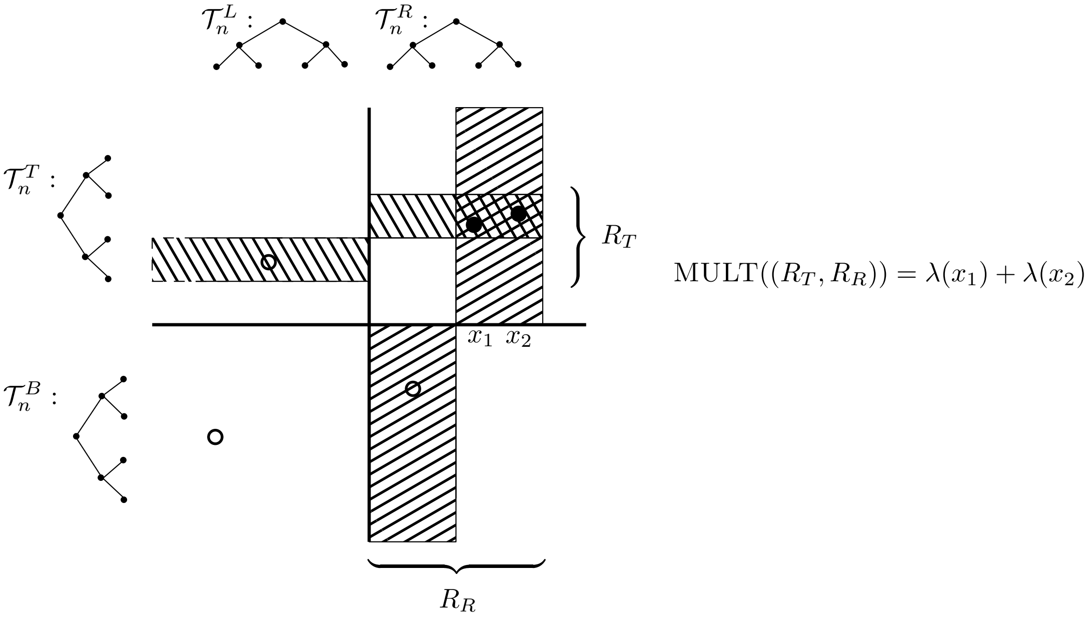


Figure 6.4: We consider four full binary trees $\mathcal{T}_n^L, \mathcal{T}_n^R, \mathcal{T}_n^B, \mathcal{T}_n^T$ for each part of the plane separately and group occurrences of patterns by the MBRs of coordinates in each part of the plane. Points from appropriate halves of MBRs from each two consecutive parts increase multiplicity of the appropriate edge in the graph.

only the condition for including an edge, based on the position of elements of pattern 1324 inside the considered region.

If an edge is inserted more than once, we simply increase its multiplicity, which can be stored e.g. in a balanced binary search tree. As every point from S_π adds at most $\mathcal{O}(\log^2 n)$ edges, in total there are $\mathcal{O}(n \log^2 n) = \tilde{\mathcal{O}}(n)$ edges in the graph. Clearly, the constructed directed multigraph is 4-partite as we connect nodes from $\mathcal{T}_n^T$ to the nodes from $\mathcal{T}_n^R$, from $\mathcal{T}_n^R$ to $\mathcal{T}_n^B$ etc. Finally, observe that for a horizontal range R and vertical range R', the multiplicity of an edge connecting nodes corresponding to ranges R and R' is upper bounded by the total weight of all points in $S_\pi \cap (R \times R')$. Hence multiplicities of edges in the graph are bounded by $U \cdot n$. □

By combining (recall Figure 2.2) the above result with Lemma 6.2.16 and Theorem 2.2.6 we obtain:

Corollary 2.2.8. *There exists an algorithm counting 4-patterns in permutation of length n in $\mathcal{O}(n^{1.48})$ time.*

Finally, to conclude the equivalence between counting 4-partite patterns and counting cycles in 4-circle-layered graphs, we describe the reduction from counting 4-cycles in 4-circle-layered graphs to counting non-trivial patterns. The idea is to first embed the graph in the plane so that every group V_i of nodes corresponds to a half-plane and edges to points in the plane. Then every 4-cycle corresponds to a rectangle with all corners in distinct quadrants. Now we appropriately tilt each quadrant, so that every rectangle corresponds to an occurrence of the pattern 1324_4. However, this change introduces many more occurrences of the pattern as now we have slightly weaker constraints on the relative position of points. This is corrected by applying the inclusion-exclusion principle for different ways of tilting the quadrants.

We remark that our approach is similar to that of Berendsohn et al. [BKM19, Section 5]. They showed a reduction from Partitioned Subgraph Isomorphism to counting short patterns in permutations by embedding the input graph in the plane with appropriate tilting and using the inclusion-exclusion principle. However, while their reduction works for arbitrary subgraphs of size k, this comes at the cost of increasing the size of the permutation pattern to $7k+1$, which in our case would result in a permutation pattern on 29 elements, hence not giving us the desired tight connection between counting 4-cycles and 4-patterns.

Lemma 6.3.2. *Counting 4-cycles in a 4-circle-layered simple graph on m edges can be reduced in $\tilde{O}(m)$ time to a constant number of instances of counting a non-trivial pattern in a permutation of length m.*

Proof. Given a 4-circle-layered graph $G = (V_0 \dot\cup V_1 \dot\cup V_2 \dot\cup V_3, E)$, where $E \subseteq \bigcup_i V_i \times V_{(i+1) \bmod 4}$, we will embed it in the plane and construct a constant number of instances of counting a non-trivial 4-partite pattern. As Lemma 6.2.18 guarantees that all such patterns are equivalent, we can focus only on the pattern 1324.

We first relabel the vertices in the following way. We (arbitrarily) map bijectively V_0 to $[|V_0|]$, V_2 to $[|V_0|+1, |V_0|+|V_2|]$ so that they correspond to x coordinates from $[|V_0|+|V_2|]$ and proceed similarly for y coordinates: we map V_3 to $[|V_3|]$ and V_1 to $[|V_3|+1, |V_3|+|V_1|]$. The division of the plane separates the sets, e.g. $v = |V_0| + \frac{1}{2}$ and $h = |V_3| + \frac{1}{2}$. Then every half of the plane corresponds to a part of the graph in the clockwise order starting from V_0 mapped to the left half. For every edge in the graph we create a point in the plane. Then we get a set of m points and every 4-cycle in G corresponds to a rectangle with corners in points in distinct quadrants.

Now we would like to transform the constructed set of points into a number of point sets S_π for some permutations π. Intuitively, every 4-cycle from G will correspond to an occurrence of the pattern 1324_4. Notice that there might be many edges incident to a node, so in the beginning some points have equal x- or y-coordinate, which we need to avoid. At first we will guarantee that no two points from distinct quadrants have equal x- or y-coordinates, which is already sufficient to be able to define an occurrence of the 4-partite pattern 1324_4. In the end we will show that we can slightly shift all points preserving relationships between points from distinct quadrants and additionally ensuring uniqueness of coordinates inside each quadrant. This will be sufficient to obtain a set of points corresponding to a permutation. Consider the following transformation of the plane:

$$\frac{TL|TR}{BL|BR} \rightarrow \frac{TL + (\frac{1}{5}, 0)|TR + (0, \frac{1}{5})}{BL + (0, -\frac{1}{5})|BR + (-\frac{1}{5}, 0)}$$

where by adding a vector to a region we denote shifting all points from the region by the vector. Let TL', TR', BL', BR' be the sets of points after the transformation. Informally, TL' is TL slightly shifted right, TR' is TR slightly shifted up etc., see Figure 6.5(a). Observe that now every 4-cycle from G corresponds to an occurrence of 1324_4 (see Figure 6.5(b) and its explanation in the caption), but there are also many more other occurrences of the pattern, which do not correspond to a cycle from G. More precisely, every occurrence of the pattern 1324_4 corresponds to 4 edges from G, but we cannot ensure that they form a cycle, or equivalently, that every two consecutive edges share an endpoint, see Figure 6.5(c).

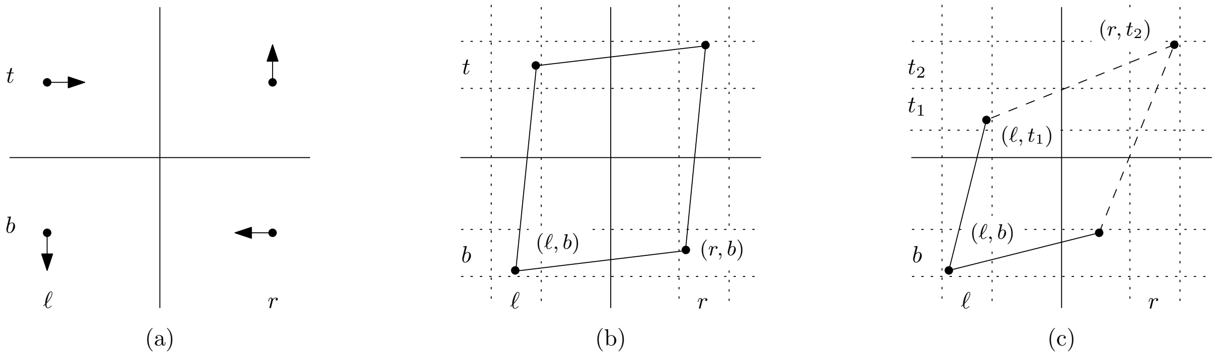


Figure 6.5: (a) Slightly shifting all points guarantees that points from distinct quadrants do not share a coordinate. (b) Every cycle from the graph corresponds to an occurrence of 1324_4. We mark the area of the "small shifts" between the dashed lines, e.g. all points that initially had y-coordinate equal to t now are between the two horizontal dashed lines surrounding t. (c) Some occurrences of 1324_4 do not correspond to a cycle in G, as the consecutive edges do not share endpoints. Points corresponding to consecutive edges that share an endpoint are connected with a solid line (e.g. (ℓ, b) and (ℓ, t_1)) and with a dashed line if they do not share (e.g. (ℓ, t_1) and (r, t_2)).

In particular, after the above transformation, in every occurrence of 1324_4 its two points from the left half-plane: $(x, y - \frac{1}{5}) \in BL'$ and $(x' + \frac{1}{5}, y') \in TL'$ satisfy that $x' + \frac{1}{5} > x$, but the two edges corresponding to these points share an endpoint only when $x' = x$. On the other hand, if we slightly modify the above transformation and set $TL' = TL + (-\frac{1}{5}, 0)$, we obtain that in every occurrence of 1324_4, when $x' - \frac{1}{5} > x$ it holds that $x' > x$ and the two edges corresponding to points from the occurrence in the left half-plane cannot share an endpoint. Hence by modifying the transformation we control if the two edges can share an endpoint (which we require in 4-cycle) or not. Now we use this property for all half-planes and plug the modified transformations into the inclusion-exclusion principle:

$$\#_{C_4}(G) = \sum_{S \subseteq \{L,R,B,T\}} (-1)^{|S|} \#_{1324_4} \left(\frac{TL + (\delta_L(S), 0) | TR + (0, \delta_T(S))}{BL + (0, -\delta_B(S)) | BR + (-\delta_R(S), 0)} \right)$$

where $\delta_X(S) = \frac{1}{5}$ if $X \in S$ or $-\frac{1}{5}$ otherwise. This reduction creates 16 instances of counting 1324_4 in sets of points divided into 4 quadrants such that points from distinct quadrant do not share a coordinate, but points from the same quadrant can share a coordinate. To avoid this issue, before shifting the plane we first transform every point (x, y) into $(x + \frac{y}{10n}, y + \frac{x}{10n})$. For instance, a point $(x, y) \in TL$ is transformed to $(x + \frac{y}{10n} + \delta_L(S), y + \frac{x}{10n})$. Notice that the choice of lengths of the shifts guarantees that no two points have the same x- or y- coordinate and the new coordinates are within $[-\frac{3}{10}, \frac{3}{10}] \times [-\frac{3}{10}, \frac{3}{10}]$ square with respect to the original location of points and the relative order between points from distinct quadrants is preserved. In the obtained instances all points have non-integer coordinates, but we can normalize them into $[m]^2$ preserving the relative order between the points as in Lemma 6.2.3. □

These two lemmas together with the results from the previous section and Theorem 2.2.6 finish the chain of reductions provided in Figure 2.2 and show the main result of this part of the paper:

Theorem 2.2.7. *For every $\gamma \geq 1$, an algorithm for counting 4-cycles in a graph on m edges in $\tilde{\mathcal{O}}(m^\gamma)$ time implies an algorithm for counting non-trivial 4-patterns in a permutation of length n in $\tilde{\mathcal{O}}(n^\gamma)$ time and vice versa.*

In particular, this gives us a reason, why a significant improvement to this algorithm will be a breakthrough:

Corollary 2.2.9. *For every $\varepsilon > 0$, there exists no algorithm that can count non-trivial 4-patterns in permutation of length n in $\mathcal{O}(n^{4/3-\varepsilon})$ time unless Conjecture 2 is false.*

6.4 From Counting 4-Cycles to Quartet Distance

6.4.1 Notation and Definitions

We consider unrooted trees on n leaves with distinct labels from $\{1, 2, \ldots, n\}$, and identify leaves with their labels. The quartet distance between two such trees T_1, T_2 is defined as the number of subsets of four distinct leaves $\{a, b, c, d\}$ (called quartets) such that the subtrees induced by $\{a, b, c, d\}$ in both trees are not related by the same topology. There are four possible topologies of trees induced by four leaves, see Figure 2.1. We call the first three of them *butterflies* and the last one is a *star*. They are also called *resolved* and *unresolved* quartets, respectively.

Recall that we reduced counting 4-cycles in a simple graph to counting 4-cycles in a simple bipartite graph in Lemma 6.1.1. In this section we provide a reduction from counting 4-cycles in a bipartite graph to computing the quartet distance between two trees. Consequently, there is no algorithm for quartet distance that runs significantly faster than in $\mathcal{O}(n^{1.48})$ time unless we can count 4-cycles faster. In particular, existence of an $\mathcal{O}(n \log n)$ algorithm for the quartet distance would imply a surprisingly fast algorithm for counting 4-cycles. Now we show how to, given a bipartite simple graph, construct two trees in such a way that the number of 4-cycles in the original graph can be efficiently extracted from the quartet distance between the trees.

6.4.2 Properties of Shapes in Bipartite Graphs

In this section we show how to reduce counting 4-cycles in a simple bipartite graph to computing the quartet distance between two trees. We first provide some insight into the structure of 4-edge subgraphs of a bipartite graph which we call shapes. We illustrate them with little symbols, e.g. $\lll$, $\lessgtr$, $\underline{\lneq}$ or ≣.

Properties of shapes. We first consider all nodes with non-zero degrees in a shape. For instance, nodes in shape $\lneq$ have the following (non-zero) degrees: $3, 1$ on the left side and $1, 1, 2$ on the right side. We call the sorted list of non-zero degrees of V_1 (respectively V_2) in a shape its left (respectively right) representation. Then two representations separated by a dash form the representation of a shape. For instance, the representation of $\lneq$ is $3, 1 - 2, 1, 1$. There are 5 possible left and right representations: $(4), (3, 1), (2, 2), (2, 1, 1)$ and $(1, 1, 1, 1)$. Next, the representation of a shape almost uniquely determines the shape, for instance $3, 1 - 1, 1, 1, 1$ corresponds only to one shape $\underline{\ll}$. The notion of representations gives us a systematic way to list all 16 possible shapes. In Table 6.1 we list 6 of them and omit another 6 shapes which are their mirror reflections, that is they are reflections along the vertical axis. For example, $\underline{\gg}$ is the mirror reflection of $\underline{\ll}$ and we omit it. In Table 6.2 we list all the remaining 4 shapes which remain unchanged under mirror reflection. Observe that the only representation which does not uniquely describe a shape is $2, 1, 1 - 2, 1, 1$ as it represents two distinct shapes: $\lessgtr$ and $\underline{\lneq}$.

$\lll$	$\lneq$	$\underline{\ll}$	$\lesssim$	$\overset{<}{<}$	$\leqq$
$4 - 1, 1, 1, 1$	$3, 1 - 2, 1, 1$	$3, 1 - 1, 1, 1, 1$	$2, 2 - 2, 1, 1$	$2, 2 - 1, 1, 1, 1$	$2, 1, 1 - 1, 1, 1, 1$

Table 6.1: Six possible shapes which change under mirror reflection.

⧖	$\lessgtr$, $\underline{\lneq}$	≣
$2, 2 - 2, 2$	$2, 1, 1 - 2, 1, 1$	$1, 1, 1, 1 - 1, 1, 1, 1$

Table 6.2: Four possible shapes which remain unchanged under mirror reflection.

The reduction. On a high level, we design the reduction in such a way that the quartet distance between the constructed trees can be obtained by counting particular shapes in the considered simple graph $G = (V_1 \cup V_2, E)$ and adding up the results. Some of the shapes can be counted in linear time, for instance the number of shapes $\lll$ in G is $(\# \lll) = \sum_{v \in V_1} \binom{\deg(v)}{4}$. As the graph G will be clear from the context, to simplify the presentation we write $(\# \lll)$ instead of $\# \lll (G)$, and similarly for all other shapes. For some shapes it is more difficult to present such a compact formula for counting it, e.g. $\# \leqq$. An extreme example is # ⧖ (G) which is exactly the sought number of 4-cycles. We will relate the number of such difficult shapes to (# ⧖) $:= \#C_4(G)$ (abbreviately denoted as C_4) and then express the quartet distance as a multiple of the number of 4-cycles plus some value that we can compute in linear time. Solving this simple equation gives us C_4.

Given a bipartite graph $G = (V_1 \cup V_2, E)$ we construct the trees T_1 and T_2 in the following way. Tree T_i consists of nodes representing all non-isolated nodes from V_i attached to the central node and nodes representing edges from E attached to the node corresponding to their endpoint from V_i. Note that the other endpoint from each edge is in V_{3-i}, as G is bipartite. Hence there is a bijection between the leaves of T_i and E, see Figure 6.6 for an example. We note that the trees T_1 and T_2 are unrooted, but for convenience we draw them as rooted ones.

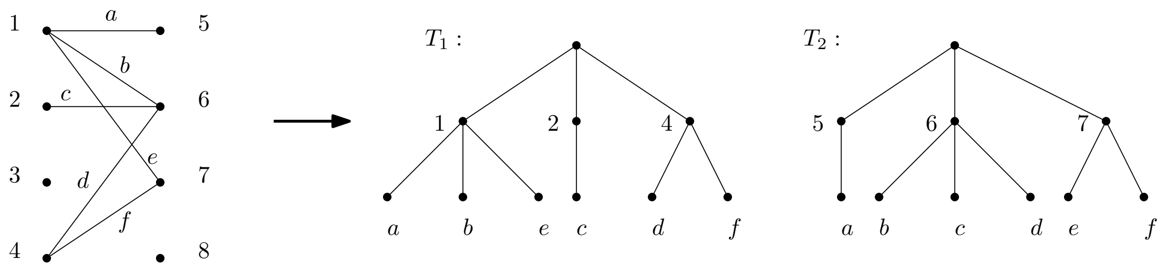


Figure 6.6: Instance of the quartet distance problem obtained from the bipartite graph on the left.

Quartets. Recall that in the quartet distance between trees we consider subtrees induced by four leaves. Observe that the above construction guarantees that the topology (star or butterfly) of a subtree of T_1 (respectively T_2) induced by a set of four leaves $L = \{e_1, e_2, e_3, e_4\}$ is uniquely determined by the left (respectively right) representation of the graph consisting of edges $\{e_1, \dots, e_4\}$, see Figure 6.7.

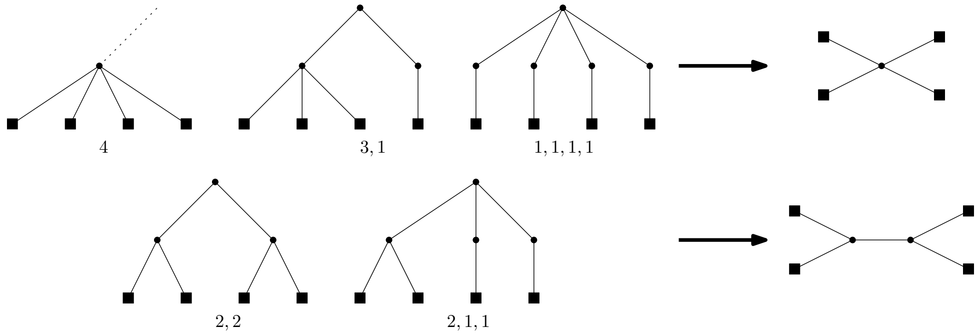


Figure 6.7: Left: all five possible representations and their corresponding trees. Right: the corresponding tree topology is either a star (upper row) or a butterfly (lower row).

As the quartet distance between T_1 and T_2, denoted as $\mathtt{QD}(T_1, T_2)$, is the number of sets of four leaves that are not related by the same topology in both trees, $\mathtt{QD}(T_1, T_2)$ equals $\binom{\#\text{ of leaves}}{4}$ minus the number of subsets of four leaves that are related by the same topology in both trees. From now on we will focus on computing only the latter number. The agreeing topologies can be either stars (upper row in Figure 6.7) or butterflies (bottom row in Figure 6.7). For stars, the order of labels on the leaves does not matter, so it is enough that the quartet induces a star in both trees, which means that the left and right representations of their shape must be either 4 or 3, 1 or 1, 1, 1, 1. Hence there are five shapes which induce a star in both trees: ⋖, ⋗, $\leq$, $\geq$ and ≡. Next, a quartet induces a butterfly in both trees if its left and right representations are either 2, 2 or 2, 1, 1. Recall from Table 6.2 that there are five different shapes with both their representations either 2, 2 or 2, 1, 1, as there are two shapes represented by $2, 1, 1 - 2, 1, 1$: ≶, ≰. In Figure 6.8 we present all the five shapes with the labels of leaves marked. Observe that only the shape ≶ has matching leaf labels between trees T_1 and T_2. Summarizing the above discussion, we obtain the following equality that shows two different ways of counting quartets that are related by the same topology in both the trees T_1 and T_2:

$$\binom{\#\text{ of leaves}}{4} - \mathtt{QD}(T_1, T_2) = \Big((\#\lessdot) + (\#\gtrdot) + (\#\underline{<}) + (\#\underline{>}) + (\#≣)\Big) + (\#\lessgtr) \qquad (6.1)$$

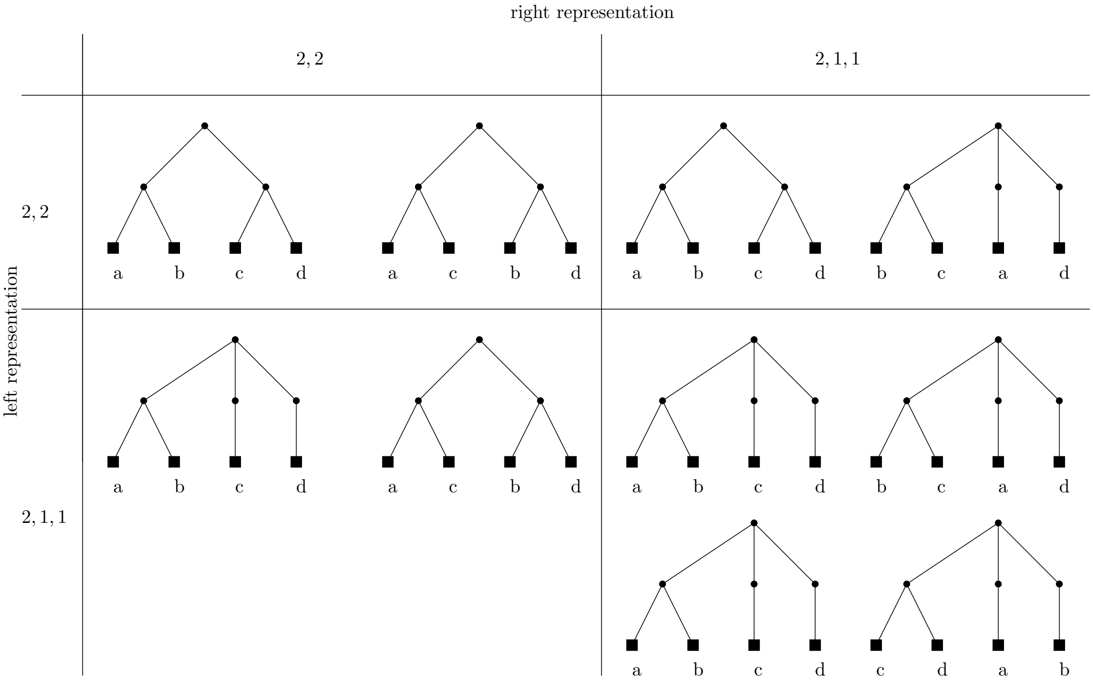


Figure 6.8: There are 5 different ways of how a quartet of leaves can induce a butterfly simultaneously in both trees. They correspond to the following 5 shapes (starting from the upper left corner in the clockwise order): $⧖, \lessgtr, \underline{\angle}, \lessgtr, \gtrless$. Among them, only $\lessgtr$ has matching leaf labels.

Hence, it suffices to calculate $(\#\lessdot), (\#\gtrdot), (\#\underline{<}), (\#\underline{>}), (\#≣)$ and $(\#\lessgtr)$ to obtain $\mathtt{QD}(T_1, T_2)$. In the following technical lemmas we show that all the above values except for $(\#≣)$ can be computed in linear time, whereas $(\#≣)$ is directly related to the number of 4-cycles in G. More precisely, for every shape R we will express its count as $(\#R) = tR + \alpha_R C_4$, where tR is an auxiliary value which can be computed from the considered bipartite graph G in linear time and α_R is a constant. For instance, $(\#\lessdot) = t\lessdot$ means that $(\#\lessdot)$ can be obtained by computing a certain auxiliary value in linear time. The main lemma of this section is that $(\#≣) = t≣ + C_4$ which implies that we can compute $(\#≣)$ from the number of 4-cycles and vice versa in linear time.

6.4.3 Counting Shapes in Simple Bipartite Graphs

Let $m = |E|$, $d(u)$ denotes degree of the node u and $N(u)$ the set of its neighbors. As for now we assume that $E \subseteq V_1 \times V_2$ (recall that the graph is bipartite) and E consists of ordered pairs (u, v). u denotes a node from V_1 and v from V_2. After presenting how to count a shape R we omit calculations for its mirror reflection Я because they are symmetric as it suffices to switch roles of V_1 and V_2.

Lemma 6.4.1. *$(\# \lessdot), (\# \lesseqgtr), (\# \leq)$ and $(\# \lessgtr)$ can be computed from G in linear time.*

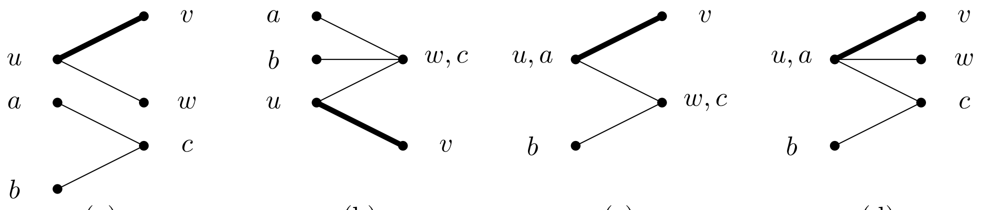
Proof. We compute the first three values directly:

1. $(\# \lessdot) = \sum_{u \in V_1} \binom{d(u)}{4}$
2. $(\# \lesseqgtr) = \sum_{(u,v) \in E} \binom{d(u)-1}{2}(d(v)-1)$
3. $(\# \leq) = \left(\sum_{u \in V_1} \binom{d(u)}{3}(m - d(u))\right) - (\# \lesseqgtr)$

In order to compute $(\# \lessgtr)$ we need two auxiliary values: $(\# >) = \sum_{v \in V_2} \binom{d(v)}{2}$ and $(\# \angle) = \sum_{(u,v) \in E}(d(u)-1)(d(v)-1)$. Then:

$$(\# \lessgtr) = \frac{1}{2}\left(\sum_{(u,v)\in E}(d(u)-1)\left((\# >) - \binom{d(v)}{2}\right) - (\# \gtreqless) - (\# \angle) - 2(\# \lesseqgtr)\right)$$

We derive the above formula in steps. For each edge (u,v) we count shapes in which the edge is one of the sides of $<$ in $\lessgtr$. See Figure 6.9(a) with names of all the nodes in $\lessgtr$. First, we have $(d(u)-1)$ possibilities for the node w which is incident to u, but different from v. Second, we need to account for the $>$ parts of $\lessgtr$ that do not have the corner in node v. There are $\left((\# >) - \binom{d(v)}{2}\right)$ of them and we obtain the term $\sum_{(u,v)\in E}(d(u)-1)\left((\# >) - \binom{d(v)}{2}\right)$.



(a) (b) (c) (d)

Figure 6.9: While computing $(\# \lessgtr)$ we iterate over all choices of edges (u,v). (a) Naming of vertices, (b),(c),(d) subtracted shapes corresponding to respectively $\gtreqless, \Sigma, \lesseqgtr$.

Now we have counted too many shapes, because we did not ensure that the node c is different from w and that both a and b are different from u. To account for the case when $w = c$ we subtract $(\# \gtreqless)$ (when both a and b are different from u, see Figure 6.9(b)) and $(\# \Sigma)$ (when a or b coincide with u, see Figure 6.9(c)). Next, for the case when $c \neq w$ we subtract $2(\# \lesseqgtr)$ for the case when a or b coincide with u, see Figure 6.9(d). This term is multiplied by 2, because we counted it both for the distinguished edge (u,v) and (u,w).

Finally, we need to divide the whole expression by 2, because every shape $\lessgtr$ is counted twice, both for the distinguished edge (u,v) and (u,w). □

To sum up, the above lemma implies that the number of shapes $\lessdot$, $\gtrdot$, $\lesseqgtr$, $\gtreqless$, $\leq$, $\geq$ and $\lessgtr$ in G can be computed in linear time. Recall that we write $(\# \lessgtr) = t_{\lessgtr} + \alpha C_4$ to denote that $(\# \lessgtr)$ equals to αC_4 plus some value $t_{\lessgtr}$ that can be computed from the graph G in linear time and α is a constant. In particular, in the previous lemma we showed that e.g. $(\# \lessdot) = t_{\lessdot}$.

Lemma 6.4.2. *The following equalities hold:*

1. $(\# \lesssim) = t\lesssim - 2C_4$
2. $(\# \substack{<\\<}) = t\substack{<\\<} + C_4$
3. $(\# \nleq) = t\nleq + 4C_4$
4. $(\# \leqq) = t\leqq - 2C_4$

Proof. We reason similarly as in Lemma 6.4.1, that is we distinguish a particular node or edge in G, count all possible choices of other edges and subtract some excess shapes. For every shape R, let $t'R$ be an auxiliary variable that will be used to express tR.

1. Let $t\lesssim = \sum_{v\in V_2}\sum_{x,y\in N(v),x<y}(d(x)-1)(d(y)-1)$
$= \frac{1}{2}\sum_{v\in V_2}\left(\left(\sum_{x\in N(v)}(d(x)-1)\right)^2 - \sum_{x\in N(v)}(d(x)-1)^2\right)$.
Then: $(\# \lesssim) = t\lesssim - 2(\# \bowtie) = t\lesssim - 2C_4$.
We rewrote the expression from the first line to show how to compute $t\lesssim$ in linear time.

2. Let $t'\substack{<\\<} = \sum_{x,y\in V_1,x<y}\binom{d(x)}{2}\binom{d(y)}{2} = \frac{1}{2}\left(\left(\sum_{x\in V_1}\binom{d(x)}{2}\right)^2 - \sum_{x\in V_1}\binom{d(x)}{2}^2\right)$.
Then: $(\# \substack{<\\<}) = t'\substack{<\\<} - (\# \lesssim) - (\# \bowtie) = t'\substack{<\\<} - (t\lesssim - 2C_4) - C_4 = t\substack{<\\<} + C_4$.
The last step briefly indicates: $t\substack{<\\<} := t'\substack{<\\<} - t\lesssim$.

3. Let $t'\nleq = \sum_{(u,v)\in E}(d(u)-1)(d(v)-1)(m-d(u)-d(v)+1)$.
Then: $(\# \nleq) = t'\nleq - 2(\# \lesssim) - 2(\#\gtrsim) - 4(\# \bowtie)$
$= t'\nleq - 2(t\lesssim - 2C_4) - 2(t\gtrsim - 2C_4) - 4C_4 = t\nleq + 4C_4$.
We used the fact that $t\gtrsim$ can be calculated in a symmetric way to $t\lesssim$.

4. Let $t'\leqq = \sum_{(u,v)\in E}(d(u)-1)\binom{m-d(u)-d(v)+1}{2}$.
Then: $(\# \leqq) = \frac{1}{2}\left(t'\leqq - 2(\# \lessgtr) - (\#\gtrapprox) - (\# \nleq) - 2(\# \lesssim) - 4(\# \substack{<\\<})\right)$
$= \frac{1}{2}\left(t'\leqq - 2t\lessgtr - t\gtrapprox - (t\nleq + 4C_4) - 2(t\lesssim - 2C_4) - 4(t\substack{<\\<} + C_4)\right)$
$= t\leqq + \frac{1}{2}(-4C_4 + 4C_4 - 4C_4) = t\leqq - 2C_4$. □

Using the above lemma we are able to relate $(\# \equiv)$ to the number of 4-cycles in the graph G.

Lemma 6.4.3. $(\# \equiv) = t\equiv + C_4$, *where* $t\equiv$ *can be computed from* G *in* $\mathcal{O}(|E|)$ *time.*

Proof. We first compute $(\# \angle) = \sum_{(u,v)\in E}(d(u)-1)(d(v)-1)$
and $(\# \leq) = \frac{1}{2}\left(\sum_{(u,v)\in E}(d(u)-1)(m-d(u)-d(v)+1) - (\# \angle)\right)$. Then:

1. $(\# \equiv) = \frac{1}{3}\big(\sum_{(u,v)\in E}\binom{m-d(u)-d(v)+1}{2} - (\# \leq) - (\# \geq)\big) = t\equiv$

$$\begin{aligned} 2.\ (\#\ ≣) &= \tfrac{1}{4}\Big((m-3)(\#\equiv) - (\#\ \underline{\nless}) - 2(\#\leqq) - 2(\#\geqq)\Big) \\ &= \tfrac{1}{4}\Big((m-3)t≣ - (t\underline{\nless} + 4C_4) - 2(t\leqq - 2C_4) - 2(t\geqq - 2C_4)\Big) \\ &= t≣ + \tfrac{1}{4}(-4C_4 + 4C_4 + 4C_4) = t≣ + C_4 \qquad \square \end{aligned}$$

Finally we use Lemmas 6.4.1 and 6.4.3 to substitute the values in Equation 6.1 obtaining:

$$C_4 = \binom{m}{4} - \Big(t\nleftarrow + t\nrightarrow + t\underline{<} + t\underline{>} + t≣ + t\lessgtr\Big) - \mathtt{QD}(T_1, T_2)$$

As for every shape R, the value tR can be calculated in linear time of the size of the graph G, we obtain the following theorem:

Theorem 6.4.4. *Counting 4-cycles in a graph with m edges can be reduced in linear time to computing the quartet distance between two trees on m leaves.*

Corollary 2.2.11. *There exists no algorithm that can compute the quartet distance between trees on n leaves in $\mathcal{O}(n^{4/3-\varepsilon})$ time unless Conjecture 2 is false.*

6.4.4 Counting Shapes in Multigraphs

In Lemma 6.4.3 we showed that the number of 4-matchings in a bipartite simple graph can be computed from the number of 4-cycles in the graph by adding a number that can be calculated in linear time. Although this was sufficient for the reduction from counting 4-cycles in a graph to computing quartet distance in Theorem 6.4.4, for the reduction in the opposite direction we need a variant of Lemma 6.4.3 for multigraphs. As the calculations for multigraphs are a generalization of those provided in Lemmas 6.4.1, 6.4.2 and 6.4.3, we provide the claim and its proof in this section. The calculations are very similar to the previous ones, but we intentionally separate them to isolate the complexity of multiedges.

Theorem 6.4.5. *In any bipartite multigraph G we have $(\#\ ≣) = t≣ + C_4$, where $t≣$ can be computed from G in $\mathcal{O}(|E|)$ time.*

Proof. Recall that every edge e can appear in the graph multiple times, so we need to take into account its multiplicity $\textsc{mult}(e)$ when counting all shapes containing the edge e. Informally, we need to be more careful while using the binomial coefficient. Let $\left[{S \atop k}\right]$ denote the number of ways of choosing k distinct edges from a set $S \subseteq E$ of multi-edges, that is we cannot take more than one copy of any edge. Recall that every edge appears in E exactly once, but separately we also have a function $\textsc{mult}$ that returns multiplicity of every edge $e \in E$.

Lemma 6.4.6. *$\left[{S \atop k}\right]$ can be computed from S in $\mathcal{O}(|S|k)$ time.*

Proof. Let $S = \{e_1, e_2, \ldots, e_{|S|}\}$ and $S_i = \{e_1, e_2, \ldots, e_i\}, S_0 = \emptyset$. As $\left[{S_i \atop 0}\right] = 1$ and $\left[{S_i \atop j}\right] = \left[{S_{i-1} \atop j}\right] + \left[{S_{i-1} \atop j-1}\right]\textsc{mult}(e_i)$ we can apply dynamic programming to compute $\left[{S_{|S|} \atop k}\right]$. □

For our purposes, k will be always at most 4. We first precompute $\left[{S \atop k}\right]$ for all $k \leq 4$ and the following sets S: $E, \{e\}$ for all edges $e \in E$, and $E(v)$ (set of all edges incident to v) for all nodes v. Then we can combine the stored values to compute $\left[{C \atop k}\right]$ for different sets C using the following lemma:

Lemma 6.4.7. *Let A and B be sets of edges such that $B \subseteq A$ and $\left[{A \atop i}\right], \left[{B \atop i}\right]$ are already computed for all $0 \leq i \leq k$. Then all values $\left[{A \setminus B \atop i}\right]$, for $0 \leq i \leq k$, can be computed in $\mathcal{O}(k^2)$ time.*

Proof. We compute $\left[{A \setminus B \atop j}\right]$ for $j = 0, 1, \ldots, k$ using the following property:

$$\begin{bmatrix} A \setminus B \\ j \end{bmatrix} = \begin{bmatrix} A \\ j \end{bmatrix} - \sum_{i=1}^{j} \begin{bmatrix} B \\ i \end{bmatrix} \begin{bmatrix} A \setminus B \\ j-i \end{bmatrix}$$

as from all the choices of j edges from A we need to subtract the choices that use exactly $i = 1, \ldots, j$ edges from B. □

Now we consider the shapes as in Lemma 6.4.1 and generalize the reasoning to multigraphs. To simplify the notation, by $E - e$ we denote $E \setminus \{e\}$ and write $\text{MULT}(u, v)$ instead of $\text{MULT}(\{u, v\})$.

1. $(\# \lessdot) = \sum_{u \in V_1} \left[{E(u) \atop 4}\right]$
2. $(\# \eqslantgtr) = \sum_{(u,v) \in E} \text{MULT}(u, v) \left[{E(u)-(u,v) \atop 2}\right] \left[{E(v)-(u,v) \atop 1}\right]$
3. $(\# \lneq) = \left(\sum_{u \in V_1} \left[{E(u) \atop 3}\right] \left[{E \setminus E(u) \atop 1}\right]\right) - (\# \eqslantgtr)$
4. Let $(\# >) = \sum_{v \in V_2} \left[{E(v) \atop 2}\right]$. Now, instead of $\angle$, we need to count a shape similar to $\angle$, but instead of choosing a single middle edge e we select an ordered pair of edges (e_1, e_2) where possibly $e_1 = e_2$:

   $(\# \sphericalangle) = \sum_{(u,v) \in E} (\text{MULT}(u, v))^2 \left[{E(u)-(u,v) \atop 1}\right] \left[{E(v)-(u,v) \atop 1}\right]$.
   Then: $(\# \lessgtr) = \frac{1}{2} \left(\sum_{(u,v) \in E} \text{MULT}(u, v) \left[{E(u)-(u,v) \atop 1}\right] \left((\# >) - \left[{E(v) \atop 2}\right]\right) - (\# \gtrless) - (\# \sphericalangle) - 2(\# \eqslantgtr)\right)$

See the more detailed explanation of $(\# \lessgtr)$ in Lemma 6.4.1. Now we consider the shapes as in Lemma 6.4.2. For every shape $R \in \{\lesssim, \overset{<}{<}, \lneq, \leqq\}$ we only show the new value of tR or $t'R$, as all the following calculations are exactly the same as in Lemma 6.4.2.

1. $t\lesssim = \sum_{v \in V_2} \sum_{x,y \in N(v), x<y} \left[{E(x)-(x,v) \atop 1}\right] \left[{E(y)-(y,v) \atop 1}\right]$
   $= \frac{1}{2} \sum_{v \in V_2} \left(\left(\sum_{x \in N(v)} \left[{E(x)-(x,v) \atop 1}\right]\right)^2 - \sum_{x \in N(v)} \left[{E(x)-(x,v) \atop 1}\right]^2\right)$.
2. $t'\overset{<}{<} = \sum_{x,y \in V_1, x<y} \left[{E(x) \atop 2}\right] \left[{E(y) \atop 2}\right] = \frac{1}{2} \left(\left(\sum_{x \in V_1} \left[{E(x) \atop 2}\right]\right)^2 - \sum_{x \in V_1} \left[{E(x) \atop 2}\right]^2\right)$.
3. $t'\lneq = \sum_{(u,v) \in E} \text{MULT}(u, v) \left[{E(u)-(u,v) \atop 1}\right] \left[{E(v)-(u,v) \atop 1}\right] \left[{(E \setminus E(u)) \setminus (E(v)-(u,v)) \atop 1}\right]$.
4. $t'\leqq = \sum_{(u,v) \in E} \text{MULT}(u, v) \left[{E(u)-(u,v) \atop 1}\right] \left[{(E \setminus E(u)) \setminus (E(v)-(u,v))) \atop 2}\right]$.

Now we proceed to the main shape $\equiv$ as in Lemma 6.4.3.

1. Let $(\#\ \overline{\angle}) = \sum_{(u,v)\in E} \text{MULT}(u,v) \left[{E(u)-(u,v) \atop 1}\right] \left[{E(v)-(u,v) \atop 1}\right]$
   and $(\#\ \leq) = \frac{1}{2}\left(\sum_{(u,v)\in E} \text{MULT}(u,v) \left[{E(u)-(u,v) \atop 1}\right] \left[{(E\setminus E(u))\setminus(E(v)-(u,v)) \atop 1}\right] - (\#\ \overline{\angle})\right)$.
   Then: $(\#\ \equiv) = \frac{1}{3}\left(\sum_{(u,v)\in E} \text{MULT}(u,v) \left[{(E\setminus E(u))\setminus(E(v)-(u,v)) \atop 2}\right] - (\#\ \leq) - (\#\ \geq)\right) = t\equiv$.

2. Similarly as in $\not\overline{\angle}$ we consider shapes $\overset{\frown}{<}$, $\overset{\frown}{\underset{}{<}}$ and $\overset{\frown}{=}$ in which an ordered pair of (not necessarily distinct) edges (e_1, e_2) connects one pair of nodes.

   Let $(\#\ \overset{\frown}{<}) = \sum_{(u,v)\in E} \text{MULT}(u,v) \left[{E(u)-(u,v) \atop 1}\right] \left(\left[{E(v)-(u,v) \atop 1}\right]^2 - 2\left[{E(v)-(u,v) \atop 2}\right]\right)$
   and $(\#\ \overset{\frown}{\underset{}{<}}) = \sum_{(u,v)\in E} (\text{MULT}(u,v))^2 \left(s_1 - \left[{E(u) \atop 2}\right]\right) - (\#\ \overset{\frown}{<})$ where $s_1 = \sum_{u\in V_1} \left[{E(u) \atop 2}\right]$
   and $(\#\ \overset{\frown}{=}) = \sum_{(u,v)\in E} (\text{MULT}(u,v))^2 \left[{(E\setminus E(u))\setminus(E(v)-(u,v)) \atop 2}\right] - (\#\ \overset{\frown}{\underset{}{<}}) - (\# \overset{\frown}{\underset{}{>}})$.

   Then: $(\#\ \overline{\equiv}) = \frac{1}{4}\left(\left[{E \atop 1}\right](\#\ \equiv) - (\#\ \underline{\nleq}) - 2(\#\ \leqq) - 2(\# \geqq) - (\#\ \overset{\frown}{=})\right)$
   $= \frac{1}{4}\left(\left[{E \atop 1}\right] \cdot t\equiv - (t\underline{\nleq} + 4C_4) - 2(t\leqq - 2C_4) - 2(t\geqq - 2C_4) - t\overset{\frown}{=}\right)$
   $= t\overline{\equiv} + \frac{1}{4}(-4C_4 + 4C_4 + 4C_4) = t\overline{\equiv} + C_4$.

This concludes the proof of Theorem 6.4.5. □

6.5 From Quartet Distance to Counting 4-Cycles

In Section 6.4 we showed that computing the quartet distance is at least as hard as counting 4-cycles. Now we will show how to use state-of-the-art algorithm for counting 4-cycles to compute the quartet distance faster.

6.5.1 High-Level Overview

As in Section 6.4, we will count quartets of leaves related by the same topology in both trees. Recall that there are two possible topologies: resolved quartet (butterfly) and unresolved quartet (star). We count shared resolved quartets using $\mathcal{O}(n \log n)$ algorithm of Brodal et al. [BFM+13] (value A computed in Section 7.3 there).

For counting shared unresolved quartets (stars), we develop a new algorithm which reduces the original question to counting 4-cycles in many different multigraphs. To provide an intuition, we first describe a slow approach. Every star has a central node, so we iterate over all central nodes $c_1 \in T_1$ and $c_2 \in T_2$. Then we need to count quartets of leaves such that they are in different subtrees connected to c_1 and c_2. Observe that this corresponds to the number of matchings of size 4 ($\overline{\equiv}$) in the multigraph in which left (respectively right) nodes correspond to subtrees connected to node c_1 (respectively c_2), and the multiplicity of an edge (a, b) is the number of common leaves in the a-th subtree connected to c_1 and the b-th subtree connected to c_2, see Figure 6.10. By Theorem 6.4.5, we can count matchings of size 4 in multigraphs by counting 4-cycles in multigraphs.

Hence we reduced computing quartet distance to $\mathcal{O}(n^2)$ black-box calls to counting 4-cycles in a possibly large multigraph which, by Theorem 2.2.6, can be done with $\mathcal{O}(\log^4 n)$ black-box

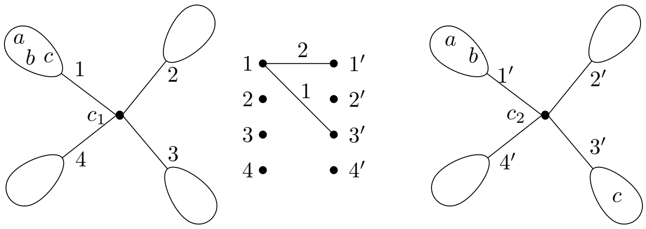


Figure 6.10: Construction of the bipartite multigraph for two central nodes $c_1 \in T_1$ and $c_2 \in T_2$.

calls to the procedure counting 4-cycles in a simple graph. To obtain a faster algorithm for computing quartet distance we need to decrease both the number of black-box calls and have some control on the total size of the constructed multigraphs. Later in this section we design a divide and conquer approach based on top tree decomposition that, combined with the state-of-the-art algorithm for counting 4-cycles, allows us to improve the state-of-the-art algorithm for quartet distance:

Theorem 6.5.1. *For any $\gamma \geq 1$, an algorithm for counting 4-cycles in a simple graph with m edges in $\mathcal{O}(m^\gamma)$ time implies an algorithm for computing the quartet distance between trees on n leaves in $\tilde{\mathcal{O}}(n^\gamma)$ time.*

As discussed before, the starting point is that we only need to count quartets of leaves that induce stars in both trees, which we call shared stars. We group the stars by their central nodes and then count quartets using the procedure for counting 4-cycles applied to many small bipartite multigraphs. However, this approach is too slow, because there are $\Theta(n^2)$ pairs of central nodes to consider. To bypass this difficulty, we will consider some of the pairs of central nodes explicitly (there will be $\mathcal{O}(n \log^2 n)$ of such explicitly considered pairs) and then process the remaining ones aggregately in bigger groups.

6.5.2 Top Trees and Representatives

First we introduce the notion of top trees and representatives that will be used in our algorithm.

Top trees. We root both trees at arbitrarily chosen leaves and choose an arbitrary left-to-right ordering of children of every node. Then we apply a hierarchical decomposition based on top trees introduced by Alstrup et al. [AHdLT05] and then extended by Bille et al. [BGLW15]. In the following paragraphs we provide the precise definition of top tree decomposition based on the condensed presentation from Bille et al. [BGLW15] using the notation from there. It can be omitted if the reader is already familiar with the approach and then they can proceed directly to Theorem 6.5.2.

Let T be a (rooted) tree on n nodes. Every node has a label and node v has $c(v)$ children $v_1, \ldots, v_{c(v)}$ ordered from left to right. Let $T(v)$ denote the subtree of v, including v and $N(T)$ denote the set of nodes from a tree T. We distinguish some subtrees of tree T as *clusters*. Every cluster has a top boundary node and possibly a bottom boundary node and there are two types of clusters.

First, for a node v and $1 \leq s \leq r \leq c(v)$, we define a cluster $T(v,s,r)$ induced by a node v and a contiguous range of subtrees of children of v starting from the s-th and ending at the r-th, that is: $N(T(v,s,r)) = \{v\} \cup \bigcup_{i\in[s,r]} N(T(v_i))$. Such a cluster has top boundary node v and no bottom boundary node.

Second, for a node v, $1 \leq s \leq r \leq c(v)$ and a node $u \neq v$ from $N(T(v,s,r))$ we define a cluster $T(v,s,r) \setminus T(u) \cup \{u\}$ to be the subtree induced by nodes from $N(T(v,s,r)) \setminus N(T(u)) \cup \{u\}$, which has edges from $T(v,s,r)$ excluding the edges from $T(u)$. Such a cluster has top boundary node v and bottom boundary node u.

If edge-disjoint clusters A and B have exactly one common boundary node and $C = A \cup B$ is a cluster, then A and B can be merged into C. Then one of the top boundary nodes of A and B becomes the top boundary node of C and possibly one of the bottom boundary nodes of A or B (if exists) becomes the bottom boundary node of C. See Figure 6.11 (Figure 2 in [BGLW15]) with all five possible ways of merging two clusters. We call the common boundary node the *merged boundary node* of cluster C. By the properties of clusters, we cannot always merge two adjacent clusters, i.e. merges of type (a) and (b) require that the merged boundary node does not belong to any other cluster than the merged two and in types (c),(d) and (e) at least one of the merged clusters does not have the bottom boundary node.

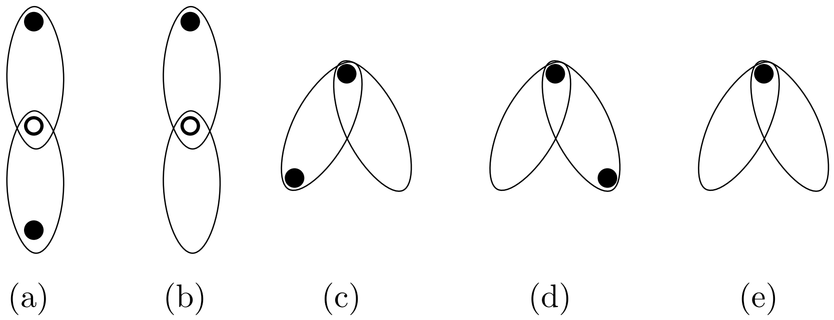


Figure 6.11: (Figure 2 in [BGLW15]) All five types of merging a cluster. Full circles denote boundary nodes of the resulting cluster and empty circle denotes the common boundary node of the merged clusters which is not boundary for the resulting cluster.

A top tree of T is an ordered and labeled binary tree describing a hierarchical decomposition of T into clusters with the above properties. The label of a cluster C with children clusters A and B denotes the type of merge, how the clusters A and B form C. There are $\mathcal{O}(n)$ clusters describing T, as we start with $n-1$ clusters for each edge and every merge decreases their number by one. Intuitively, there are many top trees satisfying the above properties but our goal is to obtain a tree with small height:

Theorem 6.5.2 (Corollary 1 from [BGLW15]). *Given a top tree on n nodes, we can create its top tree of height $\mathcal{O}(\log n)$ in $\mathcal{O}(n)$ time.*

Let $\mathcal{T}_i$ be the top tree of height $\mathcal{O}(\log n)$ representing T_i that can be found as mentioned in Theorem 6.5.2. Let a relevant pair of clusters (C_1, C_2) be a pair of clusters $C_1 \in \mathcal{T}_1$ and $C_2 \in \mathcal{T}_2$

that have at least one common leaf in T_1 and T_2. From the properties of top tree decomposition we have:

Lemma 6.5.3. *The following properties hold:*

(i) Every leaf in T_i is in $\mathcal{O}(\log n)$ distinct clusters $C \in \mathcal{T}_i$.

(ii) There are $\mathcal{O}(n \log^2 n)$ relevant pairs of clusters.

(iii) The total number of common leaves over all pairs of relevant clusters is $\mathcal{O}(n \log^2 n)$.

Proof. We show each of the properties separately:

(i) Every leaf ℓ of T_i is in exactly one two-node cluster C_i^ℓ of $\mathcal{T}_i$. All larger clusters in $\mathcal{T}_i$ that contain ℓ are ancestors of C_i^ℓ, so there are $\mathcal{O}(\log n)$ of them by Theorem 6.5.2.

(ii) We mark pairs of clusters relevant as follows. For every leaf ℓ of T_1 and T_2 we mark relevant all pairs of clusters of $\mathcal{T}_1$ and $\mathcal{T}_2$ that contain ℓ. By (i) there are $\mathcal{O}(\log n)$ distinct clusters containing ℓ in each of the trees, so there are $\mathcal{O}(\log^2 n)$ pairs of clusters marked by ℓ and hence $\mathcal{O}(n \log^2 n)$ in total.

(iii) With the procedure described in (ii), in $\mathcal{O}(n \log^2 n)$ steps we processed all common leaves of all pairs of relevant clusters. Every marking contributes 1 to the total number. □

Recall that our aim is to count all shared stars in T_1 and T_2. Our algorithm will process each relevant pair of clusters and count some particular stars for each such pair. Now, for every shared star we define which relevant pair of clusters it contributes to.

Representatives. Consider a non-leaf node $u \in T_i$. We define $R_i(u)$, the representative cluster of u in T_i, as the smallest cluster that contains u in which u is not a boundary node. This is always well-defined as T_i is rooted in a leaf. Note that $R_i(u)$ is the lowest common ancestor (in the top tree $\mathcal{T}_i$ representing T_i) of all clusters that have u as a boundary node. As merges (c),(d) and (e) (all types are in Figure 6.11) preserve boundary nodes from one of the smaller clusters we have:

Fact 6.5.4. *For every non-leaf node $u \in T_i$, cluster $R_i(u)$ was formed by either (a) or (b)-type merge and u is the merged boundary node of $R_i(u)$.*

Consider a shared star s on leaves $L = \{a, b, c, d\}$ and central nodes $c_1 \in T_1$ and $c_2 \in T_2$. Let $R_1(c_1)$ and $R_2(c_2)$ be the representative clusters of c_1 and c_2. We slightly abuse the notation and write $R_i(s) := R_i(c_i)$ identifying a star with its central node in the corresponding tree T_i. As s is a shared star, the subtree of T_i induced by leaves from L is a star in T_i for $i \in \{1, 2\}$, so each of the clusters $R_i(s)$ contains at least 2 leaves from L. We say that a star s is of type I if $R_1(s)$ and $R_2(s)$ have at least one common leaf from L, otherwise s is of type II, see Figure 6.12 with an example. In order to distinguish the sets of leaves, we draw stars of type II on leaves

$L' = \{x, y, z, t\}$. We intentionally draw the clusters as if the trees were unrooted and do not specify the type of merge, because the relation between the clusters (up/down, left/right) is irrelevant in this case. Notice that the configuration described in the bottom row of Figure 6.12 is the only one possible for a star of type II. More precisely, in a star s of type II each of the clusters $R_i(s)$ contains exactly two leaves of L' and none of the leaves of L' is both in $R_1(s)$ and $R_2(s)$. For stars of type I, there are more possibilities as both $R_1(s)$ and $R_2(s)$ might have between one and four leaves from L.

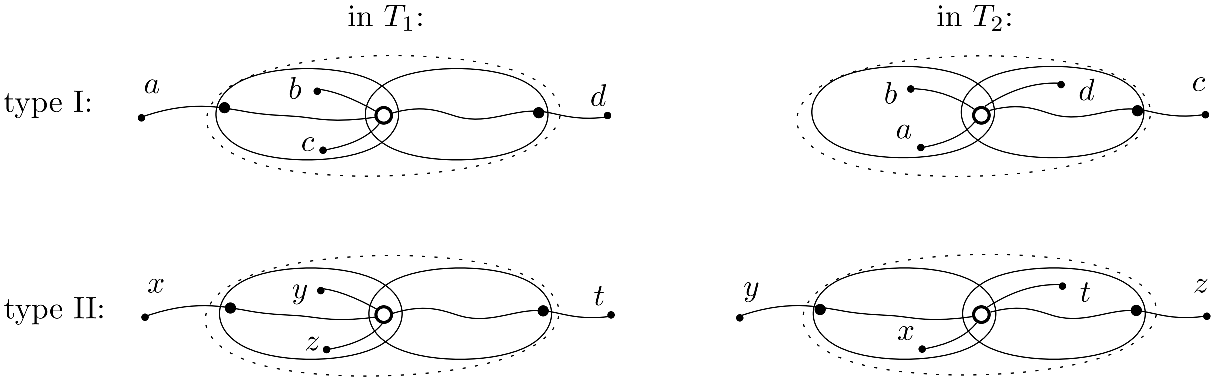


Figure 6.12: An example of a star of type I induced by leaves a, b, c, d (top panel) and a star of type II induced by leaves x, y, z, t (bottom panel). For each star we mark the representative clusters with dotted lines and the central node with an empty circle.

Proposition 6.5.5. *If there exists an $i \in \{1, 2\}$ such that $R_i(s)$ contains at least 3 leaves of a shared star s, then the star s is of type I.*

Proof. Recall that $R_1(s)$ and $R_2(s)$ contain at least 2 leaves of L. As there are 4 leaves in L in total, the claim follows. □

We define that a star s of type I contributes to the pair of clusters $(R_1(s), R_2(s))$ and s will be counted while considering this pair. By definition, in this case $(R_1(s), R_2(s))$ is a relevant pair of clusters and hence will be considered explicitly by our algorithm. However, for a star s of type II, $(R_1(s), R_2(s))$ is not necessarily a relevant pair of clusters. For a star s of type II, let $R'_i(s)$ be the smallest cluster of $\mathcal{T}_i$ containing at least 3 leaves of s.

Lemma 6.5.6. *For every star s of type II, $R'_i(s)$ is uniquely defined and contains exactly 3 leaves common with s.*

Proof. Consider the cluster $R_i(s)$. As s is of type II, two leaves of s are outside of $R_i(s)$, at two opposite sides. Observe that every time we merge $R_i(s)$ with another cluster, we extend it from the side of only one boundary node. Hence we cannot simultaneously add both the outside leaves to the cluster in a single merge. □

We define that a star s of type II contributes to the pair of clusters $(R'_1(s), R'_2(s))$. Then we have:

Proposition 6.5.7. *For every star s of type II, the pair $(R'_1(s), R'_2(s))$ is a relevant pair of clusters.*

Proof. In total there are four leaves of s. As for $i \in \{1, 2\}$, $R'_i(s)$ has three leaves, there is at least one common leaf of $R'_1(s)$ and $R'_2(s)$, so $(R'_1(s), R'_2(s))$ is a relevant pair of clusters. □

Hence $(R'_1(s), R'_2(s))$ will be considered explicitly by our algorithm. Notice that this pair of clusters has some additional properties presented in Figure 6.13. Describing them in more detail, consider a cluster $R'_1(s)$ as depicted on the left panel of Figure 6.13. Suppose $R'_1(s)$ is the result of merging clusters X_1 and Y_1. We call clusters X_1 and Y_1 subclusters, to distinguish them from the main cluster $R'_1(s)$. By definition, $R'_1(s)$ is the smallest cluster of $\mathcal{T}_1$ containing at least 3 leaves of s, so let X_1 contain one leaf of s and Y_1 contains two. Note that the fourth leaf of s, outside $R'_1(s)$ has to be connected to the boundary node of $R'_1(s)$ from the side of Y_1. In the left panel of Figure 6.13, X_1 contains a single leaf of s, Y_1 two leaves and the fourth leaf of s is outside $R'_1(s)$, from the side of Y_1. Moreover, the two leaves y, z of Y_1 are connected to a single node on the path between two boundary nodes of Y_1, but with different edges. In this situation, the type of merge (recall Figure 6.11) can be arbitrary, not necessarily only (a) or (b) as it was the case for type I.

Lemma 6.5.8. *For every star s of type II, its central node is neither a boundary nor the merged boundary node of $R'(s)$.*

Proof. As described in Lemma 6.5.6, $R'_i(s)$ strictly contains $R_i(s)$. $R_i(s)$ is the smallest cluster containing c_i that does not have c_i as the boundary node, so c_i cannot be a boundary node of $R'_i(s)$. As $R_i(s)$ is the only cluster that has c_i as the merged boundary node but not a boundary node, the claim follows. □

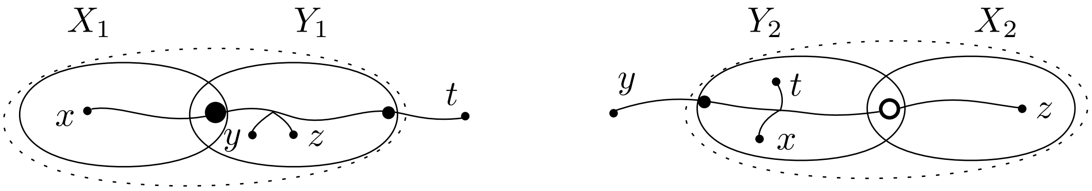


Figure 6.13: Clusters $R'_1(s)$ and $R'_2(s)$ for a star s of type II connecting nodes x, y, z, t. These are the smallest clusters containing three leaves from s. The central node of s is neither a boundary nor the merged boundary node. The type of merge can be arbitrary.

6.5.3 Counting Stars of Type II

In this section we describe how to count stars of type II. From Proposition 6.5.7 it is enough to perform the calculations only for all relevant pairs of clusters and count stars of type II contributing to each of them.

First, we list all relevant pairs of clusters and store their common leaves. This can be done by iterating over all leaves ℓ and then over all clusters containing ℓ in $\mathcal{T}_1$ and then in $\mathcal{T}_2$. Now we consider every relevant pair of clusters (C_1, C_2) and let X_i and Y_i be the two subclusters forming C_i. We shoe how count stars of type II that have the central node in Y_i for $i \in [2]$, as in Figure 6.13. The other cases are symmetric.

Let the leaves of a star s be renamed as depicted in Figure 6.13, that is in tree T_1 there is a single leaf x in subcluster X_1, two leaves y and z in Y_1 and t is outside the considered cluster $C_1 = R'_1(s)$, from the side of subcluster Y_1. Then, as s is of type II, in T_2 we have that leaves x and t are in Y_2. Among leaves y and z, let z be the leaf in X_2 and y outside $C_2 = R'_2(s)$ from the side of subcluster Y_2. Let the spine of a cluster be the path connecting its two boundary nodes. We say that a leaf ℓ connects to the spine S in node u if u is the closest node from S to ℓ.

Now we iterate over all leaves z which are both in Y_1 and X_2 and need to count leaves y such that:

(i) in T_1, y connects to the same node on the spine of Y_1 as z, but with a different edge, and

(ii) in T_2, y is outside C_2, from the side of Y_2.

See Figure 6.14 for the locations of y with respect to z in Y_1 in T_1 that we count (panel (a)) or not (panel (b)) depending on the property (i). Observe that the choice of leaves x and t is independent from y and z, hence we can count pairs x and t separately and then multiply the obtained numbers. Moreover, leaves x and t have the same properties as y and z, by swapping T_1 with T_2 and y, z with x, t. Hence we can count pairs x, t in the same way as the pairs y, z, but for different subclusters, so till the end of this subsection we focus only on counting pairs y and z.

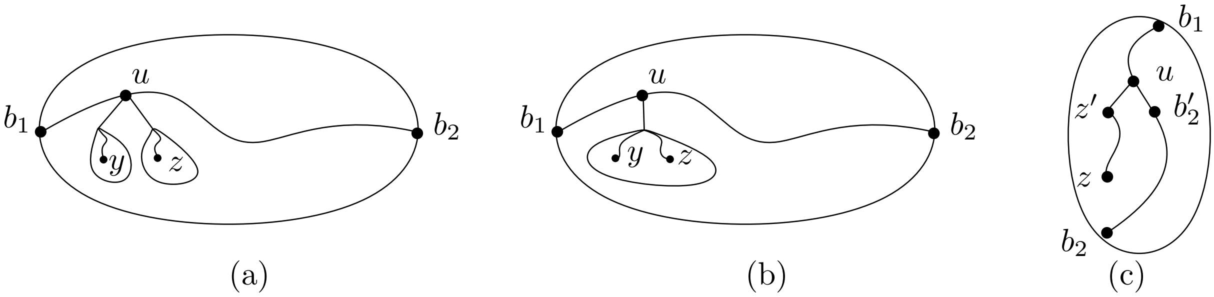


Figure 6.14: For a fixed leaf z we need to count leaves y such that they connect to the spine in the same node as z, but with a different edge. (a) Included, (b) excluded location of y with respect to z. (c) For a fixed leaf z we use nodes u, z' and b'_2 to count all leaves y satisfying both conditions (i) and (ii).

Now we show that both the above conditions (i) and (ii) on y can be phrased in terms of counting points in rectangles, for which we can use existing techniques. First we introduce some notation that will be useful for representing some parts of trees T_i.

Leaves as points. Recall that we rooted tress T_i and fixed an arbitrary left-to-right order of children for every node of trees T_i. Consider the pre-order traversals of the trees. Note that every subtree of T_i corresponds to an interval of the pre-order indices of the nodes. We identify every leaf ℓ with a point $(\textsc{pre}_1(\ell), \textsc{pre}_2(\ell)) \in [n] \times [n]$ where $\textsc{pre}_i(\ell)$ is the index of ℓ in the pre-order traversal of T_i started in its root. That notion is useful for counting shared leaves between a subtree of T_1 and a subtree of T_2, as this is the number of points in the rectangle with its sides being the intervals corresponding to the subtrees. Such queries, in turn, can be answered efficiently using existing techniques for counting points in rectilinear rectangles in the plane, also known as 2-D orthogonal range counting queries, see Lemma 6.2.4. In the following lemma we show that more complex parts of a tree can be represented as a union of a constant number of pairwise disjoint intervals of pre-order indices.

Lemma 6.5.9. *For a rooted tree T and its top tree $\mathcal{T}$, the nodes of:*

(a) clusters of $\mathcal{T}$,

(b) outside parts of clusters of $\mathcal{T}$,

(c) subtrees of unrooted tree T

can be represented as a union of a constant number of pairwise disjoint intervals of pre-order indices of T. Such a representation can be retrieved in $\mathcal{O}(1)$ time after linear time and space preprocessing of T.

Proof. Recall that by definition, there are two types of clusters in $\mathcal{T}$: $T(v, s, r)$ and $T(v, s, r) \setminus T(u) \cup \{u\}$. Observe that nodes from $T(u)$ and $T(v, 1, r)$ can be represented as an interval of pre-order indices of nodes from T. We call such intervals *basic* and precompute them all in $\mathcal{O}(n)$ time and space. Next, we can translate operations on sets of nodes into operations on their corresponding intervals, in particular set union and difference. Now we prove each of the claims from the lemma separately:

(a) For a cluster $T(v, s, r)$ it holds $N(T(v, s, r)) = N(T(v, 1, r)) \setminus N(T(v, 1, s-1)) \cup \{v\}$, so its nodes can be represented with a constant number of basic intervals, similarly for $T(v, s, r) \setminus T(u) \cup \{u\}$.

(b) Consider a cluster C. If $C = T(v, s, r)$, it only has the "above" (towards the root of T) outside part, which is $T \setminus C$. If $C = T(v, s, r) \setminus T(u) \cup \{u\}$, its "above" outside part is $T \setminus T(v, s, r)$ and the outside part "below" is $T(u) \setminus \{u\}$. In all the cases the outside parts can be represented with a constant number of basic intervals.

(c) By the subtree of an unrooted tree T we mean the subtree of a directed edge (u, v) that is the subtree of v when T is rooted in u and the (cyclic) order of neighbors of a node is induced from the original ordering of children of every node in T. If v is farther from the root of T than u we have $\textsc{subtree}((u, v)) = T(v)$, otherwise $\textsc{subtree}((u, v)) = T \setminus T(v)$. In both the cases the subtrees can be represented with a constant number of basic intervals.

In each of these cases, while operating on the sets of intervals we can ensure that all the obtained intervals are pairwise disjoint and then the claim follows. □

Lemma 6.5.10. *After $\mathcal{O}(n \log n)$ time preprocessing of trees T_1 and T_2, for any leaf z we can count leaves y which satisfy both conditions (i) and (ii) in $\mathcal{O}(\log n)$ time.*

Proof. By Lemma 6.5.9, condition (ii) that a leaf is outside a cluster from a specific side translates to the pre-order number of the leaf belonging to the union of a constant number of intervals.

For condition (i), let b_1 and b_2 be the two boundary nodes of Y_1 where b_1 is ancestor of b_2. Note that z connects to the spine $b_1 \cdots b_2$ in the lowest common ancestor (LCA) of z and b_2, call this node u. Let z' be the last node on the path from z to u and b_2' be the last node on the path from b_2 to u, as in Figure 6.14(c). After a linear-time preprocessing of T_2, node u, the LCA of z and b_2, can be found in constant time [SV88]. With a slight modification (called the extended LCA query), we can also find nodes z' and b_2' in the same complexity [GKPS05]. Now we need to count leaves y that are in the subtree of u, but not in the subtree of z' nor b_2'. Each of the three subtrees is represented by an interval of pre-order indices, so we can translate the above condition to the union of a constant number of ranges of pre-order indices.

To sum up, we translated both conditions (i) and (ii) to the union of a constant number of pre-order indices: $\text{PRE}_1(y) \in \mathcal{I}_1^1 \cup \ldots \cup \mathcal{I}_{k_1}^1$ in T_1 and $\text{PRE}_2(y) \in \mathcal{I}_1^2 \cup \ldots \cup \mathcal{I}_{k_2}^2$ in T_2, where the intervals are pairwise disjoint: $\mathcal{I}_a^i \cap \mathcal{I}_b^i = \emptyset$ for $1 \leq a < b \leq k_i, i \in \{1, 2\}$. Now we use orthogonal range query for each of the $k_1 \cdot k_2$ pairs of intervals and count points $(\text{PRE}_1(y), \text{PRE}_2(y)) \in \mathcal{I}_{j_1}^1 \times \mathcal{I}_{j_2}^2$. Summing all the obtained results we get the total number of leaves y satisfying both conditions (i) and (ii) in overall $\mathcal{O}(\log n)$ time. □

To conclude, for every relevant pair of clusters and a pair of their subclusters we iterate over all their common leaves z and count leaves y satisfying both (i) and (ii). As mentioned above, in the same way we count pairs of leaves x and t. By Lemma 6.5.3(iii) and Lemma 6.5.10 counting all stars of type II takes $\mathcal{O}(n \log^2 n \cdot \log n) = \tilde{\mathcal{O}}(n)$ time.

6.5.4 Counting Stars of Type I

Recall that we identify a star s with its central nodes c_1, c_2 and that every star s of type I contributes to the relevant pair of clusters $(R_1(s), R_2(s))$. For this reason it is enough to iterate only over all relevant pairs of clusters and count stars contributing to the considered pair. From Lemma 6.5.3(ii) and (iii) there are $\mathcal{O}(n \log^2 n)$ such pairs and the overall number of common leaves in all of them is also $\mathcal{O}(n \log^2 n)$. Now, for a pair (C_1, C_2) of relevant clusters we show how to count all start of type I with their central nodes $c_i \in T_i$ where c_i is the merged boundary node of cluster C_i for $i \in \{1, 2\}$.

We say that $(R_1(s), R_2(s))$ is the representative pair of star s. Consider a relevant pair of clusters (C_1, C_2). Before we proceed with the general case, we first consider the following special case when a star s with central nodes $c_1 \in T_1$ and $c_2 \in T_2$ has all its four leaves in clusters $C_1 = R_1(s)$ and $C_2 = R_2(s)$, or in other words, s is fully contained in clusters of its representative pair. Recall that in the in Section 6.5.1 we constructed a bipartite multigraph $\mathcal{M}$ in such a way that nodes on the left (respectively right) correspond to subtrees attached to c_1 (respectively c_2) and multiplicity of an edge is the number of common leaves of the corresponding subtrees. See Figure 6.10 for an illustration. We proceed similarly, except that we are only interested in counting stars with all the four leaves in both C_1 and C_2. Therefore we redefine the multiplicity of an edge to be the number of such common leaves in the corresponding subtrees intersected with C_1 and C_2. We are interested only in edges with non-zero multiplicities, so some of the nodes might be isolated and we need to disregard them. In the next paragraph we show how to construct the graph.

Let $\mathcal{L}$ be the set of common leaves of C_1 and C_2. Notice that every such leaf contributes to one multi-edge of $\mathcal{M}$. We iterate over all leaves in $\mathcal{L}$ and update the multiplicities of edges as follows. Given a leaf ℓ, we extract the endpoints of its corresponding edge using extended LCA queries about ℓ and c_i in T_i. Then we look up the corresponding edge in a dictionary and, if it already exists, increase its multiplicity or create a new edge otherwise. This allows us to construct $\mathcal{M}$ in $\mathcal{O}(|\mathcal{L}| \log n)$ time. Clearly, it holds $|E(\mathcal{M})| \leq |\mathcal{L}|$.

It is crucial that the time of construction of $\mathcal{M}$ depends only on $\mathcal{L}$, because now Lemma 6.5.3(iii) guarantees that the overall time of constructing multigraphs $\mathcal{M}$ for all pairs of relevant clusters is $\tilde{\mathcal{O}}(n)$.

Complexity. Before we describe the algorithm for counting all stars of type I, we summarize the complexity of the approach presented so far. Recall that γ is the smallest number such that there exists an algorithm counting 4-cycles in m-edge simple graphs in $\mathcal{O}(m^\gamma)$ time. Combining Theorem 6.4.5 and Theorem 2.2.6 we obtain that we can count matchings of size 4 ($\equiv$) in multigraphs with m edges in $\mathcal{O}(m^\gamma \log^4 m) = \tilde{\mathcal{O}}(m^\gamma)$ time. The algorithm of Vassilevska Williams et al. [WWWY15] runs in time $\mathcal{O}(m^{\frac{4\omega-1}{2\omega+1}}) = \mathcal{O}(m^{2-\frac{3}{2\omega+1}}) = \mathcal{O}(m^{1.478})$, as $\omega < 2.372$ [DWZ23; WXXZ24], so $\gamma < 1.48$.

Let m_i be the number of common leaves in the i-th considered relevant pair of clusters and hence also the bound on the number of edges in the i-th multigraph $\mathcal{M}$. From Lemma 6.5.3(iii) we know that $\sum_i m_i = \mathcal{O}(n \log^2 n)$, where i ranges over all relevant pairs of clusters. So the overall time of counting stars of type I fully contained in their representative pair is:

$$\sum_i \tilde{\mathcal{O}}(m_i^\gamma) = \tilde{\mathcal{O}}\left(\sum_i m_i^\gamma\right) = \tilde{\mathcal{O}}\left(\frac{\sum_i m_i}{n} n^\gamma\right) = \tilde{\mathcal{O}}\left(n^\gamma \log^2 n\right) = \tilde{\mathcal{O}}(n^\gamma) \tag{6.2}$$

where we used the convexity of x^γ (as $\gamma \geq 1$), $m_i \leq n$ and $\sum_i m_i = \tilde{\mathcal{O}}(n)$. To sum up, our algorithm counts all stars fully contained in their representative pairs in $\tilde{\mathcal{O}}(n^\gamma) = \mathcal{O}(n^{1.48})$ time.

Almost all stars of type I. Now we modify the above approach to count all stars of type I, not necessarily fully contained in their representative pairs. The main difficulty is that now the stars can contain leaves outside of $\mathcal{L}$ and we cannot explicitly insert them as edges in the multigraph, as we want to keep the $\tilde{\mathcal{O}}(|\mathcal{L}|)$ running time. We define a modified bipartite multigraph $\mathcal{M}'$ in a similar way as before. For every side $i \in \{1, 2\}$ of the bipartite graph, let a neighbor of c_i be explicit if its subtree contains a leaf from $\mathcal{L}$ or contains an outside part of C_i, otherwise we call it implicit. In $\mathcal{M}'$ we have three types of nodes for every side i of the bipartite graph:

(1) at most two nodes for subtrees connected to c_i that contain an outside part of the cluster C_i,

(2) at most $|\mathcal{L}|$ nodes for subtrees connected to c_i that contain a leaf from $\mathcal{L}$, but do not contain an outside part of C_i,

(3) one node representing all subtrees attached to the implicit neighbors of c_i.

Thus, every node corresponds to a collection of subtrees of the whole (unrooted) T_i. The multiplicity of an edge is simply the number of all common leaves (not only from C_1 and C_2) of the parts of the trees corresponding to endpoints of the edge. See Figure 6.15 for an example.

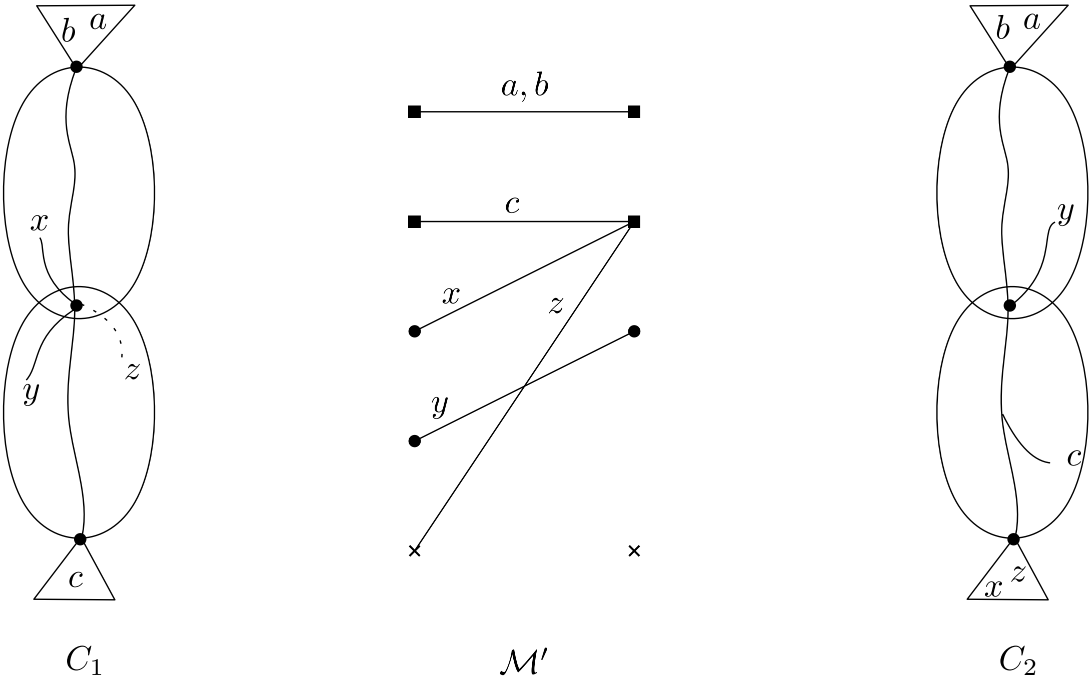


Figure 6.15: Bipartite multigraph $\mathcal{M}'$ with three types of nodes: squares denote nodes of type (1), dots denote type (2) and crosses type (3). We mark leaves of the trees with small letters $a, b, \ldots z$, outside parts of clusters C_1 and C_2 with triangles and boundary nodes (including the merged one) of C_1 and C_2 with dots. Edges of $\mathcal{M}'$ have multiplicities, but instead of numbers we denote precisely which particular leaves contribute to each of the edges. Note that in C_2 leaf c belongs to the subtree of c_2 containing the "below" part of C_2.

We need to show how to construct $\mathcal{M}'$ in $\mathcal{O}(|\mathcal{L}| \log n)$ time. We start with listing all nodes of type (2), similarly as we did for $\mathcal{M}$. We say that an edge is of type (a)-(b) for $a, b \in \{1, 2, 3\}$, when it connects a node of type (a) on the left side of the graph and type (b) on the right. Now we describe how to construct in $\mathcal{O}(|\mathcal{L}| \log n)$ time edges of each type separately.

1. (2)-(2): We obtain all these edges together with their multiplicities by iterating over all leaves from $\mathcal{L}$, as we did for $\mathcal{M}$. We only process the leaves that belong to subtrees of nodes of type (2) in both trees.

2. (1)-(1): In total there are at most 2 nodes of type (1) at each side of the graph. For every pair of such nodes $v_1 \in T_1, v_2 \in T_2$, we calculate intersection of appropriate subtrees (recall Lemma 6.5.9) to obtain multiplicities of the edges.

3. (1)-(2), (2)-(1): There are $\mathcal{O}(|\mathcal{L}|)$ nodes of type (2) and at most two nodes of type (1). For every edge of type (1)-(2) or (2)-(1) we use orthogonal range queries to obtain its multiplicity by intersecting appropriate ranges, so this step runs in $\mathcal{O}(|\mathcal{L}| \log n)$ time.

4. (2)-(3), (3)-(2), (3)-(3): These edges always have multiplicity 0, because otherwise there would exist a leaf from $\mathcal{L}$ in the subtree of an implicit node making the node in fact explicit.

5. (1)-(3), (3)-(1): Recall Lemma 6.5.9 and the fact that there are at most two nodes of type (1) and at most one node of type (3). We can represent union of all subtrees attached to implicit neighbors as union of $\mathcal{O}(|\mathcal{L}|)$ intervals by starting with interval of the considered cluster and subtracting representations of subtrees of at most $|\mathcal{L}| + 2$ explicit nodes. Similarly as in Lemma 6.5.10, in $\mathcal{O}(|\mathcal{L}|)$ orthogonal range queries we can calculate multiplicity of every edge of type (1)-(3) and (3)-(1).

To conclude, we can construct the bipartite multigraph $\mathcal{M}'$ with $\mathcal{O}(|\mathcal{L}|)$ non-zero edges in $\mathcal{O}(|\mathcal{L}| \log n)$ time and then count 4-matchings (𝍣) in $\mathcal{M}'$ in the time complexity calculated in Equation (6.2). However this approach does not count all stars of type I yet.

Missing stars. Recall that on every side of $\mathcal{M}'$ we have one node of type (3) that represents all the subtrees connected to implicit neighbors of c_i. This means that while counting 4-matchings (𝍣) in $\mathcal{M}'$ we allow choosing at most one leaf from the subtrees attached to implicit neighbors of c_i. However, some stars of type I might have more such nodes and they will not be counted with our approach. We call such stars *missing* and analyze their properties in the next paragraphs.

Consider a pair $(C_1, C_2) = (R_1(s), R_2(s))$ of representative clusters for a star s of type I, where c_i is the merged boundary node of C_i and the central node of s.

Proposition 6.5.11. *Leaves of s from the subtrees attached to implicit neighbors of c_i are in the outside parts of C_{3-i}.*

Proof. Suppose the contrary, that there exists an implicit neighbor q of c_i such that its subtree contains a leaf ℓ which is within C_{3-i}. Then ℓ appears both in C_1 and C_2 making the node q explicit. □

Thus s has at most two leaves attached to implicit neighbors of c_i, as they need to be in distinct outside parts of C_{3-i}. We call a side $i \in \{1, 2\}$ *special* if s has exactly two leaves in the subtrees attached to implicit neighbors of c_i. Now we show the key property of missing stars.

Proposition 6.5.12. *Missing star s has exactly one special side.*

Proof. If s has no special side, then s is not a missing star. Suppose that s has two special sides. Then C_1 and C_2 have no common leaf from s, which means that s is of type II, contradiction. □

Note that in terms of the bipartite graph $\mathcal{M}'$, a missing star corresponds to choosing two edges from one node of type (3) that are incident to two different nodes of type (1) and a 2-matching ($=$) between the remaining nodes. Now we show how to count such pairs of edges and the 2-matching ($=$) separately, because they concern different nodes of $\mathcal{M}'$.

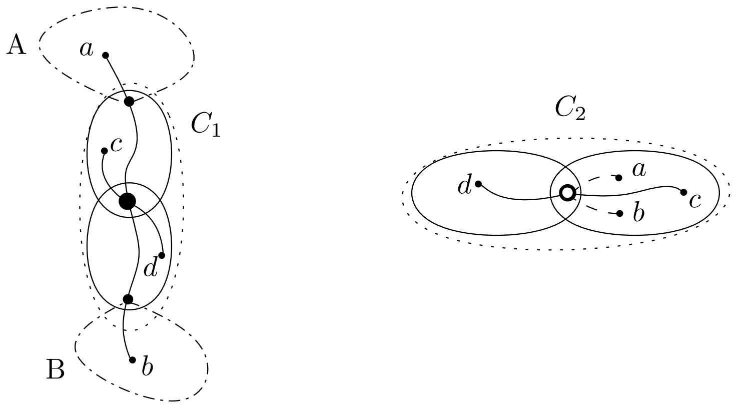


Figure 6.16: Missing star with leaves a and b from outside parts of C_1 and c and d that are both in $C_1 \cup C_2$. The left panel is appropriately rotated with respect to the rooted trees T_i, that is the upper boundary of the cluster is above and the bottom one below the cluster. For C_2 the precise rotation is irrelevant.

We show how to count all missing stars for the case of special side $i = 2$, that is the missing star consists of two leaves a and b from the outside parts A and B of C_1. Recall from Lemma 6.5.4 that representative cluster $R_i(s)$ of a star of type I is formed by (a)- or (b)-type merge (recall Figure 6.11 with all the types of merges). Suppose that B is the outside part below C_1 (in the rooting of T_1 that we use) and A is above C_1, see Figure 6.16. In T_2, leaves a and b are from two different subtrees attached to distinct implicit neighbors of c_2 which are merged together in $\mathcal{M}'$. Leaves c and d are in the subtrees of distinct explicit neighbors of c_1 and c_2. Let $\mathcal{I}$ and $\mathcal{E}$ be the set of respectively the implicit and explicit neighbors of c_2 and α_i and β_i be the number of leaves from A and from B in the subtree of the i-th neighbor of c_2, including both implicit and explicit neighbors. Then the number of choices of leaves $a \in A$ and $b \in B$ that are in the subtrees attached to two distinct implicit neighbors of c_2 is $\sum_{i,j\in\mathcal{I},i\neq j}\alpha_i\beta_j$. Now we need to multiply this number by the number of 2-matchings in $\mathcal{M}'$ with three nodes deleted: two nodes of type (1) from the side of c_1 and the node of type (3) from the side of c_2. Let E'' be the set of edges after removing these nodes. Using the notation and properties from Section 6.4.4 we can count 2-matchings in $\mathcal{O}(|\mathcal{L}|)$ time:

$$(\# =) = \frac{1}{2}\left(\sum_{(u,v)\in E''} \textsc{mult}(u,v)\left[\begin{matrix}(E''\setminus E''(u))\setminus(E''(v)-(u,v))\\ 1\end{matrix}\right]\right)$$

Now we need to compute $\sum_{i,j\in\mathcal{I},i\neq j}\alpha_i\beta_j$. First we show that we can compute every single value of α_i or β_j in $\mathcal{O}(\log n)$ time.

Lemma 6.5.13. *We can calculate values α_i and β_j in $\mathcal{O}(\log n)$ time.*

Proof. From the definition, α_i is the number of leaves that are both in the outside part A of C_1 and in the subtree attached to the i-th neighbor of c_2. By Lemma 6.5.9, both the parts can be represented with a constant number of disjoint intervals of pre-order indices of nodes of the trees. Intersection of the two sums of intervals is the sum of intersections of all pairs of intervals and each of them can be retrieved in $\mathcal{O}(\log n)$ time with the data structure for orthogonal range queries from Lemma 6.2.4. The case of β_j is symmetric and the claim follows. □

To simplify calculating $\sum_{i,j\in\mathcal{I},i\neq j}\alpha_i\beta_j$, note that we can rewrite this term as follows:

$$\sum_{\substack{i,j\in\mathcal{I}\\ i\neq j}}\alpha_i\beta_j=\sum_{\substack{i,j\in\mathcal{E}\cup\mathcal{I}\\ i\neq j}}\alpha_i\beta_j-\sum_{\substack{i,j\in\mathcal{E}\\ i\neq j}}\alpha_i\beta_j-\sum_{i\in\mathcal{E},j\in\mathcal{I}}\alpha_i\beta_j-\sum_{i\in\mathcal{I},j\in\mathcal{E}}\alpha_i\beta_j \tag{6.3}$$

Now we show that all the three subtracted terms can be computed in $\tilde{\mathcal{O}}(|\mathcal{L}|)$ time. As $|A|=\sum_{i\in\mathcal{I}\cup\mathcal{E}}\alpha_i$, we have $\sum_{i\in\mathcal{I},j\in\mathcal{E}}\alpha_i\beta_j=|A|\cdot(\sum_{j\in\mathcal{E}}\beta_j)-(\sum_{i\in\mathcal{E}}\alpha_i)(\sum_{j\in\mathcal{E}}\beta_j)$ and similarly for $\sum_{i\in\mathcal{E},j\in\mathcal{I}}\alpha_i\beta_j$. Next, $\sum_{i,j\in\mathcal{E}i\neq j}\alpha_i\beta_j=(\sum_{i\in\mathcal{E}}\alpha_i)(\sum_{j\in\mathcal{E}}\beta_j)-\sum_{i\in\mathcal{E}}\alpha_i\beta_i$. Note that in $\tilde{\mathcal{O}}(|\mathcal{L}|)$ time we can iterate over all $\mathcal{O}(|\mathcal{L}|)$ explicit neighbors of c_2 and calculate the values of $\sum_{i\in\mathcal{E}}\alpha_i,\sum_{i\in\mathcal{E}}\beta_i$ and $\sum_{i\in\mathcal{E}}\alpha_i\beta_i$. Hence, all the last three terms of (6.3) can be computed in $\tilde{\mathcal{O}}(|\mathcal{L}|)$ time.

To sum up, for the considered pair of relevant clusters with the set $\mathcal{L}$ of common leaves, we reduced in $\tilde{\mathcal{O}}(|\mathcal{L}|)$ time counting missing stars to computing $\sum_{i\neq j}\alpha_i\beta_j$, with α_i,β_j defined above and i,j iterating over all neighbors of c_2, both implicit and explicit. As $\sum_{i\neq j}\alpha_i\beta_j=(\sum_i\alpha_i)\cdot(\sum_i\beta_i)-\sum_i\alpha_i\beta_i=|A|\cdot|B|-\sum_i\alpha_i\beta_i$, we can focus only on computing the last expression. In the next paragraph we restate this subproblem again and describe in detail how to calculate the desired sum efficiently.

Computing $\sum_i\alpha_i\beta_i$. In the previous paragraph we distilled the following subproblem. Consider a relevant pair of clusters (C_1,C_2) with merged boundary nodes $c_1\in C_1,c_2\in C_2$. Let A be the outside part of T_1 above C_1 (in the considered rooting of T_1) and B below C_1. All leaves of T_2 are marked with color $\perp,A$ or B which denotes that the leaf is inside C_1, A or B, respectively and we call such marking the marking with respect to C_1. Our aim is to compute $\sum\alpha_i\beta_i$, where α_i and β_i denote respectively the number of leaves of color A and B in the subtree connected to the i-th neighbor of c_2. We need to calculate the sum for all relevant pairs of clusters efficiently. We proceed off-line, that is we compute and store answers for all the above queries during a single traversal over the tree.

Our algorithm resembles the approach of Brodal et al. in Section 5 of [BFM+13]. We keep a separate data structure supporting the following operations on T_2 in $\mathcal{O}(\log n)$ time:

- Mark(u,c) - mark node $u\in T_2$ with color $c\in\{A,B,\perp\}$,

- $\mathsf{Count}(u)$ - compute $\sum_i \alpha_i\beta_i$ where i ranges over all neighbors of node $u \in T_2$ and α_i and β_i is the number of nodes with color respectively A and B in the subtree of the i-th neighbor of u.

We consider all clusters of top tree $\mathcal{T}_1$ in the order of DFS traversal of $\mathcal{T}_1$ starting at its root. We maintain the following invariant during the traversal:

When entering a cluster $C \in \mathcal{T}_1$, all leaves in T_2 are marked with respect to cluster C.

We start with the cluster representing the whole tree T_1 so all leaves in T_2 are marked with $\perp$. With DFS traversal we visit top tree $\mathcal{T}_1$ top-down and suppose we consider cluster C_1 formed by merging clusters C' and C''. From the invariant, all leaves in T_2 are marked with respect to C_1, so we call $\mathsf{Count}(s_2)$ and store the result for all merged boundary nodes s_2 of clusters C_2 such that (C_1, C_2) is a relevant pair of clusters. Then, while entering cluster C' we need to mark all leaves from C'' with color A or B, depending on the location of C'', recurse and, while exiting, mark leaves from C'' back with $\perp$. Then we proceed similarly for C' while entering C''. From Lemma 6.5.3(i) the total size of all clusters of $\mathcal{T}_i$ is $\mathcal{O}(n \log n)$ so there will be $\mathcal{O}(n \log n)$ updates in total. Finally we show how to implement $\mathsf{Mark}(u, c)$ and $\mathsf{Count}(u)$ operations efficiently.

Lemma 6.5.14. *There exists a data structure supporting* $\mathsf{Mark}(u, c)$ *and* $\mathsf{Count}(u)$ *operations in* $\mathcal{O}(\log n)$ *time.*

Proof. Recall that T_2 is rooted, so we can apply heavy-light decomposition [ST83] to it. The root is called light and every node calls heavy its child with the largest subtree (and the leftmost in case of ties) and all other children light. Then, on every leaf-to-root path there is $\mathcal{O}(\log n)$ light nodes.

For every node v of T_2 we maintain the counter $\sum_\ell \alpha_\ell\beta_\ell$ where ℓ ranges only over the light children of v and counters α_ℓ and β_ℓ for all light children of v. Note that we do not include the ancestor of v and the heavy child of v in the sum. Every update (marking) of a node w first changes the color of w. Then we iterate over its all light ancestors ℓ' and appropriately update counter of their parents. Counters for every such ancestor w are updated in $\mathcal{O}(1)$ time as we update only $\alpha_{\ell'}, \beta_{\ell'}$ and only one summand of $\sum \alpha_\ell\beta_\ell$ changes. From the properties of heavy-light decomposition, every node has $\mathcal{O}(\log n)$ light ancestors, so the update takes $\mathcal{O}(\log n)$ time in total.

To answer $\mathsf{Count}(u)$ query, we use the $\sum_\ell \alpha_\ell\beta_\ell$ counter for u and need to add the values for its parent and the heavy child, if they exist. We obtain both the values using Lemma 6.5.13 in $\mathcal{O}(\log n)$ time. □

To conclude, we can aggregately answer all queries of $\sum_i \alpha_i\beta_i$ for all pairs of relevant clusters in total $\tilde{\mathcal{O}}(n)$ time and then count all the missing stars. Hence, together with the $\tilde{\mathcal{O}}(n)$-time approach for counting almost all shared stars of type I, we can count all shared stars of type I in $\tilde{\mathcal{O}}(n^\gamma)$ time where $\mathcal{O}(m^\gamma)$ is the best complexity of an algorithm counting 4-cycles in a graph with m edges. As we counted all stars of type II in $\tilde{\mathcal{O}}(n)$ time, the whole algorithm counting shared stars in T_1 and T_2 runs in $\tilde{\mathcal{O}}(n^\gamma)$ time. This concludes the proof of Theorem 6.5.1:

Theorem 6.5.1. *For any $\gamma \geq 1$, an algorithm for counting 4-cycles in a simple graph with m edges in $\mathcal{O}(m^\gamma)$ time implies an algorithm for computing the quartet distance between trees on n leaves in $\tilde{\mathcal{O}}(n^\gamma)$ time.*

Now we can plug in the algorithm of Vassilevska Williams et al. [WWWY15] for counting 4-cycles in $\mathcal{O}(m^{2-\frac{3}{2\omega+1}}) = \mathcal{O}(m^\gamma)$ time. Because $\omega < 2.372$ [DWZ23; WXXZ24], we have $\gamma < 1.48$ and obtain an algorithm for computing the quartet distance between two trees on n leaves in $\mathcal{O}(n^{1.48})$ time.

Furthermore, for counting 4-cycles in more dense graphs we can also use the $\mathcal{O}(n^\omega)$ algorithm by Alon et al. [AYZ97] or the $\mathcal{O}(n^{1/\omega}m^{2-2/\omega})$ approach by Eisenbrand and Grandoni [EG03]. As described in more detail in the next section, by appropriately switching between all the three approaches we obtain an algorithm computing the quartet distance in $\tilde{\mathcal{O}}(\min\{n^{1.48}, n^{1.16}d^{0.43}, nd^{0.69}\})$ time.

6.5.5 Dependency on d

In this section we analyze the complexity of the algorithm with respect to the maximum degree d of an internal node. Recall that our algorithm counts 4-cycles in multiple multigraphs. Let n_i and m_i be the number of nodes and edges in the i-th considered multigraph. When bounding the total complexity in (6.2) we only used the fact that $m_i \leq n$. However, in our construction $n_i = \mathcal{O}(d)$, so also $m_i = \mathcal{O}(d^2)$. Hence our algorithm runs in time $\tilde{\mathcal{O}}(\frac{n}{d^2}d^{2\gamma}) = \mathcal{O}(nd^{0.96})$ as $\gamma < 1.48$ [WWWY15].

For more dense graphs it is more desirable to use the algorithm by Alon et al. [AYZ97] that runs in $\mathcal{O}(n^\omega)$ time or by Eisenbrand and Grandoni [EG03] that runs in $\mathcal{O}(n^{1/\omega}m^{2-2/\omega})$ where $\omega < 2.372$ [DWZ23; WXXZ24]. By choosing the most efficient among the three algorithms for each of the subproblems, we decrease the complexity of our algorithm to $(\star) := \tilde{\mathcal{O}}\left(\sum_i \min\left\{n_i^\omega, m_i^\gamma, n_i^{1/\omega}m_i^{2-2/\omega}\right\}\right)$, where $\sum_i n_i = \tilde{\mathcal{O}}(n)$, $\sum_i m_i = \tilde{\mathcal{O}}(n)$ and for every i it holds that $n_i = \mathcal{O}(d)$ and $m_i \leq \min\{n_i^2, n\}$. As we disregard isolated nodes in the graphs, we have $m_i \geq n_i/2$. To simplify calculations we set $m_i \geq n_i$ increasing the total complexity at most by a constant factor. Let $d' = \mathcal{O}(d)$ be the smallest power of 2 satisfying $n_i \leq d'$ for all possible i.

Bounding the above sum $(\star)$ is not immediate, so we divide its terms into $\mathcal{O}(\log^2 n)$ groups identified by a pair (k, ℓ) of parameters such that $n_i \in (2^{k-1}, 2^k]$ and $m_i \in (2^{\ell-1}, 2^\ell]$. As $n_i \leq m_i$ we have $k \leq \ell$ and observe that there are at most $\tilde{\mathcal{O}}(n/2^\ell)$ terms in every group (k, ℓ) due to the bound on the total number of edges. Then we have:

$$\begin{aligned}
(\star) &\leq \sum_{k=0}^{\log d'} \sum_{\ell=k}^{\min\{2k,\log n\}} \tilde{\mathcal{O}}\left(\frac{n}{2^\ell}\min\left\{2^{k\omega}, 2^{\ell\gamma}, 2^{k/\omega+\ell(2-2/\omega)}\right\}\right) && \text{as } \min\{2k,\log n\} \leq 2k \\
&\leq \tilde{\mathcal{O}}\left(n\sum_{k=0}^{\log d'}\sum_{\ell=k}^{2k}\min\left\{2^{k\omega-\ell}, 2^{\ell(\gamma-1)}, 2^{k/\omega+\ell(1-2/\omega)}\right\}\right) && \text{let } \ell = k\cdot\alpha \text{ for } \alpha\in[1,2] \\
&\leq \tilde{\mathcal{O}}\left(n\sum_{k=0}^{\log d'}\log d' \max_{\alpha\in[1,2]} 2^{k\min\{\omega-\alpha,\alpha(\gamma-1),1/\omega+\alpha(1-2/\omega)\}}\right) && \max \text{ is obtained for } \alpha = \frac{\omega+1}{2} \\
&\leq \tilde{\mathcal{O}}\left(n\sum_{k=0}^{\log d'} 2^{k\frac{\omega-1}{2}}\right) = \tilde{\mathcal{O}}\left(nd'^{0.69}\right) = \tilde{\mathcal{O}}\left(nd^{0.69}\right)
\end{aligned}$$

In the second step we upper bounded ℓ by $2k$ which is indeed the case when $2\log d' \leq \log n$. However, for $k > \frac{1}{2}\log n$, the inner sum terminates at $\ell = \log n$ which can improve the overall complexity. To analyze that, now we bound the sum $(\star)$ slightly differently than above:

$$\begin{aligned}
(\star) &\leq \sum_{k=0}^{\log d'} \sum_{\ell=k}^{\min\{2k,\log n\}} \tilde{\mathcal{O}}\left(\frac{n}{2^\ell}\min\left\{2^{k\omega}, 2^{\ell\gamma}, 2^{k/\omega+\ell(2-2/\omega)}\right\}\right) \\
&\leq \tilde{\mathcal{O}}\left(n\sum_{k=0}^{\log d'}\sum_{\ell=k}^{\log n}\min\left\{2^{k\omega-\ell}, 2^{\ell(\gamma-1)}, 2^{k/\omega+\ell(1-2/\omega)}\right\}\right) \\
&\leq \tilde{\mathcal{O}}\left(n\log^2 n\cdot \max_{k=0,\dots,\log d'}\ \max_{\ell=k,\dots,\log n}\ \min\left\{2^{k\omega-\ell}, 2^{\ell(\gamma-1)}, 2^{k/\omega+\ell(1-2/\omega)}\right\}\right) \\
&\leq \tilde{\mathcal{O}}\left(n\cdot \max_{\substack{k=0,\dots,\log d'\\ \ell=0,\dots,\log n}} \min\left\{2^{k\omega-\ell}, 2^{\ell(\gamma-1)}, 2^{k/\omega+\ell(1-2/\omega)}\right\}\right) \\
&= \tilde{\mathcal{O}}\left(n\cdot \max_{\ell=0,\dots,\log n} \min\left\{d'^\omega/2^\ell, 2^{\ell(\gamma-1)}, d'^{1/\omega}2^{\ell(1-2/\omega)}\right\}\right)
\end{aligned}$$

where in the last step we used the observation that for every choice of ℓ, the largest value of $\min\left\{2^{k\omega-\ell}, 2^{\ell(\gamma-1)}, 2^{k/\omega+\ell(1-2/\omega)}\right\}$ is obtained for $k = \log d'$. Then, for $d' > n^{\frac{2}{\omega+1}}$ we have $d'^\omega/2^\ell > d'^{1/\omega}2^{\ell(1-2/\omega)}$ for all $\ell \leq \log n$, so among the three elements in the minimum, $d'^\omega/2^\ell$ will never be the smallest. As also $d' = \mathcal{O}(d)$ we have:

$$\begin{aligned}
(\star) &\leq \tilde{\mathcal{O}}\left(n\cdot \max_{\ell=0,\dots,\log n}\min\left\{2^{\ell(\gamma-1)}, d'^{1/\omega}2^{\ell(1-2/\omega)}\right\}\right) && \text{for } d > n^{\frac{2}{\omega+1}} \\
&= \tilde{\mathcal{O}}\left(n\cdot\min\left\{n^{\gamma-1}, d^{1/\omega}n^{1-2/\omega}\right\}\right) = \begin{cases} \tilde{\mathcal{O}}\left(n^\gamma\right), & \text{for } n \geq d > n^{2-(2-\gamma)\omega} \\ \tilde{\mathcal{O}}\left(n^{2-2/\omega}d^{1/\omega}\right), & \text{for } n^{2-(2-\gamma)\omega} \geq d > n^{\frac{2}{\omega+1}} \end{cases}
\end{aligned}$$

To conclude, we have three different complexities of the algorithm based on the relation between d and n. Now we substitute the current value of $\gamma \leq \frac{4\omega-1}{2\omega+1} = 2 - \frac{3}{2\omega+1}$ [WWWY15] and $\omega < 2.372$ [DWZ23; WXXZ24] and obtain the following running time of our algorithm:

$$\begin{aligned}
\tilde{\mathcal{O}}\left(n^\gamma\right) &= \mathcal{O}\left(n^{1.48}\right) && \text{for } n \geq d > n^{2-(2-\gamma)\omega} \\
\tilde{\mathcal{O}}\left(n^{2-2/\omega}d^{1/\omega}\right) &= \mathcal{O}(n^{1.16}d^{0.43}) && \text{for } n^{2-(2-\gamma)\omega} \geq d > n^{\frac{2}{\omega+1}} \\
\tilde{\mathcal{O}}\left(nd^{\frac{\omega-1}{2}}\right) &= \tilde{\mathcal{O}}(nd^{0.69}) && \text{for } n^{\frac{2}{\omega+1}} \geq d.
\end{aligned}$$

This concludes the proof of the main theorem:

Theorem 2.2.12. *There exists an algorithm for computing the quartet distance between two trees on n leaves and all internal nodes having degrees bounded by d in $\tilde{\mathcal{O}}(\min\{n^{1.48}, n^{1.16}d^{0.43}, nd^{0.69}\})$ time.*

Bibliography

[AAAH01] Michael H. Albert, Robert E. L. Aldred, Mike D. Atkinson, and Derek A. Holton, "Algorithms for pattern involvement in permutations," in *12th ISAAC*, ser. Lecture Notes in Computer Science, Springer, 2001, pp. 355–366.

[AB17] Amir Abboud and Greg Bodwin, "The 4/3 additive spanner exponent is tight," *J. ACM*, vol. 64, no. 4, 28:1–28:20, 2017.

[ABF23] Amir Abboud, Karl Bringmann, and Nick Fischer, "Stronger 3-sum lower bounds for approximate distance oracles via additive combinatorics," in *STOC*, ACM, 2023, pp. 391–404.

[ABH+18] Amir Abboud, Arturs Backurs, Thomas Dueholm Hansen, Virginia Vassilevska Williams, and Or Zamir, "Subtree isomorphism revisited," *ACM Trans. Algorithms*, vol. 14, no. 3, 27:1–27:23, 2018.

[ABHS19] Amir Abboud, Karl Bringmann, Danny Hermelin, and Dvir Shabtay, "SETH-based lower bounds for subset sum and bicriteria path," in *30th SODA*, SIAM, 2019, pp. 41–57.

[ABHS22] Amir Abboud, Karl Bringmann, Danny Hermelin, and Dvir Shabtay, "Scheduling lower bounds via AND subset sum," *J. Comput. Syst. Sci.*, vol. 127, pp. 29–40, 2022.

[ABKZ22] Amir Abboud, Karl Bringmann, Seri Khoury, and Or Zamir, "Hardness of approximation in p via short cycle removal: Cycle detection, distance oracles, and beyond," in *STOC*, ACM, 2022, pp. 1487–1500.

[ABW15a] Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams, "If the current clique algorithms are optimal, so is Valiant's parser," in *FOCS*, IEEE Computer Society, 2015, pp. 98–117.

[ABW15b] Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams, "Tight hardness results for LCS and other sequence similarity measures," in *FOCS*, IEEE Computer Society, 2015, pp. 59–78.

[AC05] Nir Ailon and Bernard Chazelle, "Lower bounds for linear degeneracy testing," *J. ACM*, vol. 52, no. 2, pp. 157–171, 2005.

[ACH+98] Esther M. Arkin *et al.*, "On minimum-area hulls," *Algorithmica*, vol. 21, no. 1, pp. 119–136, 1998.

[ACLL14] Amihood Amir, Timothy M. Chan, Moshe Lewenstein, and Noa Lewenstein, "On hardness of jumbled indexing," in *41st ICALP*, 2014, pp. 114–125.

[AD16] Amir Abboud and Søren Dahlgaard, "Popular conjectures as a barrier for dynamic planar graph algorithms," in *FOCS*, IEEE Computer Society, 2016, pp. 477–486.

[ADKF70] Vladimir L. Arlazarov, Yefim A. Dinic, Aleksandr Kronrod, and IgorAleksandrovich Faradžev, "On economical construction of the transitive closure of an oriented graph," vol. 194, pp. 487–488, 1970.

[AEK05] Daniel W. Archambault, William S. Evans, and David G. Kirkpatrick, "Computing the set of all the distant horizons of a terrain," *Int. J. Comput. Geometry Appl.*, vol. 15, no. 6, pp. 547–564, 2005.

[AFK+24] Amir Abboud, Nick Fischer, Zander Kelley, Shachar Lovett, and Raghu Meka, "New graph decompositions and combinatorial boolean matrix multiplication algorithms," in *STOC*, ACM, 2024, pp. 935–943.

[AFKS00] Noga Alon, Eldar Fischer, Michael Krivelevich, and Mario Szegedy, "Efficient testing of large graphs," *Combinatorica*, vol. 20, no. 4, pp. 451–476, 2000.

[AGW15] Amir Abboud, Fabrizio Grandoni, and Virginia Vassilevska Williams, "Subcubic equivalences between graph centrality problems, APSP and diameter," in *SODA*, SIAM, 2015, pp. 1681–1697.

[AH08] Boris Aronov and Sariel Har-Peled, "On approximating the depth and related problems," *SIAM J. Comput.*, vol. 38, no. 3, pp. 899–921, 2008.

[AHdLT05] Stephen Alstrup, Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup, "Maintaining information in fully dynamic trees with top trees," *ACM Trans. Algorithms*, vol. 1, no. 2, pp. 243–264, 2005.

[AHI+01] Manuel Abellanas *et al.*, "Smallest color-spanning objects," in *9th ESA*, 2001, pp. 278–289.

[AKV13] Sergey V. Avgustinovich, Sergey Kitaev, and Alexandr Valyuzhenich, "Avoidance of boxed mesh patterns on permutations," *Discrete Applied Mathematics*, no. 1-2, pp. 43–51, 2013.

[AL13] Amir Abboud and Kevin Lewi, "Exact weight subgraphs and the k-SUM conjecture," in *40th ICALP*, 2013, pp. 1–12.

[ALLV16] Michael H. Albert, Marie-Louise Lackner, Martin Lackner, and Vincent Vatter, "The complexity of pattern matching for 321-avoiding and skew-merged permutations," *Discrete Mathematics & Theoretical Computer Science*, vol. 18, no. 2, 2016.

[ALW14] Amir Abboud, Kevin Lewi, and Ryan Williams, "Losing weight by gaining edges," in *22th ESA*, ser. Lecture Notes in Computer Science, 2014, pp. 1–12.

[ANS16] Noga Alon, Humberto Naves, and Benny Sudakov, "On the maximum quartet distance between phylogenetic trees," *SIAM J. Discrete Math.*, no. 2, pp. 718–735, 2016.

[AR08] Shlomo Ahal and Yuri Rabinovich, "On complexity of the subpattern problem," *SIAM J. Discrete Math.*, no. 2, pp. 629–649, 2008.

[ARW17] Amir Abboud, Aviad Rubinstein, and R. Ryan Williams, "Distributed PCP theorems for hardness of approximation in P," in *FOCS*, IEEE Computer Society, 2017, pp. 25–36.

[AS01] Benjamin L. Allen and Mike Steel, "Subtree transfer operations and their induced metrics on evolutionary trees," *Annals of Combinatorics*, vol. 5, no. 1, pp. 1–15, Jun. 2001.

[ASU86] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman, *Compilers: Principles, Techniques, and Tools* (Addison-Wesley series in computer science / World student series edition). Addison-Wesley, 1986.

[AW14] Amir Abboud and Virginia Vassilevska Williams, "Popular conjectures imply strong lower bounds for dynamic problems," in *55th FOCS*, IEEE Computer Society, 2014, pp. 434–443.

[AWW14] Amir Abboud, Virginia Vassilevska Williams, and Oren Weimann, "Consequences of faster alignment of sequences," in *41st ICALP*, 2014, pp. 39–51.

[AWY18] Amir Abboud, Virginia Vassilevska Williams, and Huacheng Yu, "Matching triangles and basing hardness on an extremely popular conjecture," *SIAM J. Comput.*, no. 3, pp. 1098–1122, 2018.

[AYZ95] Noga Alon, Raphael Yuster, and Uri Zwick, "Color-coding," *J. ACM*, vol. 42, no. 4, pp. 844–856, 1995.

[AYZ97] Noga Alon, Raphael Yuster, and Uri Zwick, "Finding and counting given length cycles," *Algorithmica*, no. 3, pp. 209–223, 1997.

[BBL98] Prosenjit Bose, Jonathan F. Buss, and Anna Lubiw, "Pattern matching for permutations," *Inf. Process. Lett.*, no. 5, pp. 277–283, 1998.

[BC11] Petter Brändén and Anders Claesson, "Mesh patterns and the expansion of permutation statistics as sums of permutation patterns," *Electr. J. Comb.*, no. 2, 2011.

[BCDK10] Mireille Bousquet-Mélou, Anders Claesson, Mark Dukes, and Sergey Kitaev, "(2+2)-free posets, ascent sequences and pattern avoiding permutations," *J. Comb. Theory, Ser. A*, no. 7, pp. 884–909, 2010.

[BCI+19] Luis Barba, Jean Cardinal, John Iacono, Stefan Langerman, Aurélien Ooms, and Noam Solomon, "Subquadratic algorithms for algebraic 3SUM," *Discr. & Comput. Geometry*, vol. 61, no. 4, pp. 698–734, 2019.

[BD14] Wicher Bergsma and Angelos Dassios, "A consistent test of independence based on a sign covariance related to Kendall's tau," *Bernoulli*, no. 2, pp. 1006–1028, 2014.

[BD86] Hans-Jürgen Bandelt and Andreas Dress, "Reconstructing the shape of a tree from observed dissimilarity data," *Adv. Appl. Math.*, vol. 7, no. 3, pp. 309–343, Sep. 1986.

[BDF11] Mukul S. Bansal, Jianrong Dong, and David Fernández-Baca, "Comparing and aggregating partially resolved trees," *Theor. Comput. Sci.*, vol. 412, no. 48, pp. 6634–6652, 2011.

[BDP08] Ilya Baran, Erik D. Demaine, and Mihai Pătraşcu, "Subquadratic algorithms for 3SUM," *Algorithmica*, vol. 50, no. 4, pp. 584–596, 2008.

[BDT16] Arturs Backurs, Nishanth Dikkala, and Christos Tzamos, "Tight hardness results for maximum weight rectangles," in *ICALP*, ser. LIPIcs, vol. 55, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016, 81:1–81:13.

[Beh46] Felix Adalbert Behrend, "On sets of integers which contain no three terms in arithmetical progression," in *Proc. Natl. Acad. Sci. (USA)*, vol. 32(12), 1946, pp. 331–332.

[BFG17] Philip Bille, Finn Fernstrøm, and Inge Li Gørtz, "Tight bounds for top tree compression," in *SPIRE*, ser. Lecture Notes in Computer Science, vol. 10508, Springer, 2017, pp. 97–102.

[BFM+13] Gerth Stølting Brodal, Rolf Fagerberg, Thomas Mailund, Christian N. S. Pedersen, and Andreas Sand, "Efficient algorithms for computing the triplet and quartet distance between trees of arbitrary degree," in *24th SODA*, SIAM, 2013, pp. 1814–1832.

[BFP04] Gerth Stølting Brodal, Rolf Fagerberg, and Christian N. S. Pedersen, "Computing the quartet distance between evolutionary trees in time $O(n \log n)$," *Algorithmica*, vol. 38, no. 2, pp. 377–395, 2004.

[BG00] Vincent Berry and Olivier Gascuel, "Inferring evolutionary trees with strong combinatorial evidence," *Theor. Comput. Sci.*, vol. 240, no. 2, pp. 271–298, 2000.

[BGK03] Peter Buneman, Martin Grohe, and Christoph Koch, "Path queries on compressed XML," in *VLDB*, Morgan Kaufmann, 2003, pp. 141–152.

[BGLW15] Philip Bille, Inge Li Gørtz, Gad M. Landau, and Oren Weimann, "Tree compression with top trees," *Inf. Comput.*, vol. 243, pp. 166–177, 2015.

[BGMW20] Karl Bringmann, Pawel Gawrychowski, Shay Mozes, and Oren Weimann, "Tree edit distance cannot be computed in strongly subcubic time (unless APSP can)," *ACM Trans. Algorithms*, vol. 16, no. 4, 48:1–48:22, 2020.

[BH01] Gill Barequet and Sariel Har-Peled, "Polygon containment and translational min-Hausdorff-distance between segment sets are 3SUM-hard," *Int. J. Comput. Geometry Appl.*, vol. 11, no. 4, pp. 465–474, 2001.

[BHG+21] Thiago Bergamaschi, Monika Henzinger, Maximilian Probst Gutenberg, Virginia Vassilevska Williams, and Nicole Wein, "New techniques and fine-grained hardness for dynamic near-additive spanners," in *SODA*, SIAM, 2021, pp. 1836–1855.

[BI15] Arturs Backurs and Piotr Indyk, "Edit distance cannot be computed in strongly subquadratic time (unless SETH is false)," in *STOC*, ACM, 2015, pp. 51–58.

[BIS17] Arturs Backurs, Piotr Indyk, and Ludwig Schmidt, "On the fine-grained complexity of empirical risk minimization: Kernel methods and neural networks," in *NIPS*, 2017, pp. 4308–4318.

[BJK+99] Vincent Berry, Tao Jiang, Paul E. Kearney, Ming Li, and Todd Wareham, "Quartet cleaning: Improved algorithms and simulations," in *7th ESA*, ser. Lecture Notes in Computer Science, vol. 1643, Springer, 1999, pp. 313–324.

[BK15] Karl Bringmann and Marvin Künnemann, "Quadratic conditional lower bounds for string problems and dynamic time warping," in *FOCS*, IEEE Computer Society, 2015, pp. 79–97.

[BK17] Karl Bringmann and Marvin Künnemann, "Improved approximation for fréchet distance on c-packed curves matching conditional lower bounds," *Int. J. Comput. Geom. Appl.*, vol. 27, no. 1-2, pp. 85–120, 2017.

[BKM19] Benjamin Aram Berendsohn, László Kozma, and Dániel Marx, "Finding and counting permutations via CSPs," in *14th IPEC*, ser. LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, 1:1–1:16.

[BL13] Marie-Louise Bruner and Martin Lackner, "The computational landscape of permutation patterns," *CoRR*, vol. abs/1301.0340, 2013.

[BL16] Marie-Louise Bruner and Martin Lackner, "A fast algorithm for permutation pattern matching based on alternating runs," *Algorithmica*, vol. 75, no. 1, pp. 84–117, 2016.

[BL24] Gal Beniamini and Nir Lavee, "Counting permutation patterns with multidimensional trees," *CoRR*, vol. abs/2407.04971, 2024.

[BLM08] Giorgio Busatto, Markus Lohrey, and Sebastian Maneth, "Efficient memory representation of XML document trees," *Inf. Syst.*, vol. 33, no. 4-5, pp. 456–474, 2008.

[BLMN15] Mireille Bousquet-Mélou, Markus Lohrey, Sebastian Maneth, and Eric Nöth, "XML compression via directed acyclic graphs," *Theory Comput. Syst.*, vol. 57, no. 4, pp. 1322–1371, 2015.

[Bón12] Miklós Bóna, *Combinatorics of Permutations, Second Edition* (Discrete mathematics and its applications). CRC Press, 2012.

[Bri14] Karl Bringmann, "Why walking the dog takes time: Frechet distance has no strongly subquadratic algorithms unless SETH fails," in *FOCS*, IEEE Computer Society, 2014, pp. 661–670.

[BS00] Eric Babson and Einar Steingrimsson, “Generalized permutation patterns and a classification of the mahonian statistics.,” *Séminaire Lotharingien de Combinatoire*, B44b, 18 p.–B44b, 18 p. 2000.

[BS74] John Adrian Bondy and Miklós Simonovits, “Cycles of even length in graphs,” *Journal of Combinatorial Theory, Series B*, no. 2, pp. 97–105, 1974.

[BS94] Antal Balog and Endre Szemerédi, “A statistical theorem of set addition,” *Comb.*, vol. 14, no. 3, pp. 263–268, 1994.

[BTKL00] David Bryant, John Tsang, Paul E. Kearney, and Ming Li, “Computing the quartet distance between evolutionary trees,” in *11th SODA*, ACM/SIAM, 2000, pp. 285–286.

[Bun71] Peter Buneman, “The recovery of trees from measures of dissimilarity,” English, in *Mathematics the the Archeological and Historical Sciences*, United Kingdom: Edinburgh University Press, 1971, pp. 387–395.

[BvKT98] Prosenjit Bose, Marc J. van Kreveld, and Godfried T. Toussaint, “Filling polyhedral molds,” *Computer-Aided Design*, vol. 30, no. 4, pp. 245–254, 1998.

[BW12] Nikhil Bansal and Ryan Williams, “Regularity lemmas and combinatorial algorithms,” *Theory Comput.*, vol. 8, no. 1, pp. 69–94, 2012.

[CCF+05] Jianer Chen *et al.*, “Tight lower bounds for certain parameterized np-hard problems,” *Inf. Comput.*, vol. 201, no. 2, pp. 216–231, 2005.

[CDL+16] Marek Cygan *et al.*, “On problems as hard as CNF-SAT,” *ACM Trans. Algorithms*, vol. 12, no. 3, 41:1–41:24, 2016.

[CEH04] Otfried Cheong, Alon Efrat, and Sariel Har-Peled, “On finding a guard that sees most and a shop that sells most,” in *15th SODA*, 2004, pp. 1098–1107.

[CFL83] Ashok K. Chandra, Merrick L. Furst, and Richard J. Lipton, “Multi-party protocols,” in *15th STOC*, 1983, pp. 94–99.

[CGI+16] Marco L. Carmosino, Jiawei Gao, Russell Impagliazzo, Ivan Mihajlin, Ramamohan Paturi, and Stefan Schneider, “Nondeterministic extensions of the strong exponential time hypothesis and consequences for non-reducibility,” in *ITCS*, ACM, 2016, pp. 261–270.

[CGLS18] Raphaël Clifford, Allan Grønlund, Kasper Green Larsen, and Tatiana Starikovskaya, “Upper and lower bounds for dynamic data structures on strings,” in *STACS*, ser. LIPIcs, vol. 96, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018, 22:1–22:14.

[CH20] Timothy M. Chan and Qizheng He, “Reducing 3SUM to convolution-3SUM,” in *3rd SOSA*, SIAM, 2020, pp. 1–7.

[Cha08] Timothy M. Chan, “All-pairs shortest paths with real weights in $O(n^3/\log n)$ time,” *Algorithmica*, vol. 50, no. 2, pp. 236–243, 2008.

[Cha10] Timothy M. Chan, "More algorithms for all-pairs shortest paths in weighted graphs," *SIAM J. Comput.*, vol. 39, no. 5, pp. 2075–2089, 2010.

[Cha15] Timothy M. Chan, "Speeding up the four russians algorithm by about one more logarithmic factor," in *SODA*, SIAM, 2015, pp. 212–217.

[Cha18] Timothy M. Chan, "More logarithmic-factor speedups for 3SUM, (median, +)-convolution, and some geometric 3SUM-hard problems," in *29th SODA*, 2018, pp. 881–897.

[Cha88] Bernard Chazelle, "A functional approach to data structures and its use in multidimensional searching," *SIAM J. Comput.*, vol. 17, no. 3, pp. 427–462, 1988.

[Cho59] Noam Chomsky, "On certain formal properties of grammars," *Inf. Control.*, vol. 2, no. 2, pp. 137–167, 1959.

[CL15] Timothy M. Chan and Moshe Lewenstein, "Clustered integer 3SUM via additive combinatorics," in *47th STOC*, 2015, pp. 31–40.

[CMP+06] Chris Christiansen, Thomas Mailund, Christian N. S. Pedersen, Martin Randers, and Martin Stig Stissing, "Fast calculation of the quartet distance between trees of arbitrary degrees," *Algorithms for Molecular Biology*, vol. 1, 2006.

[CMPR05] Chris Christiansen, Thomas Mailund, Christian N. S. Pedersen, and Martin Randers, "Computing the quartet distance between trees of arbitrary degree," in *5th WABI*, ser. Lecture Notes in Computer Science, vol. 3692, Springer, 2005, pp. 77–88.

[CP10] Timothy M. Chan and Mihai Pătraşcu, "Counting inversions, offline orthogonal range counting, and related problems," in *21st SODA*, SIAM, 2010, pp. 161–173.

[CPQ96] Douglas E. Critchlow, Dennis K. Pearl, and Chunlin Qian, "The triples distance for rooted bifurcating phylogenetic trees," *Systematic Biology*, vol. 45, no. 3, pp. 323–334, 1996.

[CS70] John Cocke and Jacob T. Schwartz, "Programming languages and their compilers: Preliminary notes (technical report) (2nd revised ed)," *Technical report, CIMS, NYU*, 1970.

[CSC13] Shay B. Cohen, Giorgio Satta, and Michael Collins, "Approximate PCFG parsing using tensor decomposition," in *HLT-NAACL*, The Association for Computational Linguistics, 2013, pp. 487–496.

[CW90] Don Coppersmith and Shmuel Winograd, "Matrix multiplication via arithmetic progressions," *J. Symb. Comput.*, vol. 9, no. 3, pp. 251–280, 1990.

[Dah16] Søren Dahlgaard, "On the hardness of partially dynamic graph problems and connections to diameter," in *ICALP*, ser. LIPIcs, vol. 55, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016, 48:1–48:14.

[dBdGO97] Mark de Berg, Marko de Groot, and Mark H. Overmars, "Perfect binary space partitions," *Comput. Geom.*, vol. 7, pp. 81–91, 1997.

[DEKM98] Richard Durbin, Sean R. Eddy, Anders Krogh, and Graeme J. Mitchison, *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids.* Cambridge University Press, 1998.

[DG19] Bartłomiej Dudek and Paweł Gawrychowski, "Computing quartet distance is equivalent to counting 4-cycles," in *51st STOC*, ACM, 2019, pp. 733–743.

[DG20] Bartłomiej Dudek and Paweł Gawrychowski, "Counting 4-patterns in permutations is equivalent to counting 4-cycles in graphs," in *31st ISAAC*, ser. LIPIcs, vol. 181, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, 23:1–23:18.

[DG24a] Bartłomiej Dudek and Paweł Gawrychowski, "Online context-free recognition in omv time," in *CPM*, ser. LIPIcs, vol. 296, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024, 13:1–13:9.

[DG24b] Bartłomiej Dudek and Paweł Gawrychowski, "Slowing down top trees for better worst-case compression," *Theor. Comput. Sci.*, vol. 1015, p. 114 764, 2024.

[DGS20] Bartłomiej Dudek, Paweł Gawrychowski, and Tatiana Starikovskaya, "All nontrivial variants of 3-ldt are equivalent," in *STOC*, ACM, 2020, pp. 974–981.

[DHM+14] Holger Dell, Thore Husfeldt, Dániel Marx, Nina Taslaman, and Martin Wahlen, "Exponential time complexity of the permanent and the tutte polynomial," *ACM Trans. Algorithms*, vol. 10, no. 4, 21:1–21:32, 2014.

[DKPW20] Lech Duraj, Krzysztof Kleiner, Adam Polak, and Virginia Vassilevska Williams, "Equivalences between triangle and range query problems," in *30th SODA*, SIAM, 2020, pp. 30–47.

[DKS17] Søren Dahlgaard, Mathias Bæk Tejs Knudsen, and Morten Stöckel, "Finding even cycles faster via capped k-walks," in *49th STOC*, ACM, 2017, pp. 112–120.

[Dob75] Annette J. Dobson, "Comparing the shapes of trees," in *Combinatorial Mathematics III*, Anne Penfold Street and Walter Denis Wallis, Eds., Springer Berlin Heidelberg, 1975, pp. 95–100.

[Dob90] Wlodzimierz Dobosiewicz, "A more efficient algorithm for the min-plus multiplication," *International Journal of Computer Mathematics*, vol. 32, no. 1-2, pp. 49–60, 1990.

[DST80] Peter J. Downey, Ravi Sethi, and Robert Endre Tarjan, "Variations on the common subexpression problem," *J. ACM*, vol. 27, no. 4, pp. 758–771, 1980.

[DvM14] Holger Dell and Dieter van Melkebeek, "Satisfiability allows no nontrivial sparsification unless the polynomial-time hierarchy collapses," *J. ACM*, vol. 61, no. 4, 23:1–23:27, 2014.

[DWZ23] Ran Duan, Hongxun Wu, and Renfei Zhou, "Faster matrix multiplication via asymmetric hashing," in *FOCS*, IEEE, 2023, pp. 2129–2138.

[EG03] Friedrich Eisenbrand and Fabrizio Grandoni, "Detecting directed 4-cycles still faster," *Inf. Process. Lett.*, vol. 87, no. 1, pp. 13–15, 2003.

[EHM06] Jeff Erickson, Sariel Har-Peled, and David M. Mount, "On the least median square problem," *Discr. & Comput. Geometry*, vol. 36, no. 4, pp. 593–607, 2006.

[EL21] Chaim Even-Zohar and Calvin Leng, "Counting small permutation patterns," in *SODA*, SIAM, 2021, pp. 2288–2302.

[Elk10] Michael Elkin, "An improved construction of progression-free sets," in *21th SODA*, 2010, pp. 886–905.

[EMM85] George F. Estabrook, F. R. McMorris, and Christopher A. Meacham, "Comparison of undirected phylogenetic trees based on subtrees of four evolutionary units," *Systematic Zoology*, vol. 34, no. 2, pp. 193–200, 1985.

[EN03] Sergi Elizalde and Marc Noy, "Consecutive patterns in permutations," *Adv. Appl. Math.*, no. 1-2, pp. 110–125, 2003.

[Eri95] Jeff Erickson, "Lower bounds for linear satisfiability problems," in *6th SODA*, 1995, pp. 388–395.

[Eri99a] Jeff Erickson, "Finding longest arithmetic progressions," Manuscript, 1999.

[Eri99b] Jeff Erickson, "Lower bounds for linear satisfiability problems," *Chicago J. Theor. Comput. Sci.*, 1999.

[Eri99c] Jeff Erickson, "New lower bounds for convex hull problems in odd dimensions," *SIAM J. Comput.*, vol. 28, no. 4, pp. 1198–1214, 1999.

[ES35] Paul Erdős and George Szekeres, "A combinatorial problem in geometry," *Compositio Mathematica*, pp. 463–470, 1935.

[ES93] Jeff Erickson and Raimund Seidel, "Better lower bounds on detecting affine and spherical degeneracies," in *FOCS*, IEEE Computer Society, 1993, pp. 528–536.

[ES95] Jeff Erickson and Raimund Seidel, "Better lower bounds on detecting affine and spherical degeneracies," *Discret. Comput. Geom.*, vol. 13, pp. 41–57, 1995.

[FGK03] Markus Frick, Martin Grohe, and Christoph Koch, "Query evaluation on compressed trees (extended abstract)," in *LICS*, IEEE Computer Society, 2003, p. 188.

[FGLS14] Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, and Saket Saurabh, "Almost optimal lower bounds for problems parameterized by clique-width," *SIAM J. Comput.*, vol. 43, no. 5, pp. 1541–1563, 2014.

[FKP24] Nick Fischer, Piotr Kaliciak, and Adam Polak, "Deterministic 3sum-hardness," in *ITCS*, ser. LIPIcs, vol. 287, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024, 49:1–49:24.

[FLMM09] Paolo Ferragina, Fabrizio Luccio, Giovanni Manzini, and S. Muthukrishnan, "Compressing and indexing labeled trees, with applications," *J. ACM*, vol. 57, no. 1, 4:1–4:33, 2009.

[Flo62] Robert W. Floyd, "Algorithm 97: Shortest path," *Commun. ACM*, vol. 5, no. 6, p. 345, 1962.

[Fox13] Jacob Fox, “Stanley-Wilf limits are typically exponential,” *CoRR*, vol. abs/1310.8378, 2013.

[Fre17] Ari Freund, “Improved subquadratic 3SUM,” *Algorithmica*, vol. 77, no. 2, pp. 440–458, 2017.

[Fre76] Michael L. Fredman, “New bounds on the complexity of the shortest path problem,” *SIAM J. Comput.*, vol. 5, no. 1, pp. 83–89, 1976.

[Gal69] Hervé Gallaire, “Recognition time of context-free languages by on-line turing machines,” *Inf. Control.*, vol. 15, no. 3, pp. 288–295, 1969.

[GDG09] R. D. Gray, A. J. Drummond, and S. J. Greenhill, “Language phylogenies reveal expansion pulses and pauses in pacific settlement,” *Science*, vol. 323, no. 5913, pp. 479–483, 2009.

[GGH+20] Alexander Golovnev, Siyao Guo, Thibaut Horel, Sunoo Park, and Vinod Vaikuntanathan, “Data structures meet cryptography: 3sum with preprocessing,” in *52nd STOC*, ACM, 2020, pp. 294–307.

[GHR80] Susan L. Graham, Michael A. Harrison, and Walter L. Ruzzo, “An improved context-free recognizer,” *ACM Trans. Program. Lang. Syst.*, vol. 2, no. 3, pp. 415–462, 1980.

[GJ16] Paweł Gawrychowski and Artur Jeż, “LZ77 factorisation of trees,” in *FSTTCS*, ser. LIPIcs, vol. 65, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016, 35:1–35:15.

[GJ21] Pawel Gawrychowski and Wojciech Janczewski, “Conditional lower bounds for variants of dynamic LIS,” *CoRR*, vol. abs/2102.11797, 2021.

[GKLP16] Isaac Goldstein, Tsvi Kopelowitz, Moshe Lewenstein, and Ely Porat, “How hard is it to find (honest) witnesses?” In *24th ESA*, 2016, 45:1–45:16.

[GKPS05] Leszek Gasieniec, Roman M. Kolpakov, Igor Potapov, and Paul Sant, “Real-time traversal in grammar-based compressed files,” in *15th DCC*, IEEE Computer Society, 2005, p. 458.

[GM14] Sylvain Guillemot and Dániel Marx, “Finding small patterns in permutations in linear time,” in *25th SODA*, SIAM, 2014, pp. 82–101.

[GO95] Anka Gajentaan and Mark H. Overmars, “On a class of $O(n^2)$ problems in computational geometry,” *Comput. Geom.*, vol. 5, pp. 165–185, 1995.

[GP18] Allan Grønlund and Seth Pettie, “Threesomes, degenerates, and love triangles,” *J. ACM*, vol. 65, no. 4, 22:1–22:25, 2018.

[GR22] Paweł Gawrychowski and Mateusz Rzepecki, “Faster exponential algorithm for permutation pattern matching,” in *SOSA*, SIAM, 2022, pp. 279–284.

[GS17] Omer Gold and Micha Sharir, “Improved bounds for 3SUM, k-SUM, and linear degeneracy,” in *25th ESA*, 2017, 42:1–42:13.

[Gus97] Dan Gusfield, *Algorithms on Strings, Trees, and Sequences - Computer Science and Computational Biology*. Cambridge University Press, 1997.

[GV09] Sylvain Guillemot and Stéphane Vialette, "Pattern matching for 321-avoiding permutations," in *20th ISAAC*, ser. Lecture Notes in Computer Science, vol. 5878, Springer, 2009, pp. 1064–1073.

[Han04] Yijie Han, "Improved algorithm for all pairs shortest paths," *Inf. Process. Lett.*, vol. 91, no. 5, pp. 245–250, 2004.

[Han08a] Yijie Han, "A note of an $O(n^3/\log n)$ time algorithm for all pairs shortest paths," *Inf. Process. Lett.*, vol. 105, no. 3, pp. 114–116, 2008.

[Han08b] Yijie Han, "An $O(n^3(\log\log n/\log n)^{5/4})$ time algorithm for all pairs shortest path," *Algorithmica*, vol. 51, no. 4, pp. 428–434, 2008.

[Har15] Kevin Hartnett, "For 40 years, computer scientists looked for a solution that doesn't exist," *The Boston Globe*, Aug. 2015, Published : http://www.bostonglobe.com/ideas/2015/08/10/computer-scientists-have-looked-for-solution-that-doesn-exist/tXO0qNRnbKrClfUPmavifK/story.html.

[Har78] M. A. Harrison, *Introduction to Formal Language Theory*, 1st. USA: Addison-Wesley Longman Publishing Co., Inc., 1978.

[HH16] Yair Heller and Ruth Heller, "Computing the Bergsma Dassios sign-covariance," *CoRR*, vol. abs/1605.08732, 2016.

[HHK+16] Ruth Heller, Yair Heller, Shachar Kaufman, Barak Brill, and Malka Gorfine, "Consistent distribution-free K-sample and independence tests for univariate random variables," *J. Mach. Learn. Res.*, 29:1–29:54, 2016.

[HKNS15] Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak, "Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture," in *STOC*, ACM, 2015, pp. 21–30.

[HKZZ19] Thomas Dueholm Hansen, Haim Kaplan, Or Zamir, and Uri Zwick, "Faster k-sat algorithms using biased-ppsz," in *STOC*, ACM, 2019, pp. 578–589.

[HLSW23] Zhiyi Huang, Yaowei Long, Thatchaphol Saranurak, and Benyu Wang, "Tight conditional lower bounds for vertex connectivity problems," in *STOC*, ACM, 2023, pp. 1384–1395.

[Hoe48] Wassily Hoeffding, "A non-parametric test of independence," *The Annals of Mathematical Statistics*, pp. 546–557, 1948.

[HR15] Lorenz Hübschle-Schneider and Rajeev Raman, "Tree compression with top trees revisited," in *SEA*, ser. Lecture Notes in Computer Science, vol. 9125, Springer, 2015, pp. 15–27.

[HT16] Yijie Han and Tadao Takaoka, "An $O(n^3\log\log n/\log^2 n)$ time algorithm for all pairs shortest paths," *J. Discrete Algorithms*, vol. 38-41, pp. 9–19, 2016.

[HW03] Johan Håstad and Avi Wigderson, "Simple analysis of graph tests for linearity and PCP," *Random Struct. Algorithms*, vol. 22, no. 2, pp. 139–160, 2003.

[Iba97] Louis Ibarra, "Finding pattern matchings for permutations," *Inf. Process. Lett.*, vol. 61, no. 6, pp. 293–295, 1997.

[IP01] Russell Impagliazzo and Ramamohan Paturi, "On the complexity of k-SAT," *J. Comput. Syst. Sci.*, vol. 62, no. 2, pp. 367–375, 2001.

[IPZ01] Russell Impagliazzo, Ramamohan Paturi, and Francis Zane, "Which problems have strongly exponential complexity?" *J. Comput. Syst. Sci.*, vol. 63, no. 4, pp. 512–530, 2001.

[Jac89] Guy Jacobson, "Space-efficient static trees and graphs," in *FOCS*, IEEE Computer Society, 1989, pp. 549–554.

[JK17] Vit Jelinek and Jan Kynčl, "Hardness of permutation pattern matching," in *28th SODA*, SIAM, 2017, pp. 378–396.

[JKL98] Tao Jiang, Paul E. Kearney, and Ming Li, "Orchestrating quartets: Approximation and data correction," in *39th FOCS*, IEEE Computer Society, 1998, pp. 416–425.

[JKMS13] Klaus Jansen, Stefan Kratsch, Dániel Marx, and Ildikó Schlotter, "Bin packing with fixed number of bins revisited," *J. Comput. Syst. Sci.*, vol. 79, no. 1, pp. 39–49, 2013.

[JL14] Jesper Jansson and Andrzej Lingas, "Computing the rooted triplet distance between galled trees by counting triangles," *J. Discrete Algorithms*, vol. 25, pp. 66–78, 2014.

[JL16] Artur Jeż and Markus Lohrey, "Approximation of smallest linear tree grammar," *Inf. Comput.*, vol. 251, pp. 215–251, 2016.

[JM09] Dan Jurafsky and James H. Martin, *Speech and language processing: an introduction to natural language processing, computational linguistics, and speech recognition, 2nd Edition* (Prentice Hall series in artificial intelligence). Prentice Hall, Pearson Education International, 2009.

[JMS04] Joseph JáJá, Christian Worm Mortensen, and Qingmin Shi, "Space-efficient and fast algorithms for multidimensional dominance reporting and counting," in *15th ISAAC*, ser. Lecture Notes in Computer Science, vol. 3341, Springer, 2004, pp. 558–568.

[JRS17] Jesper Jansson, Ramesh Rajaby, and Wing-Kin Sung, "An efficient algorithm for the rooted triplet distance between galled trees," in *4th AlCoB*, ser. Lecture Notes in Computer Science, vol. 10252, Springer, 2017, pp. 115–126.

[JV16] Zahra Jafargholi and Emanuele Viola, "3SUM, 3XOR, triangles," *Algorithmica*, vol. 74, no. 1, pp. 326–343, 2016.

[JWMV03] Katherine St. John, Tandy J. Warnow, Bernard M. E. Moret, and Lisa Vawter, "Performance study of phylogenetic methods: (unweighted) quartet methods and neighbor-joining," *J. Algorithms*, vol. 48, no. 1, pp. 173–193, 2003.

[JX22] Ce Jin and Yinzhan Xu, "Tight dynamic problem lower bounds from generalized BMM and omv," in *STOC*, ACM, 2022, pp. 1515–1528.

[JX23] Ce Jin and Yinzhan Xu, "Removing additive structure in 3sum-based reductions," in *STOC*, ACM, 2023, pp. 405–418.

[Kas65] Tadao Kasami, "An efficient recognition and syntax algorithm for context-free language," *Technical Report AFCRL-65-758, Air Force Cambridge Research Lab, Bed-ford, MA.*, 1965.

[Ken38] Maurice G. Kendall, "A new measure of rank correlation," *Biometrika*, pp. 81–93, 1938.

[Kit11] Sergey Kitaev, *Patterns in Permutations and Words* (Monographs in Theoretical Computer Science. An EATCS Series). Springer, 2011.

[KLM19] Daniel M. Kane, Shachar Lovett, and Shay Moran, "Near-optimal linear decision trees for k-SUM and related problems," *J. ACM*, vol. 66, no. 3, 16:1–16:18, 2019.

[KMPS19] Marvin Künnemann, Daniel Moeller, Ramamohan Paturi, and Stefan Schneider, "Subquadratic algorithms for succinct stable matching," *Algorithmica*, vol. 81, no. 7, pp. 2991–3024, 2019.

[Knu68] Donald E. Knuth, *The Art of Computer Programming, Volume I: Fundamental Algorithms.* Addison-Wesley, 1968.

[KP19] Tsvi Kopelowitz and Ely Porat, "The strong 3SUM-indexing conjecture is false," *CoRR*, vol. abs/1907.11206, 2019.

[KPP16] Tsvi Kopelowitz, Seth Pettie, and Ely Porat, "Higher lower bounds from the 3SUM conjecture," in *27th SODA*, 2016, pp. 1272–1287.

[KSSY13] Flip Korn, Barna Saha, Divesh Srivastava, and Shanshan Ying, "On repairing structural problems in semi-structured data," *Proc. VLDB Endow.*, vol. 6, no. 9, pp. 601–612, 2013.

[Lee02] Lillian Lee, "Fast context-free grammar parsing requires fast boolean matrix multiplication," *J. ACM*, vol. 49, no. 1, pp. 1–15, 2002.

[LM06] Markus Lohrey and Sebastian Maneth, "The complexity of tree automata and XPath on grammar-compressed trees," *Theor. Comput. Sci.*, vol. 363, no. 2, pp. 196–210, 2006.

[LMS18] Daniel Lokshtanov, Dániel Marx, and Saket Saurabh, "Slightly superexponential parameterized problems," *SIAM J. Comput.*, vol. 47, no. 3, pp. 675–702, 2018.

[LPW20] Andrea Lincoln, Adam Polak, and Virginia Vassilevska Williams, "Monochromatic triangles, intermediate matrix products, and convolutions," in *ITCS*, ser. LIPIcs, vol. 151, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, 53:1–53:18.

[LR21] Joshua Lau and Angus Ritossa, "Algorithms and hardness for multidimensional range updates and queries," in *ITCS*, ser. LIPIcs, vol. 185, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 35:1–35:20.

[LRS19] Markus Lohrey, Carl Philipp Reh, and Kurt Sieber, "Size-optimal top dag compression," *Inf. Process. Lett.*, vol. 147, pp. 27–31, 2019.

[LW17] Kasper Green Larsen and R. Ryan Williams, "Faster online matrix-vector multiplication," in *SODA*, SIAM, 2017, pp. 2182–2189.

[LWW18] Andrea Lincoln, Virginia Vassilevska Williams, and R. Ryan Williams, "Tight hardness for shortest cycles and paths in sparse graphs," in *29th SODA*, SIAM, 2018, pp. 1236–1252.

[Man17] Pasin Manurangsi, "Almost-polynomial ratio eth-hardness of approximating densest k-subgraph," in *STOC*, ACM, 2017, pp. 954–961.

[Mar07] Dániel Marx, "On the optimality of planar and geometric approximation schemes," in *FOCS*, IEEE Computer Society, 2007, pp. 338–348.

[MS11] Robin A. Moser and Dominik Scheder, "A full derandomization of schöning's k-sat algorithm," in *STOC*, ACM, 2011, pp. 245–252.

[MT04] Adam Marcus and Gábor Tardos, "Excluded permutation matrices and the Stanley-Wilf conjecture," *J. Comb. Theory, Ser. A*, no. 1, pp. 153–160, 2004.

[NKMP11] Jesper Nielsen, Anders K. Kristensen, Thomas Mailund, and Christian N. S. Pedersen, "A sub-cubic time algorithm for computing the quartet distance between two general trees," *Algorithms for Molecular Biology*, vol. 6, p. 15, 2011.

[NWRE05] Luay Nakhleh, Tandy Warnow, Don Ringe, and Steven N. Evans, "A comparison of phylogenetic reconstruction methods on an indo-european dataset," *Transactions of the Philological Society*, vol. 103, no. 2, pp. 171–192, 2005.

[OBr04] Kevin O'Bryant, "A Complete Annotated Bibliography of Work Related to Sidon Sequences," *The Electronic Journal of Combinatorics*, vol. Dynamic Survey 11, 2004.

[Păt10] Mihai Pătraşcu, "Towards polynomial lower bounds for dynamic problems," in *42nd STOC*, 2010, pp. 603–610.

[Pav15] John Pavlus, "A new map traces the limits of computation," *Quanta Magazine*, Sep. 2015, Published online: https://www.quantamagazine.org/edit-distance-reveals-hard-computational-problems-20150929/.

[PK09] Adam Pauls and Dan Klein, "K-best A* parsing," in *ACL/IJCNLP*, The Association for Computer Linguistics, 2009, pp. 958–966.

[PP36] Erdős Paul and Turán Paul, "On some sequences of integers," *Journal of the London Mathematical Society*, vol. s1-11, no. 4, pp. 261–264, 1936.

[PW10] Mihai Pătraşcu and Ryan Williams, "On the possibility of faster SAT algorithms," in *21st SODA*, 2010, pp. 1065–1075.

[RF79] D. F. Robinson and L. R. Foulds, "Comparison of weighted labelled trees," in *Combinatorial Mathematics VI*, A. F. Horadam and W. D. Wallis, Eds., Springer Berlin Heidelberg, 1979, pp. 119–126.

[RF81] D.F. Robinson and L.R. Foulds, "Comparison of phylogenetic trees," *Mathematical Biosciences*, vol. 53, no. 1, pp. 131–147, 1981.

[RRS+21] Jakub Radoszewski, Wojciech Rytter, Juliusz Straszynski, Tomasz Walen, and Wiktor Zuba, "Hardness of detecting abelian and additive square factors in strings," in *ESA*, ser. LIPIcs, vol. 204, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 77:1–77:19.

[RSCJ10] Alexander M. Rush, David A. Sontag, Michael Collins, and Tommi S. Jaakkola, "On dual decomposition and linear programming relaxations for natural language processing," in *EMNLP*, ACL, 2010, pp. 1–11.

[Ruz93] Imre Z. Ruzsa, "Solving a linear equation in a set of integers I," *Acta Arithmetica LXV.3*, 1993.

[RW13] Liam Roditty and Virginia Vassilevska Williams, "Fast approximation algorithms for the diameter and radius of sparse graphs," in *STOC*, ACM, 2013, pp. 515–524.

[Ryt85] Wojciech Rytter, "Fast recognition of pushdown automaton and context-free languages," *Information and Control*, vol. 67, no. 1-3, pp. 12–22, 1985.

[Ryt95] Wojciech Rytter, "Context-free recognition via shortest paths computation: A version of Valiant's algorithm," *Theor. Comput. Sci.*, vol. 143, no. 2, pp. 343–352, 1995.

[RZ11] Liam Roditty and Uri Zwick, "On dynamic shortest paths problems," *Algorithmica*, vol. 61, no. 2, pp. 389–401, 2011.

[SBF+13] Andreas Sand, Gerth Stølting Brodal, Rolf Fagerberg, Christian N. S. Pedersen, and Thomas Mailund, "A practical $O(n \log^2 n)$ time algorithm for computing the triplet distance on binary trees," *BMC Bioinformatics*, vol. 14, no. S-2, S18, 2013.

[SBMN13] Richard Socher, John Bauer, Christopher D. Manning, and Andrew Y. Ng, "Parsing with compositional vector grammars," in *ACL (1)*, The Association for Computer Linguistics, 2013, pp. 455–465.

[Sch21] Dominik Scheder, "PPSZ is better than you think," in *FOCS*, IEEE, 2021, pp. 205–216.

[Sei86] Joel I. Seiferas, "A simplified lower bound for context-free-language recognition," *Inf. Control.*, vol. 69, no. 1-3, pp. 255–260, 1986.

[SEO03] Michael A. Soss, Jeff Erickson, and Mark H. Overmars, "Preprocessing chains for fast dihedral rotations is hard or even impossible," *Comput. Geom.*, vol. 26, no. 3, pp. 235–246, 2003.

[SHJ+13] Andreas Sand *et al.*, "Algorithms for computing the triplet quartet distances for binary general trees," in *Biology*, 2013.

[Sip97] Michael Sipser, *Introduction to the theory of computation.* PWS Publishing Company, 1997.

[SN87] N. Saitou and M. Nei, "The neighbor-joining method: A new method for reconstructing phylogenetic trees.," *Molecular Biology and Evolution*, vol. 4, no. 4, pp. 406–425, 1987.

[SP93] Mike A. Steel and David Penny, "Distributions of tree comparison metrics—some new results," *Systematic Biology*, vol. 42, no. 2, pp. 126–141, 1993.

[SPM+07] Martin Stig Stissing, Christian N. S. Pedersen, Thomas Mailund, Gerth Stølting Brodal, and Rolf Fagerberg, "Computing the quartet distance between evolutionary trees of bounded degree," in *5th APBC*, ser. Advances in Bioinformatics and Computational Biology, vol. 5, Imperial College Press, 2007, pp. 101–110.

[SR10] Sagi Snir and Satish Rao, "Quartets maxcut: A divide and conquer quartets algorithm," *IEEE/ACM Trans. Comput. Biology Bioinform.*, vol. 7, no. 4, pp. 704–718, 2010.

[SS42] Raphaël Salem and Donald C. Spencer, "On sets of integers which contain no three terms in arithmetical progression," in *Proc. Natl. Acad. Sci. (USA)*, vol. 28(12), 1942, pp. 561–563.

[SS85] Rodica Simion and Frank W. Schmidt, "Restricted permutations," *Eur. J. Comb.*, no. 4, pp. 383–406, 1985.

[SS90] Jeanette P. Schmidt and Alan Siegel, "The spatial complexity of oblivious k-probe hash functions," *SIAM J. Comput.*, vol. 19, no. 5, pp. 775–786, 1990.

[ST83] Daniel Dominic Sleator and Robert Endre Tarjan, "A data structure for dynamic trees," *J. Comput. Syst. Sci.*, vol. 26, no. 3, pp. 362–391, 1983.

[Str69] Volker Strassen, "Gaussian elimination is not optimal.," *Numerische Mathematik*, vol. 13, pp. 354–356, 1969.

[SV88] Baruch Schieber and Uzi Vishkin, "On finding lowest common ancestors: Simplification and parallelization," *SIAM J. Comput.*, vol. 17, no. 6, pp. 1253–1262, 1988.

[SvH96] K Strimmer and A von Haeseler, "Quartet puzzling: A quartet maximum-likelihood method for reconstructing tree topologies," *Molecular Biology and Evolution*, vol. 13, no. 7, p. 964, 1996.

[SY12] Sagi Snir and Raphael Yuster, "Reconstructing approximate phylogenetic trees from quartet samples," *SIAM J. Comput.*, vol. 41, no. 6, pp. 1466–1480, 2012.

[Tak04] Tadao Takaoka, "A faster algorithm for the all-pairs shortest path problem and its application," in *COCOON*, ser. Lecture Notes in Computer Science, vol. 3106, Springer, 2004, pp. 278–289.

[Tak05] Tadao Takaoka, "An $O(n^3 \log\log n/\log n)$ time algorithm for the all-pairs shortest path problem," *Inf. Process. Lett.*, vol. 96, no. 5, pp. 155–161, 2005.

[Tak92] Tadao Takaoka, "A new upper bound on the complexity of the all pairs shortest path problem," *Inf. Process. Lett.*, vol. 43, no. 4, pp. 195–199, 1992.

[Val75] Leslie G. Valiant, "General context-free recognition in less than cubic time," *J. Comput. Syst. Sci.*, vol. 10, no. 2, pp. 308–315, 1975.

[Vat15] Vincent Vatter, "Permutation classes," in *Handbook of Enumerative Combinatorics*, Miklós Bóna, Ed., CRC Press, 2015.

[War62] Stephen Warshall, "A theorem on boolean matrices," *J. ACM*, vol. 9, no. 1, pp. 11–12, 1962.

[WDL16] Luca Weihs, Mathias Drton, and Dennis Leung, "Efficient computation of the Bergsma—Dassios sign covariance," *Comput. Stat.*, no. 1, pp. 315–328, 2016.

[WDM18] Luca Weihs, Mathias Drton, and Nicolai Meinshausen, "Symmetric rank covariances: a generalized framework for nonparametric measures of dependence," *Biometrika*, no. 3, pp. 547–562, 2018.

[WF74] Robert A. Wagner and Michael J. Fischer, "The string-to-string correction problem," *J. ACM*, vol. 21, no. 1, pp. 168–173, 1974.

[Wil05] Ryan Williams, "A new algorithm for optimal 2-constraint satisfaction and its implications," *Theor. Comput. Sci.*, vol. 348, no. 2-3, pp. 357–365, 2005.

[Wil07] Ryan Williams, "Matrix-vector multiplication in sub-quadratic time: (some preprocessing required)," in *SODA*, SIAM, 2007, pp. 995–1001.

[Wil15] Virginia Vassilevska Williams, "Hardness of easy problems: Basing hardness on popular conjectures such as the strong exponential time hypothesis (invited talk)," in *10th IPEC*, 2015, pp. 17–29.

[Wil16] Richard Ryan Williams, "Strong ETH breaks with merlin and arthur: Short noninteractive proofs of batch evaluation," in *CCC*, ser. LIPIcs, vol. 50, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016, 2:1–2:17.

[Wil18a] R. Ryan Williams, "Faster all-pairs shortest paths via circuit complexity," *SIAM J. Comput.*, vol. 47, no. 5, pp. 1965–1985, 2018.

[Wil18b] Virginia Vassilevska Williams, "On some fine-grained questions in algorithms and complexity," in *International Congress of Mathematicians (ICM)*, 2018.

[WS78] M.S. Waterman and T.F. Smith, "On the similarity of dendrograms," *Journal of Theoretical Biology*, vol. 73, no. 4, pp. 789–800, 1978.

[WW13] Virginia Vassilevska Williams and Ryan Williams, "Finding, minimizing, and counting weighted subgraphs," *SIAM J. Comput.*, vol. 42, no. 3, pp. 831–854, 2013.

[WW18] Virginia Vassilevska Williams and R. Ryan Williams, "Subcubic equivalences between path, matrix, and triangle problems," *J. ACM*, no. 5, 27:1–27:38, 2018.

[WWMA12] Robert S. Walker, Søren Wichmann, Thomas Mailund, and Curtis J. Atkisson, "Cultural phylogenetics of the Tupi language family in lowland South America," *PLOS ONE*, vol. 7, no. 4, pp. 1–9, Apr. 2012.

[WWWY15] Virginia Vassilevska Williams, Joshua R. Wang, Richard Ryan Williams, and Huacheng Yu, "Finding four-node subgraphs in triangle time," in *26th SODA*, SIAM, 2015, pp. 1671–1680.

[WX20] Virginia Vassilevska Williams and Yinzhan Xu, "Monochromatic triangles, triangle listing and APSP," in *FOCS*, IEEE, 2020, pp. 786–797.

[WXXZ24] Virginia Vassilevska Williams, Yinzhan Xu, Zixuan Xu, and Renfei Zhou, "New bounds for matrix multiplication: From alpha to omega," in *SODA*, SIAM, 2024, pp. 3792–3835.

[Yan70] Takemi Yanagimoto, "On measures of association and a related problem," *Annals of the Institute of Statistical Mathematics*, no. 1, pp. 57–63, 1970.

[You67] Daniel H. Younger, "Recognition and parsing of context-free languages in time n^3," *Inf. Control.*, vol. 10, no. 2, pp. 189–208, 1967.

[YS05] V. Yugandhar and Sanjeev Saxena, "Parallel algorithms for separable permutations," *Discrete Applied Mathematics*, no. 3, pp. 343–364, 2005.

[Yu18] Huacheng Yu, "An improved combinatorial algorithm for boolean matrix multiplication," *Inf. Comput.*, vol. 261, pp. 240–247, 2018.

[YZ97] Raphael Yuster and Uri Zwick, "Finding even cycles even faster," *SIAM J. Discrete Math.*, no. 2, pp. 209–222, 1997.

[Zwi06] Uri Zwick, "A slightly improved sub-cubic algorithm for the all pairs shortest paths problem with real edge lengths," *Algorithmica*, vol. 46, no. 2, pp. 181–192, 2006.