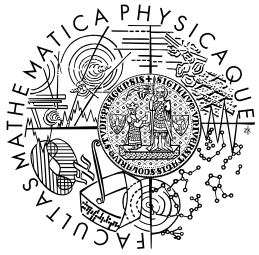


FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

DOCTORAL THESIS

Debarati Das

New Bounds for Combinatorial Problems and Quasi-Gray Codes

Computer Science Institute of Charles University

Supervisor of the doctoral thesis:	Prof. Mgr. Michal Koucký, Ph.D.
Study programme:	Informatics (P1801)
Study branch:	Discrete Models and Algorithms(4I4)

Prague 2019

I declare that I carried out this doctoral thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ..... signature of the author

Dedicated to my parents...

Title: New Bounds for Combinatorial Problems and Quasi-Gray Codes

Author: Debarati Das

Department: Computer Science Institute of Charles University

Supervisor: Prof. Mgr. Michal Koucký, Ph.D., Computer Science Institute of Charles University

Abstract: This thesis consists of two parts. In part I, a group of combinatorial problems pertaining to strings, boolean matrices and graphs is studied. For given two strings x and y, their edit distance is the minimum number of character insertions, deletions and substitutions required to convert x into y. In this thesis we provide an algorithm that computes a constant approximation of edit distance in truly sub-quadratic time. Based on the provided ideas, we construct a separate sub-quadratic time algorithm that can find an occurrence of a pattern P in a given text T while allowing a few edit errors. Afterwards we study the boolean matrix multiplication (BMM) problem where given two boolean matrices, the aim is to find their product over boolean semi-ring. For this problem, we present two combinatorial models and show in these models BMM requires $\Omega(n^3/2^{O(\sqrt{\log n})})$ and $\Omega(n^{7/3}/2^{O(\sqrt{\log n})})$ work respectively. Furthermore, we also give a construction of a sparse sub-graph that preserves the distance between a designated source and any other vertex as long as the total weight increment of all the edges is bounded by some constant.

In part II, we study the efficient construction of quasi-Gray codes. We give a construction of space optimal quasi-Gray codes over odd sized alphabets with read complexity $4\log_m n$. Moreover, this part also presents a construction of quasi-Gray codes of length $2^n - 20n$ over binary alphabet set with read complexity $6 + \log n$.

Keywords: Boolean matrix multiplication, Combinatorial lower bounds, Edit distance, Pattern matching, Graph algorithm, Quasi-Gray code.

Acknowledgements

Firstly, I am immensely grateful to my advisor, Prof. Michal Koucký, for giving me the opportunity to work in the ERC project LBCAD under his supervision. He has introduced me to several fundamental problems in theoretical computer science in past four years. Without any slightest doubt, solving those problems was the most challenging yet cherishable experience in my career. What I admire the most about Michal is his unique way of approaching any problem through proper structuring and pre-assessment of the possible outcomes, to name but a few. I learnt from him not only how to tackle any problem but also the art of perseverance and dedication. It has been a great privilege to be his student. I am thankful to him for his continuous support and encouragement that kept me motivated and confident throughout the journey of my Ph.D. I also want to thank his lovely family for inviting me at their home on several occasions.

Next, I want to acknowledge Prof. Michael Saks (Rutgers, The State University of NJ) with whom a significant part of this thesis has been collaborated with. I consider myself extremely fortunate for having an academic association with him. I thoroughly enjoyed the insightful discussions with him, which was a great learning experience for me. On a similar note, I want to thank my other collaborators - Diptarka Chakraborty, Nitin Saurabh and Elazar Goldenberg. Special thanks go to Diptarka and Nitin for providing me a wonderful company in Prague.

During the course of my Ph.D., I made several academic visits that I wish to acknowledge. I would like to thank Prof. Eric Allender, Dr. Karl Bringmann and Prof. Mikkel Thorup for hosting me at DIMACS (Rutgers), MPII (Saarbrucken) and BARC (Copenhagen), respectively.

I am also grateful to the European Research Council (ERC) under the European Union's Seventh Framework Programme (FP/2007-2013)/ERC Grant Agreement no. 616787 for the financial support. This dissertation would not have been possible without their support. Thanks are also due to the Computer Science Institute of Charles University, Prague for providing the organization support and infrastructure during my Ph.D. journey.

I do not have enough words that can even highlight the role of my parents, Maa and Baba, in my life. Right from my childhood, my education and career have been their sole priority. On countless occasions, they have sacrificed in their lives only for fulfilling my dreams and aspirations. I am indebted to them for their unconditional love, support and faith in every stage of my life. My heartfelt thanks to my brother, Debajyoti, for his kind affection and constant support. I also want to thank my relatives, cousins, in-laws and friends for their love and support.

At the end, I wish to thank my better half, Vineet, in every sense. This work would not have been possible without his support and encouragement. I firmly believe that together we will achieve the best in our lives in all dimensions.

Contents

Preface **5**

I. Combinatorial Lower Bounds & Efficient String and Graph Algorithms **6**

1 Introduction **7**
1.1 Problem Catalogue 7
1.1.1 Edit Distance 7
1.1.2 Approximate Pattern Matching 9
1.1.3 Boolean Matrix Multiplication 11
1.1.4 Weight Tolerant Subgraph for Single Source Shortest Path 12
1.2 Organisation 15

2 Constant approximation of Edit Distance **16**
2.1 Preliminaries 16
2.1.1 Reduction to the gap problem 16
2.1.2 Formal framework of the algorithm 16
2.2 Warm up: A detailed overview of the Covering algorithm 18
2.3 Covering Algorithm: pseudo-code and analysis 24
2.3.1 Pseudo-code 24
2.3.2 Analysis and correctness of CA 28
2.3.3 Time complexity of CA 35
2.4 Min-cost Paths in Shortcut Graphs 36
2.5 Conclusion and Bibliographical Notes 39

3 Approximate Pattern Matching **41**
3.1 Preliminaries 41
3.2 Offline Approximate Pattern Matching 42
3.2.1 Technique Overview 42
3.2.2 Covering phase 43
3.2.3 Correctness of the covering algorithm 45
3.2.4 Time complexity of the covering algorithm 46
3.3 Min-cost Path in a Grid Graph with Shortcuts 47
3.4 Online Approximate Pattern Matching 49
3.4.1 The online algorithm 49
3.5 Conclusion and Bibliographical Notes 51

4 Combinatorial Lower Bounds of Boolean Matrix Multiplication **53**
4.1 Combinatorial Models 53
4.2 Technique Overview 54
4.3 Notation and Preliminaries 55
4.3.1 Matrices 55
4.3.2 Model 56
4.3.3 (r, t)-graphs: The hard instance 57
4.3.4 Diverse and unhelpful graphs 58
4.4 Union Circuits 60

4.5 Circuits with Partitions . . . 61
4.5.1 The cost of chargeable gates in a partition . . . 65
4.5.2 Large number of partitions . . . 67
4.5.3 Density lemma . . . 67
4.5.4 The main proof . . . 68
4.6 Conclusion and Open Problems . . . 69

5 Weight Tolerant Subgraph for Single Source Shortest Path 71
5.1 Preliminaries . . . 71
5.1.1 Definitions . . . 71
5.1.2 Max-flow and farthest min-cut . . . 72
5.1.3 Overview of the construction . . . 73
5.2 Farthest Min-cut of Shortest Path Subgraph . . . 76
5.2.1 Computing farthest min-cut of shortest path subgraph . . . 76
5.2.2 Disjoint shortest path lemma . . . 77
5.3 Construction of k-WTSS and Locality Lemma . . . 79
5.4 Construction of k-WTSS(t) . . . 80
5.4.1 Description of the algorithm . . . 80
5.4.2 Analysis . . . 83
5.5 Lower Bound on the Size of k-WTSS . . . 88
5.6 Conclusion and Bibliographical Notes . . . 89

II. Efficient Construction of Quasi-Gray Codes 90

6 Introduction 91
6.1 Organisation . . . 93

7 Preliminaries and Overview 94
7.1 Preliminaries . . . 94
7.1.1 Construction of Gray codes . . . 95
7.2 Overview of the construction . . . 96
7.3 The Key Tools . . . 97
7.3.1 Chinese Remainder Theorem for Counters . . . 98
7.3.2 Permutation Group and Decomposition of Counters . . . 100

8 Space-optimal Quasi-Gray Codes Over Odd Sized Alphabets 102
8.1 Construction of the Counter . . . 102

9 Quasi-Gray Codes Over Even Sized Alphabets 111
9.1 Quasi-Gray Codes over Binary Alphabet . . . 111
9.1.1 Counters via Linear Transformation . . . 111
9.1.2 Construction of the counter . . . 112
9.2 Getting counters for Even m . . . 114
9.3 Bibliographical Notes . . . 115

10 Conclusion 116

Bibliography 117

List of Figures **128**

List of Tables **129**

List of publications **130**

Preface

The thesis is organised in two parts. Part I studies a group of combinatorial problems related to strings, graphs and boolean matrices. Part II of the thesis presents efficient construction of quasi-Gray Codes.

Part I of the thesis is based on the following publications:

[CDGKS 18] Diptarka Chakraborty, Debarati Das, Elazar Goldenberg, Michal Koucký, Michael Saks. Approximating edit distance within constant factor in truly sub-quadratic time. In *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018*, pages 979-990.

[DKS 18] Debarati Das, Michal Koucký, Michael Saks. Lower bounds for combinatorial algorithms for boolean matrix multiplication. In *35th Symposium on Theoretical Aspects of Computer Science, STACS 2018*, pages 23:1-23:14.

[CD 18] Diptarka Chakraborty and Debarati Das. Sparse weight tolerant subgraph for single source shortest path. In *16th Scandinavian Symposium and Workshops on Algorithm Theory, SWAT 2018*, pages 15:1-15:15.

[CDK 18] Debarati Das, Diptarka Chakraborty, Michal Koucký. Approximate online pattern matching in sub-linear time. *CoRR,abs/1810.03551, 2018*

Part II of the thesis is based on the following publication:

[CDKS 18] Diptarka Chakraborty, Debarati Das, Michal Koucký, Nitin Saurabh. Space-Optimal Quasi-Gray Codes with Logarithmic Read Complexity. In *26th Annual European Symposium on Algorithms , ESA 2018*, pages 12:1-12:15.

Part I

Combinatorial Lower Bounds & Efficient String and Graph Algorithms

1. Introduction

Algorithms are sequential instructions for solving computational problems ranging from addition of two numbers to prediction of the weather. For decades, researchers have taken significant interest in optimizing the resources (such as time and space) used by algorithms. Every improved algorithm brings down the amount of resources needed to solve a particular problem. Nevertheless, the inherent difficulty of a computational problem poses a restriction on such optimization, which is one of the key concerns in computer science. Such restriction is formally referred to as the complexity lower bound. To prove a lower bound for a computational problem, one needs to refute the possibility of any algorithm that can solve the problem using resources lesser than the specified lower bound. As a direct formulation of such a proof can be hard, one feasible approach is to define a meaningful restricted computational model and prove the lower bound under this model.

In this thesis, we study a special class of problems, called *combinatorial problems*, that operate on combinatorial objects such as graphs, hypergraphs, set systems. Algorithms that are designed for assisting in solving combinatorial problems are called *combinatorial algorithms*. These algorithms either attempt to reduce the computational complexity, by exploiting structural similarities in different parts of a combinatorial object, or proving some lower bounds on the computation by showing that, different parts of the object acquire distinct structural properties. Considerable interest has been taken in developing and understanding combinatorial algorithms. This interest is motivated both by intellectual curiosity and by the fascinating property of combinatorial algorithms where, all intermediate computations have a natural combinatorial interpretation in terms of the original problem. Moreover, they are easy to implement and are more useful for practice.

Part I of this thesis is a testimony to the multi-faceted nature of combinatorial problems. Here, the contributions are twofold: First we study a collection of problems related to strings and graphs and for them we devise new algorithms with improved efficiency. Next, we introduce new combinatorial models for boolean matrix multiplication (BMM) problem and prove lower bounds for BMM under these models. Although the aforementioned problems have their own significance with distinct motivation and impact, they share a common flavour in a sense that all of them can be reduced to either reachability or shortest path problem in graphs.

In the rest of the chapter, we introduce each of these problems, along with an overview of related existing work and discuss in brief our main technical contributions.

1.1 Problem Catalogue

1.1.1 Edit Distance

Edit distance is a measure of similarity between two strings. The *edit distance*(aka *Levenshtein distance*) [Lev66] between strings x, y, denoted by $d_{\text{edit}}(x, y)$, is the

minimum number of character insertions, deletions, and substitutions needed to convert x into y. It is a widely used distance measure between strings that finds applications in fields such as computational biology, pattern recognition, text processing and information retrieval. Due to its immense practical impact, for decades, researchers have taken significant interest in designing algorithms that can efficiently compute $d_{\text{edit}}(x, y)$. Additionally, one might also be interested in finding an *alignment* of x and y, i.e., a series of edit operations that transform x into y.

Previous Works: Edit distance can be evaluated exactly in quadratic time via dynamic programming (Wagner and Fischer [WF74]). Landau *et al.* [LMS98] gave an algorithm that finds an optimal alignment in time $O(n + d_{\text{edit}}(x, y)^2)$, improving on a previous $O(n \cdot d_{\text{edit}}(x, y))$ algorithm of Ukkonen [Ukk85]. Masek and Paterson [MP80] obtained the first (slightly) sub-quadratic $O(n^2/\log n)$ time algorithm, and the current asymptotically fastest algorithm Grabowski [Gra16] runs in time $O(n^2 \log\log n/\log^2 n)$. Backurs and Indyk [BI15] showed that a *truly sub-quadratic algorithm* ($O(n^{2-\delta})$ for some $\delta > 0$) would imply a $2^{(1-\gamma)n}$ time algorithm for CNF-satisfiabilty, contradicting the Strong Exponential Time Hypothesis (SETH). Abboud et al. [AHWW16] showed that even shaving off an arbitrarily large polylog factor from n^2 can be hard, as the consequence implies NEXP does not have non-uniform NC^1 circuits. For further "barrier" results, see [ABW15, BK15].

A long line of research has been carried out on *approximating* edit distance. The exact $O(n+k^2)$ time algorithm (where k is the edit distance of the input) of Landau *et al.* [LMS98] yields a linear time $\sqrt{n}$-factor approximation. This approximation factor was improved, first to $n^{3/7}$ [BYTKK04], then to $n^{1/3+o(1)}$ [BES06] and later to $2^{\widetilde{O}(\sqrt{\log n})}$ [AO09], all with slightly superlinear runtime. Batu *et al.* [BEK+03] provided an $O(n^{1-\alpha})$-approximation algorithm with runtime $O(n^{\max\{\frac{\alpha}{2}, 2\alpha-1\}})$. The strongest result of this type is the $(\log n)^{O(1/\epsilon)}$ factor approximation (for every $\epsilon > 0$) with running time $n^{1+\epsilon}$ of Andoni *et al.* [AKO10]. In streaming model a similar kind of approximation algorithm was given by [CGK16]. This algorithm is based on random walks. Random walk was also used for computing the Dyck language edit distance in an earlier paper by Saha [Sah14]. Abboud and Backurs [AB17] showed that a truly sub-quadratic deterministic time $1+o(1)$-factor approximation algorithm for edit distance would imply new circuit lower bounds.

Independent of our work, Boroujeni *et al.* [BEG+18a] obtained a truly sub-quadratic *quantum* algorithm that provides a constant factor approximation. Their latest results [BEG+18b] are a $(3+\epsilon)$ factor with runtime $\widetilde{O}(n^{2-4/21}/\epsilon^{O(1)})$ and a faster $\widetilde{O}(n^{1.708})$-time with a larger constant factor approximation. Andoni and Nguyen [AN10] found a randomized algorithm that approximates Ulam distance of two permutations of $\{1, \ldots, n\}$ (edit distance with only insertions and deletions) within a (large) constant factor in time $\widetilde{O}(\sqrt{n} + n/k)$, where k is the Ulam distance of the input; this was improved by Naumovitz *et al.* [NSS17] to a $(1+\varepsilon)$-factor approximation (for any $\varepsilon > 0$) with similar runtime.

Our Contribution: This thesis contributes, by presenting a truly sub-

quadratic time *classical* algorithm that approximates edit distance within a constant factor.

Theorem 1. *There is a randomized algorithm* **ED-UB** *that on input strings x, y of length n over any alphabet Σ outputs a constant approximation of $d_{edit}(x, y)$ in time $\widetilde{O}(n^{12/7})$, with probability at least $1 - n^{-5}$.*

If the output is U, then the algorithm implicitly finds an alignment of cost at most U. The algorithm can be modified to explicitly output such an alignment.

1.1.2 Approximate Pattern Matching

Finding the occurrences of a pattern in a larger text is one of the fundamental problems in computer science. Due to its large practical applications this problem has been studied extensively under several variations [KJP77, GS81, Abr87, Cro92, GPR95, CGPR95, Ind98, Nav01, KP18]. One of the most natural variation is where we are allowed to have a small number of errors while matching the pattern. This problem of pattern matching while allowing errors is known as *approximate pattern matching*. The kind of possible errors varies with the applications. Generally we capture the amount of errors by the distance metric defined over the set of strings. One common and widely used distance measure is the edit distance. In our work we focus on the approximate pattern matching problem under edit distance. This problem has various applications ranging from computational biology, signal transmission, web searching, text processing and many more.

Given a pattern P of length w and a text T of length n over some alphabet Σ, and an integer k we want to identify all the substrings of T at edit distance at most k from P. The number of such substrings might be quadratic in n. Hence, we just focus on finding the set of all right-end positions in T of these substrings that are at distance at most k. More specifically, for a position t in T, let k_t be the smallest edit distance of a substring of T ending at t-th position in T. (We number positions in T and P from 1.) The goal is to compute the sequence $k_1, k_1, \ldots, k_n$ for P and T.

In the recent past, researchers also studied the approximate pattern matching problem in the online setting. The online version of this problem mostly arises in real life applications that require matching pattern in a massive data set, like in telecommunications, monitoring Internet traffic, building firewall to block viruses, malware connections and many more. Formally, the online approximate pattern matching problem can be defined as follows: first we are given a pattern P and then the text T arrives symbol by symbol. Upon receipt of the t-th symbol of the text, we should output the corresponding k_t. The online algorithm runs in *amortized time* $O(\ell)$ if it runs in total time $O(n \cdot \ell)$ and it uses *succinct space* $O(s)$ if in addition to storing P it uses at most $O(s)$ cells of memory at any time.

Previous Works: The approximate pattern matching problem is one of the most extensively studied problems in modern computer science due to its direct applicability to data driven applications. In contrast to the exact pattern matching here a text location has a match if the distance between the pattern

and the text is within some tolerated limit. In this thesis we study the approximate pattern matching under edit distance metric. For this problem, the very first $O(nw)$-time algorithm was given by Sellers [Sel80] in 1980. Masek and Paterson [MP80] proposed an $O(nw/\log n)$-time $O(n)$-space algorithm using Four Russians [LAAA75] technique. All the hardness results true for edit distance also holds for *approximate pattern matching*. The result of Backurs and Indyk [BI15] indicate that this $O(nw)$ bound cannot be improved significantly unless the Strong Exponential Time Hypothesis (SETH) is false. Moreover Abboud et al. [AHWW16] showed that even shaving an arbitrarily large polylog factor would imply that NEXP does not have non-uniform NC^1 circuits. More hardness results can be found in [ABW15, BK15, AB17, AR18]. In the parametrized version [Mye86, LV89, GP90] gave $O(kn)$-time algorithms where k is the upper limit of allowed edit operations. All of these algorithms use either $O(w^2)$ or $O(n)$ space. However [GG88, UW93] reduced the space usage to $O(w)$ while maintaining the run time. A faster algorithm was given by Cole and Hariharan [CH98], which has a runtime of $O(n(1 + k^4/w))$. We refer the interested readers to a beautiful survey by Navarro [Nav01] for a comprehensive treatment on this topic.

All the above mentioned algorithms assume that the entire text is available from the very beginning of the process. However in the online version, the pattern is given at the beginning and the text arrives in a stream, one symbol at a time. Clifford *et al.* [CEPP08] gave a *black-box algorithm* for online approximate matching where the supported distance metrics are hamming distance, matching with wildcards, k-mismatch, L_1 and L_2 norm. Their algorithm has a run time of $O(\sum_{j=1}^{\log_2 w} T(n, 2^{j-1})/n)$ per symbol arrival, where $T(n, w)$ is the running time of the best offline algorithm. This result was extended in [CS09] by introducing an algorithm solving online approximate pattern matching under edit distance metric in time $O(k \log w)$ per symbol arrival. This algorithm uses $O(w)$-space. In [CS10] the runtime was further improved to $O(k)$ per symbol. Recently, Starikovskaya [Sta17] gave a randomized algorithm which has a worst case time complexity of $O((k^2\sqrt{w} + k^{13}) \log^4 w)$ and uses space $O(k^8\sqrt{w} \log^6 w)$. Unfortunately this algorithm takes both sublinear time and sublinear space for very small values of k. On the lower bound side, Clifford, Jalsenius and Sach [CJS15] showed in the *cell-probe model*, the expected amortized run time of any randomized algorithm solving online approximate pattern matching problem must be $\Omega(\sqrt{\log w}/(\log \log w)^{3/2})$ per output.

Our Contribution: In our work we focus on finding an approximation to the sequence $k_1, k_1, \ldots, k_n$ for P and T. For real numbers $c, k \geq 0$, a sequence $\tilde{k}_1, \ldots, \tilde{k}_n$ is (c, k)-approximation to $k_1, \ldots, k_n$, if for each $t \in \{1, \ldots, n\}$, $k_t \leq \tilde{k}_t \leq c \cdot k_t + k$. Hence, c is the multiplicative error and k is the additive error of the approximation. An algorithm computes (c, k)-approximation to approximate pattern matching if it outputs a (c, k)-approximation of the true sequence $k_1, k_1, \ldots, k_n$ for P and T. We refer $(c, 0)$-approximation simply as c-approximation. In this thesis we prove the following theorem.

Theorem 2. *There is a constant $c \geq 1$ and there is a randomized algorithm that computes c-approximation to approximate pattern matching in time $O(n \cdot w^{3/4})$ with probability at least $(1 - 1/n^3)$.*

For the online setting we show the following result.

Theorem 3. *There is a constant $c \geq 1$ so that there is a randomized online algorithm that computes $(c, w^{8/9})$-approximation to approximate pattern matching in amortized time $O(w^{1-(7/54)})$ and succinct space $O(w^{1-(1/54)})$ with probability at least $1 - 1/poly(n)$.*

To the best of our knowledge this is the first online approximation algorithm that takes sublinear (in the length of the pattern) running time and sublinear succinct space for the approximate pattern matching problem. Our notion of the succinct space data structure used in this work is a bit different than the conventional notion of [Pat08, GST17]. We discuss more about this in Chapter 3.

1.1.3 Boolean Matrix Multiplication

Boolean matrix multiplication (BMM) is one of the core problems in discrete algorithms where, given two boolean matrices the aim is to find their product where all the operations are done over boolean semiring. It has numerous applications including triangle detection in graphs [Ita77], context-free grammar parsing [Val75], and transitive closure etc. [FM71, Fur70, Mun71]. Boolean matrix multiplication has a natural interpretation as a path problem in graphs: Given a layered graph with three layers A, B, C and edges between layers A and B and between B and C, compute the bipartite graph between A and C in which $a \in A$ and $c \in C$ are joined if and only if they have a common neighbor. If we identify the bipartite graph between A and B with its $A \times B$ boolean adjacency matrix $\mathcal{P}$ and the graph between B and C with its $B \times C$ boolean adjacency matrix $\mathcal{Q}$ then the desired graph between A and C is just the boolean product $\mathcal{P} \times \mathcal{Q}$.

Previous Upper Bounds: Boolean matrix multiplication is the combinatorial counterpart of integer matrix multiplication. Both involve the computation of n^2 output values, each of which can be computed in a straightforward way in time $O(n)$ yielding a $O(n^3)$ algorithm. One of the celebrated classical results in algorithms is Strassen's discovery [Str69] that computes ordinary matrix multiplication in *truly subcubic* time, i.e. in time $O(n^\omega)$ for some $\omega < 3$. The algorithm computes the n^2 entries by computing and combining carefully chosen (and highly non-obvious) polynomial functions of the matrix entries. [CW90, Wil12, Gal14] made subsequent improvements by reducing the value of ω.

Despite of the intrinsic combinatorial nature, the asymptotically fastest algorithm known for BMM is obtained by treating the boolean entries as integers and applying fast integer matrix multiplication. The intermediate calculations done for this algorithm seemingly have little to do with the combinatorial structure of the underlying bipartite graphs and therefore lack clarity. Moreover these fast integer matrix multiplication algorithms are impractical because of the huge constant factor hidden in $O(\cdot)$. This indeed motivated the study of a different class of algorithms called "combinatorial" algorithms for BMM. The first non-trivial combinatorial algorithm, famous as Four Russians Algorithm by Arlazarov, Dinic, Kronrod, Faradzhev [VZA70] solves BMM using $O(n^3/\log^2(n))$ operations. In past 10 years a series of combinatorial algorithms [BW12, Cha15, Yu15] have been

developed for BMM, all having complexities of the form $O(n^3/(\log n)^c)$ for increasingly large constants c. The best and most recent of these, due to Yu [Yu15] has complexity $\hat{O}(n^3/\log^4 n)$ (where the $\hat{O}$ notation suppresses $poly(\log\log(n))$ factors.)

While each of these combinatorial algorithms saves just a polylogarithmic factor in contrast to the power of n saving by the algebraic algorithms, one obvious question that arises is: Is there a truly subcubic combinatorial algorithm for BMM?

Our Contribution: In this thesis we address this question and provide results that manifest towards a negative answer. Before proceeding, we need to first make a precise notion of a combinatorial algorithm. To formalize, a combinatorial algorithm requires some computational model which specifies what the algorithm states are, what operations can be performed, and what the cost of those operations is. In particular, we would like that our model is powerful enough to be able to simulate all of the known combinatorial algorithms with running time no larger than their actual running time, but not so powerful that it allows for fast (e.g. quadratic time) algorithms that are not implementable on a real computer. Our work takes a step towards this direction. In our work first we give a relatively relaxed model which is an extension of the *row union model* by Angluin [Ang76]. Particularly in Angluin's model non-identical unions (unions over rows having different indices) having identical content are charged separately. Whereas in our model a group of different unions having identical content is charged only once. Our first lower bound shows even in this model, there are matrices for which BMM has almost cubic cost.

Theorem 4 (Informal statement)**.** *In the row-union model with removed repetitions the cost of Boolean matrix multiplication is* $\Omega(n^3/2^{O(\sqrt{\log n})})$.

Subsequently we propose a more general model where we allow the rows to be divided into pieces. This model is indeed capable of efficiently simulating Four Russians algorithm, but is sufficiently more general. We then prove a super quadratic lower bound in this model.

Theorem 5 (Informal statement)**.** *In the row-union model with partitioning and removed repetitions the cost of Boolean matrix multiplication is* $\Omega(n^{7/3}/2^{O(\sqrt{\log n})})$.

1.1.4 Weight Tolerant Subgraph for Single Source Shortest Path

In the real world, networks are prone to failures and most of the time such failures are unpredictable as well as unavoidable in any physical system such as communication network or road network. As these networks can be naturally modelled by graphs, this yields several graph theoretical problems like reachability, shortest path, distance preserver in fault tolerant model. Normally such failures are much smaller in number compared to the size of the graph. Thus we can associate a parameter to capture the number of edge or vertex failures and try to build fault tolerant data structures of size depending on this failure parameter for various graph theoretic problems.

Unfortunately, in case of single source shortest path problem, it is already known from [DTCR08] that there are graphs with n vertices for which to preserve the distances under even single edge failure, we need to store a subgraph of size at least $\Omega(n^2)$. On the other hand, in case of reachability problem we know the construction of connectivity preserving subgraph of size only $O(2^k n)$ due to [BCR16] where k is the number of edge failures. However, in case of real world applications, it is not always the case that there are failures of edges or vertices. Instead, for weighted graphs, weight of some edge or vertex may increase. For simplicity, we consider weight to be only on the edges of the graph. In general, weight of an edge captures aspect like congestion on a particular link in a network. So it is quite natural to consider the scenario when some links (or edges) become more congested. Again the good thing is that most of the time such a congestion is bounded, i.e., over a network total increase in congestion is bounded because of many reasons like bounded maximum number of consumers present in a network at any particular time etc. One can easily capture the increase in congestion by a parameter k that bounds the amount of total increase in weight over all the edges of a graph. Occurrence of such bounded congestion motivates us to study the single source shortest path problem under this model.

Related works: Single source shortest path is a well studied problem under the edge or vertex failure model. One can easily define k-Fault Tolerant Shortest-path Subgraph (k-FTSS) that preserves the distance information from a specific source vertex under at most k edge failures. Unfortunately, we know that there are weighted graphs for which no sparse k-FTSS exists even for $k = 1$, i.e., there are weighted graphs with n vertices for which any 1-FTSS must contain $\Omega(n^2)$ many edges [DTCR08]. This lower bound on size of 1-FTSS is true even for undirected graphs. However, better bounds are known for unweighted graphs for $k \leq 2$. Parter and Peleg [PP13] provided a construction of $O(n^{3/2})$ sized 1-FTSS and showed that this bound is optimal. Later, Parter [Par15] extended the construction to the case $k = 2$ for undirected graphs on the cost of weakening the bound. He gave an algorithm to compute 2-FTSS of size $O(n^{5/3})$ along with a matching lower bound. So far there is nothing known for the case $k > 2$.

However, the situation is much better for single source reachability problem which is closely related to single source shortest path problem. Baswana, Choudhary and Roditty [BCR16] showed that we can compute k-FTRS, which is a subgraph that preserves the reachability information from a given source under at most k edge failures, containing $2^k n$ many edges. They also provided a matching lower bound. Another interesting related problem is to compute fault tolerant reachability oracle. It is trivial to see that using $O(2^k n)$ size k-FTRS [BCR16] one can answer any reachability query in $O(n)$ time for any constant value of k. However for $k \leq 2$, $O(n)$ size data structure is known that can answer any single source reachability query in $O(1)$ time [LT79, Cho16]. Very recently, existence of an efficient algorithm to find strongly connected component in fault tolerant model has also been shown [BCR17].

Now let us come back to the shortest path problem. Instead of preserving the exact distances (between any pair of vertices), if we consider to preserve the distances only approximately, then much better results are known. In the literature such approximate distance preserving subgraphs are called *spanners*. Construc-

tion of spanners with both additive and multiplicative stretch have been studied extensively [Erd, ADD+93, ACIM99, BKMP05, Woo10, AB16, ABP17]. Fault tolerant version of spanners were first introduced in the geometric setting [Luk99, LNS02, CZ04]. For k edge failures, construction of a $(2l-1)$ multiplicative spanner of size $\widetilde{O}(kn^{1+1/l})$, for any $k, l \geq 1$, was provided in [CLPR09] whereas for k vertex failures, the upper bound on size is known to be $\widetilde{O}(k^{2-1/l}n^{1+1/l})$ [DK11]. In case of single vertex failure in an undirected graph, construction of a $O(n \log n)$ sized subgraph that preserves distances within a multiplicative factor of 3, is known due to [BK13]. The bound on the size was later improved to $3n$ [PP14]. Braunschvig, Chechik and Peleg [BCPS15] initiated the study of additive spanners. For β-additive spanner, Parter and Peleg [PP14] provided a $\Omega(n^{1+\epsilon(\beta)})$ size lower bound where $\epsilon(\beta) \in (0, 1)$. They also constructed a 4-additive spanner of size $O(n^{4/3})$ that is resilient to single edge failure. For single vertex failure, constructions of additive spanners were given in [Par14, BGG+15]. Very recently, for any fixed $k \geq 1$, construction of a sub-quadratic size 2-additive spanner resilient to k edge or vertex failures has been shown for unweighted undirected graphs [BGPW17]. In the same paper, authors also showed that to achieve $O(n^{2-\epsilon})$ upper bound, one must allow $\Omega(\epsilon k)$ additive error.

Another closely related problem is the replacement path problem where given a source and destination vertex and an edge, the objective is to find a path from source to destination avoiding that particular given edge. Though the problem was initially defined for single edge failure, later it was extended to multiple edge failures also. Readers may refer to [Wil11, RZ12, GW12, WY13] for recent progresses on this problem.

Our Contribution: In this thesis, we introduce the bounded weight increment model and under this model we study the single source shortest path problem of weighted directed graphs. The main goal is to find a sparse subgraph that preserves distance between a designated source and any other vertices under weight increment operation. We define such subgraph as follows.

Definition 6 (Informal definition)**.** *Given a weighted, directed graph G with a designated source vertex s and an integer $k \geq 1$, a subgraph H of G is said to be* k-Weight Tolerant Shortest-path Subgraph (k-WTSS) *of G if H preserves the distance between the source s and any other vertex of G as long as the total weight increment of all edges is bounded by k and the weight is incremented integrally.*

The main contribution of our work is to provide an efficient construction of a sparse k-WTSS for any $k \geq 1$, where sparsity of k-WTSS depends on the parameter k.

Theorem 7 (Informal statement)**.** *Given a directed weighted graph G with n vertices and m edges, there exists an $O((k)^k m^2 n)$-time algorithm that for any given integer $k \geq 1$ constructs a k-WTSS of G such that the in-degree of every vertex in the k-WTSS is bounded by $e \cdot (k-1)!2^k$.*

Next, we also prove a lower bound of $c \cdot 2^k n$ for some constant $c \geq 5/4$, on the size of k-WTSS.

1.2 Organisation

Part I of the thesis is organised as follows: In Chapter 2 we provide an algorithm that computes the edit distance of two given strings within a constant approximation factor and has a truly sub-quadratic runtime. Based on this edit distance algorithm, in Chapter 3 we give another algorithm that solves the approximate pattern matching problem under both offline and online setting in sublinear (in the length of the pattern) running time and sublinear succinct space. In Chapter 4 we prove lower bounds for combinatorial models for boolean matrix multiplication problem. In Chapter 5 we give the construction of a spare subgraph of a directed weighted graph that preserves the distance between a designated source and any other vertices under bounded weight increment operation. We also provide a comparative lower bound on the size of the sparse subgraph.

Chapter 2 is based on [CDG+18], Chapter 3 is based on [CDK18], Chapter 4 is based on [DKS18], Chapter 5 is based on [CD18].

2. Constant approximation of Edit Distance

Recall the edit distance between two strings x and y is the minimum number of character insertions, deletions and substitutions required to convert x into y. In this chapter we present an algorithm that given two strings x, y of length n, computes a constant approximation of their edit distance $d_{\text{edit}}(x, y)$ in time $\widetilde{O}(n^{2-2/7})$. We also discuss about how to improve this runtime further using recursion while maintaining the constant factor approximation guarantee. Formally our main result is the following:

Theorem 8. *[main theorem] There is a randomized algorithm* **ED-UB** *that on input strings x, y of length n over any alphabet Σ outputs a value $k \in (0, n]$ such that $d_{edit}(x, y) \leq k \leq 1220 d_{edit}(x, y)$ with probability at least $1 - n^{-5}$. Furthermore, if $d_{edit}(x, y) = \theta n$, then the algorithm runs in time $\tilde{O}(n^{2-2/7}\theta^{4/7})$.*

2.1 Preliminaries

2.1.1 Reduction to the gap problem

Instead of providing the algorithm computing edit distance of x and y directly, we design a *gap algorithm* $\mathbf{GAP\text{-}UB}_\theta$, which distinguishes inputs with $d_{\text{edit}}(x, y) \leq \theta n$ (where the output is at most $610\theta n$), and those with $d_{\text{edit}}(x, y) > 610\theta n$ (where the output is greater than $610\theta n$). Formally we show:

Theorem 9. *For every $\theta \in [n^{-1/5}, 1]$, there is a randomized algorithm $\mathbf{GAP\text{-}UB}_\theta$ that on input strings x, y of length n outputs $u = \mathbf{GAP\text{-}UB}_\theta(x, y)$ such that: (1) $d_{edit}(x, y) \leq u$ and (2) on any input with $d_{edit}(x, y) \leq \theta n$, $u \leq 610\theta n$ with probability at least $1 - n^{-7}$. The runtime of $\mathbf{GAP\text{-}UB}_\theta$ is $\tilde{O}(n^{2-2/7}\theta^{4/7})$.*

Proof of Theorem 8 from Theorem 9. The construction of the main algorithm **ED-UB** from **GAP-UB** is standard: Run the exact algorithm of [LMS98] with runtime $O(n + k^2)$ time on instances of edit distance k, for $O(n + n^{2-2/5})$ time. If it terminates then it outputs the exact edit distance. Otherwise, the failure to terminate implies $d_{\text{edit}}(x, y) \geq n^{4/5}$. Now run $\mathbf{GAP\text{-}UB}_{\theta_j}(x, y)$ for $\theta_j = (1/2)^j$ for $j = \{0, \ldots, \frac{\log n}{5}\}$ and output the minimum of all upper bounds obtained. Let j be the largest index with $\theta_j n \geq d_{\text{edit}}(x, y)$ (such an index exists since $j = 0$ works). The output is at most $610\theta_j n \leq 1220 d_{\text{edit}}(x, y)$. We run at most $O(\log n)$ iterations, each with runtime $\tilde{O}(n^{2-2/7})$. □

We devote the rest of the chapter to prove Theorem 9.

2.1.2 Formal framework of the algorithm

We use a standard two-dimensional representation of edit distance. Visualize x as lying on a horizontal axis and y as lying on a vertical axis, with horizontal coordinate $i \in \{1, \ldots, n\}$ corresponding to x_i and vertical component j corresponding to y_j. The width $\mu(I)$ of interval $I \subseteq \{0, 1, \ldots, n\}$ is $\max(I) - \min(I) = |I| - 1$.

Also, x_I denotes the substring of x indexed by $I \setminus \{\min(I)\}$. (Note: $x_{\min(I)}$ is not part of x_I, e.g., $x = x_{\{0,\ldots,n\}}$. This convention is motivated by Proposition 10.) We refer to I as an *x-interval* to indicate that it indexes a substring of x, and J as a *y-interval* to indicate that it indexes a substring of y. A *box* is a set $I \times J$ where I is a x-interval and J is a y-interval; $I \times J$ corresponds to the substring pair (x_I, y_J). $I \times J$ is a w-box if $\mu(I) = \mu(J) = w$. We often abbreviate $d_{\text{edit}}(x_I, y_J)$ by $d_{\text{edit}}(I, J)$. A *decomposition* of an x-interval I is a sequence $I_1, \ldots, I_\ell$ of subintervals with $\min(I_1) = \min(I)$, $\max(I_\ell) = \max(I)$ and for $j \in [\ell - 1]$, $\max(I_j) = \min(I_{j+1})$.

Edit distance graph. A edit distance graph, associated to x, y is a directed weighted graph $G_{x,y}$, with vertex set $\{0, \ldots, n\} \times \{0, \ldots, n\}$ and all edges of the form $(i-1, j) \to (i, j)$ (H-steps), $(i, j-1) \to (i, j)$ (V-steps) and $(i-1, j-1) \to (i, j)$ (D-steps). Every H-step or V-step costs 1, and D-steps cost 1 if $x_i \neq y_j$ and 0 otherwise. There is a 1-1 correspondence that maps a path from $(0, 0)$ to (n, n) to an *alignment* from x to y, i.e. a set of character deletions, insertions and substitutions that changes x to y, where an H-step $(i-1, j) \to (i, j)$ means "delete x_i", a V-step $(i, j-1) \to (i, j)$ means "insert y_j between x_i and x_{i+1}" and a D-step $(i-1, j-1) \to (i, j)$ means "replace x_i by y_j, unless they are already equal". We have:

Proposition 10. *The cost of an alignment, cost(τ), is the sum of edge costs of its associated path τ, and $d_{edit}(x, y)$ is equal to cost($G_{x,y}$), the min-cost of an alignment path from $(0, 0)$ to (n, n).*

For $I, J \subseteq \{0, \ldots, n\}$, $G_{x,y}(I \times J) \cong G_{x_I, y_J}$ is the grid graph induced on $I \times J$, and $d_{\text{edit}}(I, J) = \text{cost}(G_{x,y}(I \times J))$. The natural high-level idea of **GAP-UB$_\theta$** appears (explicitly or implicitly) in previous work.

Certified box. A *certified boxes* is a pair $(I \times J, \kappa)$, where κ is an upper bound on the *normalized edit distance* $\Delta_{\text{edit}}(x_I, y_J) = d_{\text{edit}}(x_I, y_J)/\mu(I)$.

Our algorithm has two phases. First, the *covering phase* identifies a set $\mathcal{R}$ of *certified boxes*. Second, the *min-cost path phase*, takes input $\mathcal{R}$ and uses a straightforward customized variant of dynamic programming to find an upper bound $U(\mathcal{R})$ on $d_{\text{edit}}(x, y)$ in time quasilinear in $|\mathcal{R}|$. The central issue is to ensure that the covering phase outputs $\mathcal{R}$ that is sufficiently informative so that $U(\mathcal{R}) \leq c \cdot d_{\text{edit}}(x, y)$ for constant c, while running in sub-quadratic time.

Simplifying assumptions. The input strings x, y have equal length n. (It is easy to reduce to this case: pad the shorter string to the length of the longer using a new symbol. The edit distance of the new pair is between the original edit distance and twice the original edit distance. This factor 2 increase in approximation factor can be avoided by generalizing our algorithm to the case $|x| \neq |y|$, but we omit this from this chapter.) We assume n is a power of 2 (by padding both strings with a new symbol, which leaves edit distance unchanged). We assume that θ is a (negative) integral power of 2. The algorithm involves integer parameters w_1, w_2, d, all of which are chosen to be powers of 2.

2.2 Warm up: A detailed overview of the Covering algorithm

In this section we give a detailed overview of the covering phase algorithm, its proof of correctness and run time analysis. For the sake of comprehensibility we ignore minor technical details. In Section 2.3, we provide the pseudo-code corresponding to this overview, with technical differences mainly to improve run-time. We will illustrate the sub-quadratic time analysis with the sample input parameter $\theta = n^{-1/50}$ and algorithm parameters $w_1 = n^{1/10}$, $w_2 = n^{3/10}$ and $d = n^{1/5}$.

Adequate approximating sequence. An *adequate approximating sequence* for some path τ is a sequence σ of certified boxes $(I_1 \times J_1, \kappa_1), \ldots, (I_\ell \times J_\ell, \kappa_\ell)$ that satisfies:

1. $I_1, \ldots, I_\ell$ is a decomposition of $\{0, \ldots, n\}$.
2. $I_i \times J_i$ is an *adequate cover of* τ_i, where $\tau_i = \tau_{I_i}$ denotes the minimal subpath of τ whose projection to the x-axis is I_i, and adequate cover means that the (vertical) distance from the start vertex (resp. final vertex) of τ_i and the lower left (resp. upper right) corner of $I_i \times J_i$, is at most a constant multiple of $\text{cost}(\tau_i) + \theta$.
3. The sequence σ is *adequately bounded*, i.e., $\sum_i \mu(I_i)\kappa_i \leq c(\text{cost}(\tau) + \theta n)$, for a constant c.

This is a slight oversimplification of Definition 18 of (k, ζ)-*approximation of* τ by a sequence of certified boxes.

The covering phase outputs a set $\mathcal{R}$ of certified boxes. The goal is that $\mathcal{R}$ includes an *adequate approximating sequence* for some min-cost path τ in $G_{x,y}$. The intuition behind the second condition is that τ_i is "almost" a path between the lower left and upper right corners of $I_i \times J_i$. Now τ_i might have a vertical extent J' that is much larger than its horizontal extent I_i, in which case it is impossible to place a square $I_i \times J_i$ with corners close to both endpoints of τ_i. But in that case, τ_i has a very high cost (at least $|\mu(J') - \mu(I_i)|$. The closeness required is adjusted based on $\text{cost}(\tau_i)$, with relaxed requirements if $\text{cost}(\tau_i)$ is large.

The output of the min-cost path phase should satisfy the requirements of **GAP-UB**$_\theta$. Lemma 5 shows that if the min-cost path phase receives $\mathcal{R}$ that contains a (k, θ)-approximating sequence to some min-cost path τ, then it will output an upper bound to $d_{\text{edit}}(x, y)$ that is at most $k'(d_{\text{edit}}(x, y) + \theta n)$ for some k'. So that on input x, y with $d_{\text{edit}}(x, y) \leq \theta n$, the output is at most $2k'\theta n$, satisfying the requirements of **GAP-UB**$_\theta$. This formalizes the intuition that an adequate approximating sequence captures enough information to deduce a good bound on $\text{cost}(\tau)$.

Once and for all, we fix a min-cost path τ. Our task for the covering phase is that, with high probability, $\mathcal{R}$ includes an adequate approximating sequence for τ.

τ-match. A τ-*match* for an x-interval I is a y-interval J such that $I \times J$ is an adequate cover of τ_I.

It is easy to show (Proposition 14) that this implies $d_{\text{edit}}(I, J) \leq (\text{cost}(\tau_I) + \theta\mu(I))$.

τ-compatible. A box $I \times J$ is said to be *τ-compatible* if J is a τ-match for I and a box sequence is *τ-compatible* if every box is τ-compatible.

A τ-compatible certified box sequence whose distance upper bounds are (on average) within a constant factor of the actual cost, satisfies the requirements for an adequate approximating sequence. Our covering algorithm will ensure that $\mathcal{R}$ contains such a sequence.

A natural decomposition is $\mathcal{I}_{w_1}$, with all parts of width w_1 (think of w_1 as a power of 2 that is roughly $n^{1/10}$) so $\ell = n/w_1$ and $I_j = \{(j-1)w_1, \cdots, (j)w_1\}$. The naïve approach to building $\mathcal{R}$ is to include certified boxes for enough choices of J to guarantee a τ-match for each I_j.

δ-aligned interval. An interval of width w_1 is *δ-aligned* if its upper and lower endpoints are both multiples of δw_1.

(A slightly altered definition of δ-aligned boxes is used for the formal proof in Section 2.3)

We restrict attention to x-intervals in $\mathcal{I}_{w_1}$, called ***x-candidates*** and θ-aligned y-intervals of width w_1 called ***y-candidates***. It can be shown (see Proposition 15) that an x-interval I always has a τ-match J that is θ-aligned. (In this overview we will fix δ to θ for simplification; the actual algorithm has $O(\log n)$ iterations during which the value of δ varies, giving improvements in runtime.) For each x-candidate I, designate one such τ-match as the *canonical τ-match*, $J^\tau(I)$ for I, and $I \times J^\tau(I)$ is the *canonical τ-compatible box for I*.

In the *exhaustive approach*, for each (x-candidate, y-candidate)-pair (I, J), its edit distance is computed in time $O(w_1^2)$, and the certified box $(I \times J, \Delta_{\text{edit}}(I, J))$ is included. There are $\frac{n}{w_1}\frac{n}{\theta w_1}$ boxes, so the time for all edit distance computations is $O(\frac{n^2}{\theta})$, which is worse than quadratic. (The factor $\frac{1}{\theta}$ can be avoided by standard techniques, but this is not significant to the quest for a sub-quadratic algorithm, so we defer this until the next section.) Note that $|\mathcal{R}|$ is $\frac{n^2}{\theta(w_1)^2}$ (which is $n^{1.82}$ for our sample parameters) so at least the min-cost path phase (which runs in time quasi-linear in $\mathcal{R}$) is truly sub-quadratic.

To achieve a sub-quadratic runtime, two natural goals are: (1) Reduce the amortized time per box, required to certify boxes significantly below $(w_1)^2$ and (2) Reduce the total number of certified boxes created significantly below $\frac{n^2}{\theta(w_1)^2}$. Neither goal is always achievable, and our covering algorithm combines them in order to ensure that at least one of them is always satisfied.

Reducing amortized time for certifying boxes: the *dense case* algorithm. We aim to reduce the amortized time per certified box to be much smaller than $(w_1)^2$. We divide our search for certified boxes into iterations $i \in \{0, \ldots, \log n\}$. For iteration i, start with $\epsilon_i = 2^{-i}$. For all candidate pairs I, J with $\Delta_{\text{edit}}(I, J) \leq \epsilon_i$, our goal is to include the certified box $(I \times J, c\epsilon_i)$ for a fixed constant c. If we succeed, then for each I_j and its canonical τ-match $J^\tau(I_j)$, and for the largest index i for which $\Delta_{\text{edit}}(I_j, J^\tau(I_j)) \leq \epsilon_i$, iteration i will certify $(I_j \times J^\tau(I_j), \kappa_j)$ with $\kappa_j \leq c\epsilon_i \leq 2c\Delta_{\text{edit}}(I_j, J^\tau(I_j))$, as needed.

For a string z of size w_1, let $\mathcal{H}(z, \rho)$ be the set of x-candidates I with $\Delta_{\text{edit}}(z, x_I) \leq \rho$ and $\mathcal{V}(z, \rho)$ be the set of y-candidates J with $\Delta_{\text{edit}}(z, y_J) \leq \rho$. In iteration i, for each x-candidate I, we will specify a set $\mathcal{Q}_i(I)$ of y-candidates that includes $\mathcal{V}(x_I, \epsilon_i)$ and is contained in $\mathcal{V}(x_I, 5\epsilon_i)$. The set of certified boxes

$(I \times J, 5\epsilon_i)$ for all x-candidates I and $J \in \mathcal{Q}_i(I)$ satisfies the goal of iteration i.

Iteration i proceeds in rounds. In each round we select an x-candidate I, called the *pivot*, for which $\mathcal{Q}_i(I)$ has not yet been specified. Compute $\Delta_{\text{edit}}(x_I, y_J)$ for all y-candidates J and $\Delta_{\text{edit}}(x_I, x_{I'})$ for all x-candidates I'; these determine $\mathcal{H}(x_I, \rho)$ and $\mathcal{V}(x_I, \rho)$ for any ρ. For all $I' \in \mathcal{H}(x_I, 2\epsilon_i)$, set $\mathcal{Q}_i(I') = \mathcal{V}(x_I, 3\epsilon_i)$. By the triangle inequality, for each $I' \in \mathcal{H}(x_I, 2\epsilon_i)$, $\mathcal{V}(x_I, 3\epsilon_i)$ includes $\mathcal{V}(x_{I'}, \epsilon_i)$ and is contained in $\mathcal{V}(x_{I'}, 5\epsilon_i)$ so we can certify all the boxes with upper bound $5\epsilon_i$. Mark intervals in $\mathcal{H}(x_I, 2\epsilon_i)$ as *fulfilled* and proceed to the next round, choosing a new pivot from among the unfulfilled x-candidates.

The number of certified boxes produced in a round is $|\mathcal{H}(x_I, 2\epsilon_i)| \times |\mathcal{V}(x_I, 3\epsilon_i)|$. If this is much larger than $O(\frac{n}{\theta w_1})$, the number of edit distance computations, then we have significantly reduced amortized time per certified box. (For example, in the trivial case $i = 0$, every candidate box will be certified in a single round.) But in worst case, there are $\frac{n}{w_1}$ rounds each requiring $\Omega(\frac{nw_1}{\theta})$ time, for an unacceptable total time $\Theta(n^2/\theta)$.

(d, ϵ)-dense. An x-candidate I is (d, ϵ)-*dense* if $|\mathcal{V}(x_I, \epsilon)| \geq d$, i.e., x_I is ϵ-close in edit distance to at least d y-candidates. If a x-candidate is not (d, ϵ)-*dense* then it is (d, ϵ)-*sparse*.

(A more formal definition is given in Section 2.3)

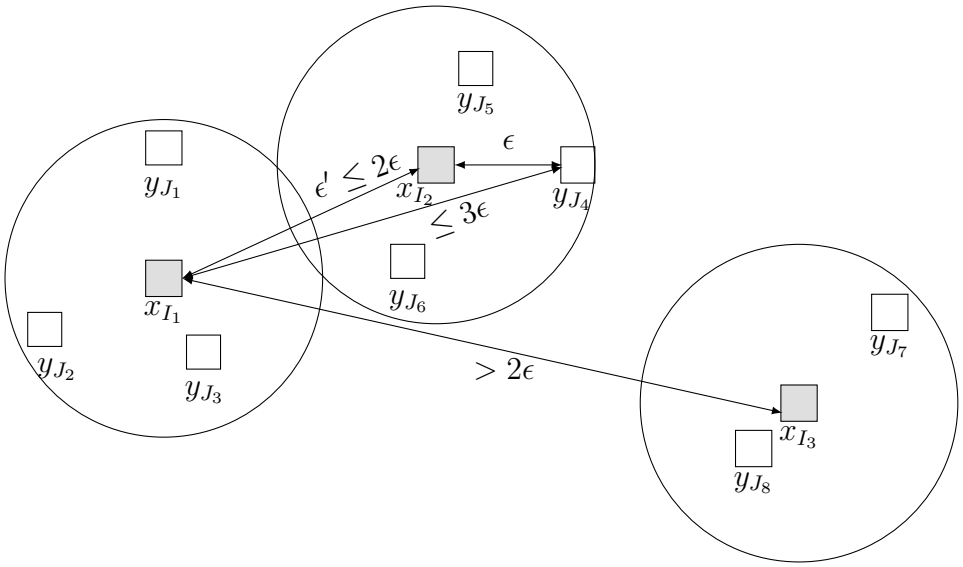


Figure 2.1: $\mathcal{V}(x_{I_1}, \epsilon)$ and $\mathcal{V}(x_{I_3}, \epsilon)$ are disjoint.

Now we discuss a situation where the number of rounds are much less than $\frac{n}{w_1}$. Since any two pivots are necessarily greater than $2\epsilon_i$ apart, the sets $\mathcal{V}(x_I, \epsilon_i)$ for distinct pivots are disjoint. If in each round i, we manage to select a (d, ϵ_i)-dense (think of $d = n^{1/5}$) pivot I, then the number of rounds is $O(\frac{n}{w_1 d\theta})$ and the overall time will be $\Theta(\frac{n^2}{d\theta^2})$. For the sample parameters this is $\Theta(n^{1.84})$. But there's no reason to expect that we'll only choose dense pivots; indeed there need not be any dense pivot.

Let's modify the process a bit. When choosing potential pivot I, first test whether or not it is (approximately) (d, ϵ_i)-dense. This can be done with high

probability, by randomly sampling $\tilde{\Theta}(\frac{n}{\theta w_1 d})$ y-candidates and finding the fraction of the sample that are within ϵ_i of x_I. If this fraction is less than $\frac{\theta w_1 d}{2n}$ then I is *declared sparse* and abandoned as a pivot; otherwise I is *declared dense*, and used as a pivot. With high probability, all (d, ϵ_i)-dense intervals that are tested are declared dense, and all tested intervals that are not $(d/4, \epsilon_i)$-dense are declared sparse, so we assume this is the case. Then all pivots are processed (as above) in time $O(\frac{n^2}{d\theta^2})$ (under sample parameters: $O(n^{1.84})$). We pay $\tilde{O}(\frac{n}{w_1 d\theta})(w_1)^2$ to test each potential pivot (at most $\frac{n}{w_1}$ of them) so the overall time to test potential pivots is $\tilde{O}(\frac{n^2}{d\theta})$ (with sample parameters: $\tilde{O}(n^{1.82})$).

Each iteration i (with different ϵ_i) splits x-candidates into two sets, $\mathcal{S}_i$ of intervals that are declared sparse, and all of the rest for which we have found the desired set $\mathcal{Q}_i(I)$. With high probability every interval in $\mathcal{S}_i$ is indeed (d, ϵ_i)-sparse, but a sparse interval need not belong to $\mathcal{S}_i$, since it may belong to $\mathcal{H}(x_I, 2\epsilon_i)$ for some selected pivot I.

For every x-candidate $I \notin \mathcal{S}_i$ we have met the goal for the iteration. If $\mathcal{S}_i$ is very small for all iterations, then the set of certified boxes will suffice for the min-cost path algorithm to output a good approximation. But if $\mathcal{S}_i$ is not small, another approach is needed.

Reducing the number of candidates explored: the *diagonal extension algorithm.* For each x-candidate I, although it suffices to certify the single box $(I, J^\tau(I))$ with a good upper bound, since τ is unknown, the exhaustive and dense case approaches both include certified boxes for all y-candidates J. The potential savings in the dense case approach comes from certifying many boxes simultaneously using a relatively small number of edit distance computations.

Here's another approach: for each x-candidate I try to quickly identify a relatively small subset $\mathcal{Y}(I)$ of y-candidates that is guaranteed to include $J^\tau(I)$. If we succeed, then the number of boxes we certify is significantly reduced, and even paying quadratic time per certified box, we will have a sub-quadratic algorithm.

We need the notion of *diagonal extension* of a box. The *main diagonal* of box $I \times J$, is the segment joining the lower left and upper right corners.

Diagonal Extension. The square box $I' \times J'$ is a *diagonal extension* of a square subbox $I \times J$ if the main diagonal of $I \times J$ is a subsegment of the main diagonal of $I' \times J'$.(see Definition 16)

Given square box $I \times J$ and $I' \subset I$ the *diagonal extension of $I \times J$ with respect to I'* is the unique diagonal extension of $I \times J$ having x-interval I'. The key observation (Proposition 17) is: if $I \times J$ is an adequate cover of τ_I then any diagonal extension $I' \times J'$ is an adequate cover of $\tau_{I'}$.

Now let w_1, w_2 be two numbers with $w_1 | w_2$ and $w_2 | n$. (Think of $w_1 = n^{1/10}$ and $w_2 = n^{3/10}$.) We use the decomposition $\mathcal{I}_{w_2}$ of $\{0, \ldots, n\}$ into intervals of width w_2. The set of y-candidates consists θ-aligned vertical intervals of width w_2 and has size $\frac{n}{\theta w_2}$. To identify a small set of potential matches for $I' \in \mathcal{I}_{w_2}$, we will identify a set (of size much smaller than $\frac{n}{w_2}$) of w_1-boxes $\mathcal{B}(I')$ having x-interval in $\mathcal{I}_{w_1}(I')$ (the decomposition of I' into width w_1 intervals). For each box in $\mathcal{B}(I')$ we determine the diagonal extension $I' \times J'$ with respect to I', compute $\kappa = \Delta_{\text{edit}}(I', J')$ and certify $(I' \times J', \kappa)$. Our hope is that $\mathcal{B}(I')$ includes a τ-compatible w_1-box $I'' \times J^\tau(I'')$, then the observation above implies that its diagonal extension provides an adequate cover for $\tau_{I'}$.

Here's how to build $\mathcal{B}(I')$: Randomly select a polylog(n) size set $\mathcal{H}(I')$ of

w_1-intervals from $\mathcal{I}_{w_1}(I')$. For each $I'' \in \mathcal{H}(I')$ compute $\Delta_{\text{edit}}(I'', J'')$ for each y-candidate J'', and let $\mathcal{J}(I'')$ consist of the d candidates J'' with smallest edit distance to I''. Here d is a parameter; think of $d = n^{1/5}$ as before. $\mathcal{B}(I')$ consists of all $I'' \times J''$ where $I'' \in \mathcal{H}(I')$ and $J'' \in \mathcal{J}(I'')$.

To bound runtime: Each $I' \in \mathcal{I}_{w_2}$ requires $\tilde{O}(\frac{n}{\theta w_1})$ width-w_1 $\Delta_{\text{edit}}()$ computations, taking time $\tilde{O}(\frac{nw_1}{\theta})$. Diagonal extension step requires $\tilde{O}(d)$ width-w_2 $\Delta_{\text{edit}}()$ computations, for time $\tilde{O}(dw_2^2)$. Summing over $\frac{n}{w_2}$ choices for I' gives time $\tilde{O}(n^2 \frac{w_1}{\theta w_2} + ndw_2)$ (with sample parameters: $\tilde{O}(n^{1.82})$).

Why should $\mathcal{B}(I')$ include a box that is an adequate approximation to $\tau_{I'}$? The intuition behind the choice of $\mathcal{B}(I')$ is that an adequate cover for $\tau_{I'}$ should typically be among the cheapest boxes of the form $I' \times J'$, and if $I' \times J'$ is cheap then for a randomly chosen w_1-subinterval I'', we should also have $I'' \times J^\tau(I'')$ is among the cheapest boxes for I''.

Clearly this intuition is faulty: I' may have many inexpensive matches J' such that $I' \times J'$ is far from $\tau_{I'}$, which may all be much cheaper than the match we are looking for. In this bad situation, there are many y-intervals J' such that $\Delta_{\text{edit}}(I', J')$ is smaller than the match we are looking, and this is reminiscent of the *good* situation for the dense case algorithm, where we hope that I' has lots of close matches. This suggests combining the two approaches, and leads to our full covering algorithm.

The full covering algorithm. Given the dense case and diagonal extension algorithms, the full covering algorithm is easy to describe. The parameters w_1, w_2, d are as above. We iterate over $i \in \{0, \ldots, \log n\}$ with $\epsilon_i = 2^{-i}$. In iteration i, we first run the dense case algorithm, and let $\mathcal{S}_i$ be the set of intervals declared sparse. Then run the diagonal extension algorithm described earlier (with small modifications): For each w_2-interval I', select $\mathcal{H}(I') = \mathcal{H}_i(I')$ to consist of $\theta(\log^2 n)$ independent random selections from $\mathcal{S}_i$. For each $I'' \in \mathcal{H}_i(I')$, find the set of vertical candidates J'' for which $\Delta_{\text{edit}}(I'', J'') \leq \epsilon_i$. Since I'' is (almost certainly) d-sparse, the number of such J'' is at most d. Proceeding as in the diagonal extension algorithm, we produce a set $\mathcal{P}_i(I')$ of $\tilde{O}(d)$ certified w_2-boxes with x-interval I'. Let $\mathcal{R}_D$ (resp. $\mathcal{R}_E$) be the set of all certified boxes produced by the dense case iterations, resp. diagonal extension iterations. The output is $\mathcal{R} = \mathcal{R}_D \cup \mathcal{R}_E$. (See Figure 2.2 for an illustration of the output $\mathcal{R}$.)

The runtime is the sum of the runtime of the dense case and diagonal extension algorithms, as analyzed above. Later, we will give a more precise runtime analysis for the pseudo-code.

To finish this extended overview, we sketch the argument that $\mathcal{R}$ satisfies the covering phase requirements.

Claim 1. *Let I' be an interval in the w_2-decomposition. Either (1) the output of the dense case algorithm includes a sequence of certified w_1-boxes that adequately approximates the subpath $\tau_{I'}$, or (2) with high probability the output of the sparse case algorithm includes a single w_2-box that adequately approximates $\tau_{I'}$.*

(This claim is formalized in Claim 4.) Stitching together the subpaths for all I' implies that $\mathcal{R}$ will contain a sequence of certified boxes that adequately approximates τ.

To prove the claim, we establish a sufficient condition for each of the two conclusion and show that if the sufficient condition for the second conclusion

fails, then the sufficient condition for the first holds.

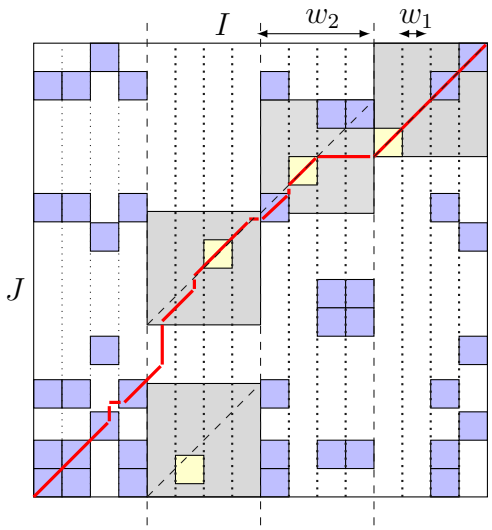


Figure 2.2: Illustration of the Covering Algorithm: Blue boxes are low cost boxes in dense w_1-strips, while the yellow ones are in sparse w_1-strips. The red line corresponds to the path τ that we are trying to cover. In each w_2-strip, τ is covered by either a collection of many w_1-boxes or it is covered by a diagonal extension of a low cost w_1-box. The various boxes might overlap vertically which is not shown in the picture.

Let $\mathcal{I}'$ denote the w_1-decomposition $\mathcal{I}_{w_1}(I')$ of I'. Every interval $I'' \in \mathcal{I}'$ has a θ-aligned τ-match $J^\tau(I'')$. It will be shown (see Proposition 15), that $\Delta_{\text{edit}}(I'', J^\tau(I'')) \leq 2\frac{\text{cost}(\tau_{I''})}{\mu(I'')} + \theta$. Let $u(I'')$ denote this upper bound. Consider the first alternative in the claim. During the dense case iteration $i = 0$, every interval is declared dense, and $(I'' \times J^\tau(I''), 5)$ is in $\mathcal{R}_D$ for all I''. To get an adequate approximation, we try to show that later iterations provide much better upper bounds on these boxes, i.e., $(I'' \times J^\tau(I''), \gamma(I'')) \in \mathcal{R}_D$ for a small enough value of $\gamma(I'')$. By definition of adequate approximation, it is enough that $\sum_{I'' \in \mathcal{I}'} \gamma(I'') \leq c \sum_{I'' \in \mathcal{I}'} u(I'')$, for some c. Let $t(I'')$ be the last (largest) iteration for which $\epsilon_{t(I'')} \geq u(I'')$ and $I'' \notin \mathcal{S}_{t(I'')}$ (which is well defined since $\mathcal{S}_0 = \emptyset$). Let $b(I'') = \epsilon_{t(I'')}$. Since $b(I'') \geq u(I'') \geq \Delta_{\text{edit}}(I'', J^\tau(I''))$, the box $(I'' \times J^\tau(I''), 5b(I''))$ is certified. The collection $\{(I'' \times J^\tau(I''), 5b(I''))\}$ is a sequence of certified boxes that satisfies the first two conditions for an adequate approximation of τ. The third condition will follow if:

$$\sum_{I'' \in \mathcal{I}'} 5b(I'') \leq c \sum_{I'' \in \mathcal{I}'} u(I'') \tag{2.1}$$

so this is sufficient to imply the first condition of the claim.

Next consider what we need for the second alternative to hold. Let $\mathcal{S}_i(I')$ be the set of intervals declared sparse in iteration i. An interval $I'' \in \mathcal{S}_i(I')$ is a *winner* (for iteration i) if $\Delta_{\text{edit}}(I'', J^\tau(I'')) \leq \epsilon_i$, and $\mathcal{W}_i(I')$ is the set of winners. In iteration i of the diagonal extension algorithm, we sample $\theta(\log^2 n)$ elements

of $\mathcal{S}_i(I')$. If for at least one iteration i our sample includes a winner I'' then the second condition of the claim will hold: $I'' \times J^\tau(I'')$ is extended diagonally to a w_2-box, and by the diagonal extension property, the extension is an adequate cover of $\tau_{I'}$, which we will certify with its exact edit distance.

Thus for the second alternative to fail with non-negligible probability:

$$\text{For all } i, |\mathcal{W}_i(I')| < |\mathcal{S}_i(I') - \mathcal{W}_i(I')|, \tag{2.2}$$

We argue that if the failure condition (2.2) holds, then the success condition (2.1) holds. Multiply (2.2) by ϵ_i and sum on i to get:

$$\sum_{I'' \in \mathcal{I}'} \sum_{i: I'' \in \mathcal{W}_i(I')} \epsilon_i < \sum_{I'' \in \mathcal{I}'} \sum_{i: I'' \in \mathcal{S}_i(I') - \mathcal{W}_i(I')} \epsilon_i. \tag{2.3}$$

For a given interval $I'' \in \mathcal{I}_{w_1}(I')$, consider the iterations i for which $I'' \in \mathcal{W}_i(I')$ and those for which $I'' \in \mathcal{S}_i(I') - \mathcal{W}_i(I')$. First of all if $\epsilon_i \geq u(I'')$ and $I'' \in \mathcal{S}_i(I')$ then since $\Delta_{\text{edit}}(I'', J^\tau(I'')) \leq u(I'') \leq \epsilon_i$ we conclude $I'' \in \mathcal{W}_i(I')$. So $I'' \in \mathcal{S}_i(I') - \mathcal{W}_i(I')$ implies that $\epsilon_i < u(I'')$, so the inner sum of the right side of (2.3) is at most $2u(I'')$ (by summing a geometric series).

Furthermore, for i with $u(I'') \leq \epsilon_i < b(I'')$, $I'' \in \mathcal{S}_i$ by the choice of $t(I'')$. Either $b(I'')/2 \leq u(I'')$ or $u(I'') < b(I'')/2$. The latter implies $I'' \in \mathcal{W}_{t(I'')+1}(I')$, and then $b(I'')/2$ is upper bounded by the inner sum on the left of (2.3). Therefore:

$$\begin{aligned}
\sum_{I''} b(I'') &\leq \sum_{I''} \left(2u(I'') + \sum_{i: I'' \in \mathcal{W}_i(I')} 2\epsilon_i \right) \\
&< \sum_{I''} \left(2u(I'') + 2 \sum_{i: I'' \in \mathcal{S}_i(I') - \mathcal{W}_i(I')} \epsilon_i \right) \\
&\leq 6 \sum_{I''} u(I''),
\end{aligned}$$

as required for (2.1).

This completes the overview of the covering algorithm.

2.3 Covering Algorithm: pseudo-code and analysis

In this section first we present the pseudo-code of the covering algorithm. Next we formally prove that each box output by the covering algorithm is correctly certified and they include an *adequate approximating sequence* for some min-cost path τ in $G_{x,y}$. Lastly we provide the formal run-time analysis of the covering algorithm.

2.3.1 Pseudo-code

The pseudo-code consists of *CoveringAlgorithm* which calls procedures *DenseStripRemoval* (the dense case algorithm) and *SparseStripExtensionSampling* (the diagonal extension algorithm). These are abbreviated, respectively by CA, DSR

and SSES. The technical differences between the pseudo-code and the informal description, are mainly to improve runtime analysis.

The parameters of CA are as described in the overview: x, y are input strings of length n, θ comes from **GAP-UB**$_\theta$, $w_1 < w_2 < n$ and $d < n$ are integral powers of 2, as are the auxiliary input parameters. The output is a set $\mathcal{R}$ of certified boxes. The algorithm uses global constants $c_0 \geq 0$ and $c_1 \geq 120$, where the former one is needed for Proposition 23.

We use a subroutine **SMALL-ED** which takes strings z_1, z_2 of length w and parameter κ and outputs ∞ if $\Delta_{\text{edit}}(z_1, z_2) > \kappa$ and otherwise outputs $\Delta_{\text{edit}}(z_1, z_2)$. The algorithm of [Ukk85] implements **SMALL-ED** in time $O(\kappa w^2)$.

One technical difference from the overview, is that the pseudo-code saves time by restricting the search for certified boxes to a portion of the grid close to the main diagonal. Recall that **GAP-UB**$_\theta$ has two requirements, that the output upper bounds $d_{\text{edit}}(x, y)$ (which will be guaranteed by the requirement that $\mathcal{R}$ contains no falsely certified boxes), and that if $d_{\text{edit}}(x, y) \leq \theta n$, the output is at most $c\theta n$ for some constant c. We therefore design our algorithm assuming $d_{\text{edit}}(x, y) \leq \theta n$, in which case every min-cost $G_{x,y}$-path τ consists entirely of points within $\frac{\theta}{2}n$ steps from the main diagonal, i.e. $|i-j| \leq \frac{\theta}{2}n$. So we restrict our search for certified boxes as follows: set $m = \frac{1}{4}\theta n$, and consider the $\frac{n}{m}$ overlapping equally spaced boxes of width $8m = 2\theta n$ lying along the main diagonal. Together these boxes cover all points within θn of the main diagonal.

The algorithm of the overview is executed separately on each of these n/m boxes. Within each of these executions, we iterate over $i \in \{0, \ldots, \log \frac{1}{\theta}\}$ (rather than $\{0, \ldots, \log n\}$ as in the overview). In each iteration we apply the dense case algorithm and the diagonal extension algorithm as in the overview. The output is the union over all n/m boxes and all iterations, of the boxes produced.

In the procedures DSR and SSES, the input G is an induced grid graph corresponding to a box $I_G \times J_G$, as described in the "framework" part of Section 2.1. The procedure DSR on input G, sets $\mathcal{T}$ to be the w_1-decomposition of I_G (the x-candidates) and $\mathcal{B}$ to be the set of $\frac{\epsilon_i}{8}$-aligned y-candidates. As in the overview, the dense case algorithm produces a set of certified boxes (called $\mathcal{R}_1$ in the pseudo-code) and a set $\mathcal{S}$ of intervals declared sparse.

SSES is invoked if $\mathcal{S} \neq \emptyset$ and iterates over all x-intervals I' in the decomposition $\mathcal{I}_{w_2}(I_G)$. The algorithm skips I' if $\mathcal{S}$ contains no subset of I', and otherwise selects a sample $\mathcal{H}$ of $\theta(\log^2 n)$ subintervals of I' from $\mathcal{S}$. For each sample interval I'' it finds the vertical candidates J'' for which $\Delta_{\text{edit}}(I'', J'') \leq \epsilon_i$, does a diagonal extension to I' and certifies each box with an exact edit distance computation.

There are a few parameter changes from the overview that provide some improvement in the time analysis: During each iteration i, rather than take our vertical candidates to be from a θ-aligned grid, we can afford a coarser grid that is $\epsilon_i/8$-aligned. Also, the local parameter d in DSR and SSES is set to d/ϵ_i during iteration i.

There is one counterintuitive quirk in SSES: each certified box is replicated $O(\log n)$ times with higher distance bounds. This is permissible (increasing the distance bound cannot decertify a box), but seems silly (why add the same box with a higher distance bound?). This is just a convenient technical device to ensure that the second phase min-cost path algorithm gives a good approximation.

Algorithm 1 CA$(x, y, n, w_1, w_2, d, \theta)$
CoveringAlgorithm

Input: Strings x, y of length n, $w_1, w_2, d \in [n]$, $w_1 < w_2 < \theta n/4$, and $\theta \in [0, 1]$. n, w_1, w_2, θ are powers of 2.

Output: A set $\mathcal{R}$ of certified boxes in G.

Initialization: $G = G_{x,y}$, $\mathcal{R}_D = \mathcal{R}_E = \emptyset$.
Let $m = \frac{\theta n}{4}$
for $k = 0, \ldots, \frac{4}{\theta}$ **do**
  Let $I = J = \{km, km + 1, \ldots, (k + 8)m\}$.
  for $i = \lceil \log 1/\theta \rceil, \ldots, 0$ **do**
    Set $\epsilon_i = 2^{-i}$.
    Invoke DSR$(G(I \times J), n, w_1, \frac{d}{\epsilon_i}, \frac{\epsilon_i}{8}, \epsilon_i)$ to get $\mathcal{S}$ and $\mathcal{R}_1$.
    if $\mathcal{S} \neq \emptyset$ **then**
      Invoke SSES$(G(I \times J), \mathcal{S}, n, w_1, w_2, \frac{d}{\epsilon_i}, \frac{\epsilon_i}{8}, \epsilon_i, \theta)$ to get $\mathcal{R}_2$.
    else
      $\mathcal{R}_2 = \emptyset$.
    end if
    Add items from $\mathcal{R}_1$ to $\mathcal{R}_D$ and from $\mathcal{R}_2$ to $\mathcal{R}_E$.
  end for
end for
Output $\mathcal{R} = \mathcal{R}_D \cup \mathcal{R}_E$.

Algorithm 2 DSR$(G, n, w, d, \delta, \epsilon)$
DenseStripRemoval

Input: $G = G_{x,y}(I_G \times J_G)$ for some $I_G, J_G \subseteq \{0, 1, \ldots, n\}$, $w, d \in [n]$, the endpoints of I_G and J_G are multiples of w and $\delta, \epsilon \in [0, 1]$.

Output: Set $\mathcal{S}$ which is a subset of the w-decomposition of I_G and a set $\mathcal{R}$ of δ-aligned certified w-boxes all with distance bound $5\epsilon_i$.

Initialization: $\mathcal{S} = \mathcal{R} = \emptyset$. $\mathcal{T} = \mathcal{I}_w(I_G)$.
$\mathcal{B}$, the set of y-candidates, is the set of width w δ-aligned subintervals of J_G (having endpoints a multiple of δw.)
while $\mathcal{T}$ is non-empty **do**
  Pick $I \in \mathcal{T}$
  Sample $c_0|\mathcal{B}|\frac{1}{d}\log n$ intervals $J \in \mathcal{B}$ uniformly at random and for each test if $\Delta_{\text{edit}}(x_I, y_J) \leq \epsilon$.
  if for at most $\frac{c_0}{2}\log n$ sampled J's, **SMALL-ED**$(x_I, y_J, \epsilon) < \infty$ **then**
    $\mathcal{S} = \mathcal{S} \cup \{I\}$; $\mathcal{T} = \mathcal{T} - \{I\}$. (*$I$ is declared sparse*)
  else
    (*I is declared dense and used as a pivot*)
    Compute:
    $\mathcal{Y} = \{J \in \mathcal{B};\ \textbf{SMALL-ED}(x_I, y_J, 3\epsilon) < \infty\}$.
    $\mathcal{X} = \{I' \in \mathcal{T};\ \textbf{SMALL-ED}(x_I, x_{I'}, 2\epsilon) < \infty\}$.
    Add $(I' \times J', 5\epsilon)$ to $\mathcal{R}$ for all pairs $(I', J') \in \mathcal{X} \times \mathcal{Y}$.
    $\mathcal{T} = \mathcal{T} - \mathcal{X}$.
  end if
end while
Output $\mathcal{S}$ and $\mathcal{R}$.

Algorithm 3 SSES($G, \mathcal{S}, n, w_1, w_2, d, \delta, \epsilon, \theta$)
SparseStripExtensionSampling

Input: $G = G_{x,y}(I_G, J_G)$ with $I_G, J_G \subseteq \{0, 1, \ldots, n\}$, w_1, w_2, d, n are powers of 2, with $w_1, w_2, d < n$ and $w_1 < w_2$. Endpoints of I_G and J_G are multiples of w_2, $\mathcal{S}$ is a subset of the w_1-decomposition of I_G and δ, ϵ, θ are non-positive integral powers of 2.

Output: A set $\mathcal{R}$ of certified w_2-boxes in G.

Initialization: $\mathcal{R} = \emptyset$.
$\mathcal{B}$, the set of y-candidates, is the set of width w δ-aligned subintervals of J_G (endpoints are multiples of δw.)
for $I' \in \mathcal{I}_{w_2}(I_G)$ **do**
  if $\mathcal{S}$ includes a subset of I' **then**
    Select $c_1 \log^2 n$ intervals $I \in \mathcal{S}$ independently and uniformly at random from $\mathcal{I}_{w_1}(I') \cap \mathcal{S}$, to obtain $\mathcal{H}$.
    for each $I \in \mathcal{H}$ and each $J \in \mathcal{B}$ **do**
      if SMALL-ED$(x_I, y_J, \epsilon) < \infty$ **then**
        Let J' be such that $I' \times J'$ is the diagonal extension of $I \times J$ in $I' \times J_G$.
        Let $p =$ **SMALL-ED**$(x_{I'}, y_{J'}, 3\epsilon)$
        if $p < \infty$ **then**
          For $k = 0, \ldots, \log n$, add $(I' \times J', p + \theta + 2^{-k})$ to $\mathcal{R}$.
        end if
      end if
    end for
  end if
end for
Output $\mathcal{R}$.

2.3.2 Analysis and correctness of CA

For the analysis we must prove that $\mathcal{R}$ contains an "adequate approximation" of some min-cost alignment path τ. To state this precisely, we start with definitions and observations and prove some simple preliminary claims that formalize intuitive notions from the overview.

Cost and normalized cost. The *cost* of a path τ, $\text{cost}(\tau)$, from (u_1, u_2) to (v_1, v_2) in a grid-graph (see Section 2.1), is the sum of the edge costs, and the *normalized cost* is $\text{ncost}(\tau) = \frac{\text{cost}(\tau)}{v_1 - u_1}$. $\text{cost}(G(I \times J))$ (or simply $\text{cost}(I \times J)$), the *cost of subgraph* $G(I \times J)$, is the min-cost of a path from the lower left to the upper right corner. The *normalized cost* is $\text{ncost}(I \times J) = \frac{1}{\mu(I)}\text{cost}(I \times J)$.

We note the following simple fact without proof:

Proposition 11. *For $I, J, J' \subseteq \{0, \ldots, n\}$, $|d_{edit}(x_I, y_J) - d_{edit}(x_I, y_{J'})| \leq |J \Delta J'|$, where Δ denotes symmetric difference.*

Projections and subpaths. The *horizontal projection* of a path $\tau = (i_1, j_1), \ldots, (i_\ell, j_\ell)$ is the set of $\{i_1, \ldots, i_\ell\}$. We say that τ *crosses box* $I \times J$ if the vertices of τ belong to $I \times J$ and its horizontal projection is I. If the horizontal projection of τ contains I', $\tau_{I'}$ denotes the (unique) minimal subpath of τ whose projection is I'.

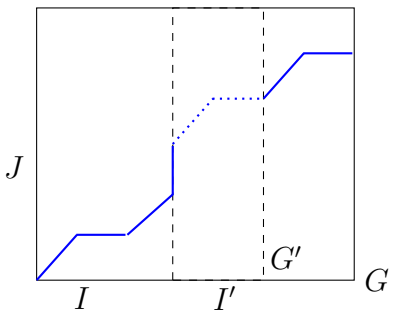


Figure 2.3: Illustration of a path that crosses a grid: Solid blue edges are the edges of a given path τ. Dotted blue edges are the edges of path $\tau_{I'}$ that crosses the dashed strip G'.

Proposition 12. *Let τ be a path with horizontal projection I, and let $I_1, \ldots, I_\ell$ be a decomposition of I. Then the τ_{I_j} are edge-disjoint and so:*

$$\begin{aligned} cost(\tau) &\geq \sum_{i=1}^{\ell} cost(\tau_{I_i}) \\ ncost(\tau) &\geq \sum_{i=1}^{\ell} \frac{\mu(I_i)}{\mu(I)} ncost(\tau_{I_i}). \end{aligned}$$

Definition 13. $(1-\delta)$**-cover.** *Let τ be a path with horizontal projection I and let $I' \times J'$ be a (not necessarily square) box with $I' \subseteq I$. For $\delta \in [0, 1]$ the box $I' \times J'$ $(1-\delta)$-covers τ if the initial, resp. final, vertex of the subpath $\tau_{I'}$ is within $\delta\mu(I')$ vertical units of $(\min(I'), \min(J'))$, resp. $(\max(I'), \max(J'))$.*

Proposition 14. *Let $I' \times J'$ be a (not necessarily square) box that $(1-\delta)$-covers path τ.*

1. *ncost$(I' \times J') \leq$ ncost$(\tau_{I'}) + 2\delta$.*
2. *If J'' is any vertical interval, then $I' \times J''$ $(1 - \delta - |J'\Delta J''|/\mu(I'))$ covers τ.*

Proof. For the first part, let J^0 be the vertical projection of $\tau_{I'}$. Then ncost$(I' \times J^0) \leq$ ncost$(\tau_{I'})$ since $\tau_{I'}$ joins the lower left corner of $I' \times J^0$ to the upper right corner. Since $I' \times J'$ $(1-\delta)$-covers τ, $|J'\Delta J^0| \leq 2\delta\mu(I')$, and by Proposition 11, ncost$(I' \times J') \leq$ ncost$(\tau_{I'}) + 2\delta$.

For the second part, observe that the vertical distance between the lower (resp. upper) corners of $I' \times J'$ and $I' \times J''$ is at most $|J'\Delta J''|$. □

δ-aligned interval. A y-interval J of width w is *δ-aligned* for $\delta \in (0,1)$ if its endpoints are multiples of δw (which we require to be an integer). (See Figure 2.4)

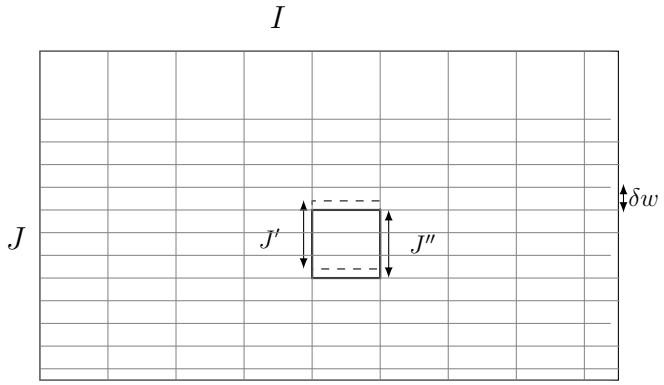


Figure 2.4: An illustration of δ-aligned interval. Here interval J'' is δ-aligned but J' is not.

Proposition 15. *Let τ be a path that crosses $I \times J$. Suppose that $I' \subseteq I$ has width w, and $\mu(J) \geq w$.*

1. *There is an interval J^1 with $\mu(J^1) = \mu(I')$ so that ncost$(I' \times J^1) \leq$ 2ncost$(\tau_{I'})$ and $I' \times J^1$ $(1 -$ ncost$(\tau_{I'}))$-covers τ.*
2. *There is a δ-aligned interval $J' \subseteq J$ of width w so that ncost$(I' \times J') \leq$ 2ncost$(\tau_{I'}) + \delta$ and $I' \times J'$ $(1 -$ ncost$(\tau_{I'}) - \delta)$-covers τ.*

(J^1, J' are "τ-matches" for I', in the sense of the overview.)

Proof. Let $\tau' = \tau_{I'}$ be the min-cost subpath of τ that projects to I'. Let J^0 be the vertical projection of τ'. Note that $|\mu(J^0) - \mu(I')| \leq$ cost(τ'). Arbitrarily choose an interval J^1 of width $\mu(I')$ that either contains or is contained in J^0. Then $|J^0\Delta J^1| = |\mu(J^0) - \mu(I')| \leq$ cost(τ'), so by Proposition 11 ncost$(I' \times J^1) \leq$ 2ncost(τ'). Furthermore $I' \times J^1$ $(1 -$ ncost$(\tau'))$ covers τ'. Let J' be the closest δ-aligned interval to J^1, so $|J'\Delta J^1| \leq \delta\mu(I')$ and so ncost$(I' \times J') \leq$ ncost$(I' \times J^1) + \delta \leq$ 2ncost$(\tau') + \delta$. Finally since $I' \times J'$ is a vertical shift of $I' \times J^1$ of normalized length at most δ, we have $I' \times J'$ $(1 -$ ncost$(\tau') - \delta)$ covers τ'. □

Definition 16. *1. The* main diagonal *of a box is the segment joining the lower left and upper right corners.*

2. For a square box $I' \times J'$, and $I' \subseteq I$, the true diagonal extension *of $I' \times J'$ to I is the square box $I \times \hat{J}$ whose main diagonal contains the main diagonal of $I' \times J'$.*

3. For a w-box $I' \times J'$ contained in strip $I \times J$, the adjusted diagonal extension of $I' \times J'$ within $I \times J$ *is the box $I \times J''$ obtained from the true diagonal extension of $I' \times J'$ to I by the minimal vertical shift so that it is a subset of $I \times J$. (The adjusted diagonal extension is the true diagonal extension if the true diagonal extension is contained in $I \times J$; otherwise it's lower edge is $\min(J)$ or its upper edge is $\max(J)$.)*

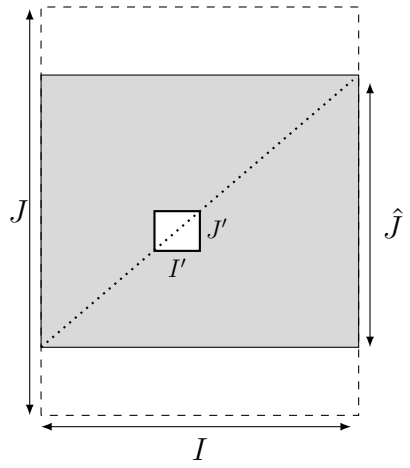


Figure 2.5: Illustration of diagonal extension: Given a w-box $I' \times J'$ its true diagonal extension is the grey box $I \times \hat{J}$.

Proposition 17. *Suppose path τ crosses $I \times J$ and $\text{ncost}(\tau_I) \leq \epsilon$. Let $w = \mu(I)$. Let $I' \times J'$ be a w'-box that $(1-\delta)$-covers $\tau_{I'}$. Then the adjusted diagonal extension $I \times J''$ of $I' \times J'$ within $I \times J$ $(1-(\epsilon+\delta\frac{w'}{w}))$-covers τ and satisfies $\text{ncost}(I \times J'') \leq 3\epsilon + 2\delta\frac{w'}{w}$.*

Proof. It suffices to show that $I \times J''$ $(1 - (\epsilon + \delta w'/w))$-covers τ, since then Proposition 14 gives us the needed upper bound on $\text{ncost}(I \times J'')$.
Case 1. $I \times J''$ is equal to the true diagonal extension. If $\epsilon \geq 1$, the claim follows trivially, so we can assume $\epsilon < 1$. Let τ_I, $\tau_{I'}$ be the min-cost subpath of τ that projects on I and I' respectively.

We will give an upper bound on the vertical distance from the final vertex of τ to the upper right corner of $I \times J''$. Let τ_u be the subpath of τ that starts at the final vertex of $\tau_{I'}$ and ends at the final vertex of τ_I. Let I_u and J_u be the horizontal and vertical projections of τ_u. The start vertex of τ_u has vertical distance at most $\delta w'$ from the main diagonal of $I \times J''$. The final vertex of τ_u therefore has vertical distance at most $\delta w' + |\mu(I_u) - \mu(J_u)|$ from the upper corner of $I \times J''$, and this is at most $\delta w' + \epsilon w$, since $\text{cost}(\tau) \geq |\mu(I_u) - \mu(J_u)|$. A similar argument gives

the same upper bound on the vertical distance between the start vertex of τ_I and the lower left corner of $I \times J''$, so $G''(I \times J'')$ $(1-(\epsilon+\delta w'/w))$-covers τ.
Case 2. $I \times J''$ is not the true diagonal extension. Extend the set J to $\bar{J}$ by adding $\mu(I)$ elements before and after. (It is possible that $\bar{J}$ is not a subset of $\{0,\ldots,n\}$; in this case we imagine that y is extended to a sequence y^* by adding $\mu(I)$ new symbols to the beginning and end of y and that we are in the grid graph G_{x,y^*}.) Let $I \times J'''$ be the adjusted diagonal extension of $I' \times J'$ to $I \times \bar{J}$. This is equal to the true diagonal extension, and so by Case 1, $I \times J'''$ $(1-(\epsilon+\delta w'/w))$-covers τ. We claim that $I \times J''$ does also. Assume J''' falls below $\min(J)$ (the case that J''' is above $\max(J)$ is similar). Then $I \times J''$ is obtained by shifting $I \times J'''$ up until the lower edge coincides with $\min(J)$. The lower vertex of τ_I has y-coordinate at least $\min(J)$.

If the y-coordinate of the upper vertex of τ_I is at most $\max(J'')$, then J'' contains vertical projection of τ_I, and $I \times J''$ $(1-\epsilon)$-covers τ. If the y-coordinate of the upper vertex of τ_I is greater than $\max(J'')$, shifting $I \times J'''$ up to $I \times J''$ can only decrease the vertical distance from the the lower left corner to the start of τ_I and from the upper corner to the end of τ_I, so $I \times J''$ $(1-(\epsilon+\delta w'/w))$-covers τ. □

(k,ζ)-approximation of a path. This formalizes the notion of adequate approximation of a path by a certified box sequence.

Definition 18. *Let G be a grid graph on $I \times J$. Let $\zeta, \epsilon \in [0,1]$. Let τ be a path that crosses G. A sequence of certified boxes $\sigma = \{(I_1 \times J_1, \epsilon_1), (I_2 \times J_2, \epsilon_2), \ldots, (I_\ell \times J_\ell, \epsilon_\ell)\}$* (k,ζ)-approximates *τ provided that:*

1. *$I_1, \ldots, I_\ell$ is a decomposition of I.*
2. *For each $i \in [\ell]$, $I_i \times J_i$ $(1-\epsilon_i)$-covers τ.*
3. *$\sum_{i\in[\ell]} \epsilon_i \mu(I_i) \le (k \cdot ncost(\tau) + \zeta)\mu(I)$.*

Proposition 19. *Suppose path τ crosses $I \times J$ and $I_1, \ldots, I_m$ is a decomposition of I, and for $i \in [m]$, σ_i is a certified box sequence that (k,ζ)-approximates τ_{I_i}. Then $\sigma_1, \ldots, \sigma_m$ (k,ζ)-approximates τ.*

Proof. It is obvious that σ is a sequence of certified boxes, that the horizontal projections of all the boxes form a decomposition of I and that each box (I_i, J_i, ϵ_i) $(1-\epsilon_i)$-covers τ. The final condition is verified by splitting the sum on the left into m sums where the jth sum includes terms for $I_i \subseteq I_j$, and is bounded above by $(k \cdot \mathrm{ncost}(\tau_{I_j} + \zeta)\mu(I_j)$. Summing the latter sum over j and using Proposition 12 we get that σ (k,ζ)-approximates the path τ.

□

Definition 20. **(d,δ,ϵ)-dense and -sparse.** *Fix a box $I \times J$. An interval $I' \subseteq I$ of width w is* (d,δ,ϵ)-sparse *(wrt $I \times J$) for integer d and $\epsilon, \delta \in (0,1]$ if there are at most d δ-aligned w-boxes in $I' \times J$ of ncost at most ϵ, and is* (d,δ,ϵ)-dense *otherwise.*

The sets $\mathcal{S}_i$ and $\mathcal{S}_i(I')$. For fixed k in the outer loop of CA, the set $\mathcal{S}$ created in iteration i of CA is denoted by $\mathcal{S}_i$. For any interval I', $\mathcal{S}_i(I')$ is the set of subintervals of I' belonging to $\mathcal{S}_i$.

Successful Sampling. The algorithm uses random sampling in two places, in the i loop inside CA and within the conditional on $\mathcal{S}$ containing a set from $\mathcal{I}_{w_1}(I')$ in SSES. We now specify what we need from the random sampling.

Definition 21. *A run of the algorithm has* successful sampling *provided that for all $k \in \{0, \ldots, 4/\theta\}$ and $i \in \{0, \ldots, \log \frac{1}{\theta}\}$ in the nested CA loops:*

- *For every w_1 interval I with endpoints a multiple of w_1, if I is $(\frac{d}{\epsilon_i}, \frac{\epsilon_i}{8}, \epsilon_i)$-dense interval (in terms of global parameters), DSR does not assign I to $\mathcal{S}$ and if I is $(\frac{d}{4\epsilon_i}, \frac{\epsilon_i}{8}, \epsilon_i)$-sparse, DSR places I in $\mathcal{S}$.*
- *On all calls to SSES, for every w_2 interval I with endpoints a multiple of w_2, if $|\mathcal{W}_i(I)|$ has size at least $|\mathcal{S}_i(I) - \mathcal{W}_i(I)|/32$ then the sample $\mathcal{H}$ selected contains an element of $\mathcal{W}_i(I)$. (Here $\mathcal{S}_i(I)$ and $\mathcal{W}_i(I)$ are that defined in the proof of Claim 4, whose definitions don't depend on the randomness used to select $\mathcal{H}$.)*

We will need the following variant of the Chernoff bound.

Proposition 22 (Chernoff bound)**.** *There is a constant c_0 such that the following is true. Let $1 \leq d \leq n$ be integers, B be a set and $E \subseteq B$. Let us sample $c_0 \frac{|B|}{d} \log n$ samples from B independently at random with replacement.*

1. *If $|E| \geq d$ then the probability that less than $\frac{c_0}{2} \log n$ samples are from E is at most $1/n^{10}$.*
2. *If $|E| \leq d/4$ then the probability that at least $\frac{c_0}{2} \log n$ samples are from E is at most $1/n^{10}$.*

Proposition 23. *For large enough n, a run of CA has* successful sampling *with probability at least $1 - n^{-7}$.*

Proof. By Proposition 22, the probability that the first condition fails for a particular k, i, I is at most n^{-10}. The number of choices for k, i, I is at most $(\frac{4}{\theta} + 1) \cdot (1 + \log \frac{1}{\theta}) \frac{n}{w_1} \leq n^2$ (for large enough n) so the overall probability that (1) fails is at most n^{-8}.

The probability that the second condition fails for a particular k, i, I is $(1 - \frac{1}{32})^{c_1 \log^2 n} \leq n^{-10}$. The number of k, i, I is less than n^2 (for large enough n), so the overall failure probability is at most n^{-8} . □

We assume that coins are fixed in a way that gives successful sampling.

Correctness of the covering algorithm:

The main property of CA to be proved is:

Theorem 24. *Let x, y be strings of length n, $1/n \leq \theta \leq 1$ be a real. Let w_1, w_2, d satisfy $w_1 \leq \theta w_2$, $w_2 \leq \frac{\theta n}{4}$ and $1 \leq d \leq \frac{\theta n}{w_1}$. Assume n, w_1, w_2, d, θ are powers of 2. Let $\mathcal{R}$ be the set of weighted boxes obtained by running CA$(x, y, n, w_1, w_2, d, \theta)$ with $c_1 > 120$. Then (1) Every $(I \times J, \epsilon) \in \mathcal{R}$ is correctly certified, i.e., $\Delta_{edit}(x_I, y_J) \leq \epsilon$, and (2) In a run that satisfies successful sampling, for every path τ from the source to the sink in $G = G_{x,y}$ of cost at most θ there is a subset of $\mathcal{R}$ that $(45, 15\theta)$-approximates τ.*

Proof. All boxes output are correctly certified: Each box in $\mathcal{R}_E$ comes from SSES which only certifies boxes with at least their exact edit distance. For $(I \times J, \epsilon) \in \mathcal{R}_D$, there must be an I' such that $\Delta_{\text{edit}}(x_{I'}, y_J) \leq \frac{3}{5} \cdot \epsilon$ and $\Delta_{\text{edit}}(x_{I'}, x_I) \leq \frac{2}{5} \cdot \epsilon$ and so by triangle inequality $\Delta_{\text{edit}}(x_I, y_J) \leq \epsilon$.

It remains to establish (2). Fix a source-sink path τ of normalized cost κ.

By Proposition 19 it is enough to show that for each $I' \in \mathcal{I}_{w_2}$, $\mathcal{R}$ contains a box sequence that $(45, 15\theta)$-approximates $\tau_{I'}$. So we fix $I' \in \mathcal{I}_{w_2}$.

The main loop (on k) of CA processes G in overlapping boxes. Since $\text{ncost}(\tau) \leq \theta$, one of these boxes, which we'll call $I \times J$, must contain $\tau_{I'}$.

Claim 2. *Let $I' \in \mathcal{I}_{w_2}$. There exist intervals $I, J \subseteq \mathbb{N}$, $I = J$ that are enumerated in the main loop of CA such that $I' \subseteq I$ and $\tau_{I'}$ crosses $G(I' \times J)$.*

Proof. Since τ is of cost at most θ, it cannot use more than $\theta n/2$ horizontal edges as for each horizontal edge of cost 1, it must use one vertical edge of cost 1. Similarly for vertical edges. So τ is confined to diagonals $\{-\theta n/2, \ldots, 0, \ldots, \theta n/2\}$ of G. By the choice of m in CA, there will be I and J considered in the main loop of the algorithm such that $I' \subseteq I$ and $\tau_{I'}$ crosses $G(I \times J)$. In particular, $I = J = \{km, km + 1, \ldots, (k + 8)m\}$, where k is the largest integer such that $km \leq \min(I') - \frac{\theta n}{2}$ has the desired property. □

Let I, J be as provided by the claim. Let $\mathcal{I}'$ be the w_1-decomposition of I'. We will show one of the following must hold: (1) $\mathcal{R}_D$ contains a sequence of certified w_1-boxes that $(45, 15\theta)$-approximates $\tau_{I'}$, or (2) There is a single certified w_2-box in $\mathcal{R}_E$ that $(45, 15\theta)$-approximates $\tau_{I'}$.

Let $t = \log \frac{1}{\theta}$. For $i = t, \ldots, 0$, let $\epsilon_i = 2^{-i}$ and let $\mathcal{S}_i$ be the set $\mathcal{S}$ obtained at the iteration i of $\text{CA}(x, y, n, w_1, w_2, d, \theta)$.

We note:

Claim 3. *Let $i \in \{0, \ldots, \log 1/\theta\}$. Suppose $I'' \in \mathcal{I}_{w_1}(I)$ and $J'' \subseteq J$ is $\epsilon_i/8$-aligned. If $I'' \notin \mathcal{S}_i$ and cost$(I'' \times J'') \leq \epsilon_i$ then $(I'' \times J'', 5\epsilon_i) \in \mathcal{R}_D$.*

Proof. If $I'' \notin \mathcal{S}_i$ then in the call to $\text{DSR}(G(I \times J), n, w_1, d/\epsilon_i, \epsilon_i/8, \epsilon_i)$ there is an iteration of the main loop,where the selected interval $\tilde{I}$ from $\mathcal{T}$ is declared dense and $\Delta_{\text{edit}}(x_{\tilde{I}}, x_{I''}) \leq 2\epsilon_i$. Since $\Delta_{\text{edit}}(x_{I''}, y_{J''}) \leq \epsilon_i$, $\Delta_{\text{edit}}(x_{\tilde{I}}, y_{J''}) \leq 3\epsilon_i$ and so $I'' \in \mathcal{X}$ and $J'' \in \mathcal{Y}$. Thus, DSR certifies $(I'' \times J'', 5\epsilon_i)$, which is added to $\mathcal{R}_D$. □

The theorem follows from:

Claim 4. *For an interval $I' \in \mathcal{I}_{w_2}$, assuming successful sampling $\mathcal{R}_E$ or $\mathcal{R}_D$ contains a $(45, 15\theta)$-approximation of $\tau_{I'}$.*

The proof is similar to that of Claim 1, with adjustments for some technicalities.

Proof. Let $\tau' = \tau_{I'}$ and $\kappa = \text{ncost}(\tau')$. Let $\mathcal{I}' = \mathcal{I}_{w_1}(I')$. For $I'' \in \mathcal{I}'$, let $\kappa_{I''} = \text{ncost}(\tau_{I''})$. By Proposition 15, for all $I'' \in \mathcal{I}'$ and $\epsilon_i \geq \kappa_{I''}$ there is an $\epsilon_i/8$-aligned vertical interval $J_i^\tau(I'')$, such that $\text{ncost}(I'' \times J_i^\tau(I'')) \leq 2\kappa_{I''} + \epsilon_i/8$ and $I'' \times J_i^\tau(I'')$ $(1 - \kappa_{I''} - \epsilon_i/8)$-covers $\tau_{I'}$.

Let $s(I'')$ be the largest integer such that $\epsilon_{s(I'')} \geq 3\kappa_{I''} + \kappa + \theta$. Let $t(I'') \leq s(I'')$ be the largest integer such that $I'' \notin \mathcal{S}_{t(I'')}$. (Since $\theta n/w_1 \geq d$, $\mathcal{S}_0 = \emptyset$, so $t(I'')$ is well-defined.) Let $a(I'') = \epsilon_{s(I'')}$ (this plays a similar role to $u(I'')$ in Section 2.2) and $b(I'') = \epsilon_{t(I'')}$.

For all $\epsilon_i \in [a(I''), b(I'')]$, $\text{ncost}(I'' \times J_i^\tau(I'')) \leq \epsilon_i$ and $I'' \times J_i^\tau(I'')$ $(1-\epsilon_i)$-covers τ'. By the definition of $b(I'')$ and Claim 3, $\mathcal{R}_D$ contains the certified box $(I'' \times J^\tau_{t(I'')}(I''), 5b_{I''})$. So $\mathcal{R}_D$ contains a $(45, 15\theta)$-approximation of τ' provided that:

$$\sum_{I'' \in \mathcal{I}'} 5b(I'') \leq \frac{45}{8} \sum_{I'' \in \mathcal{I}'} a(I'') \tag{2.4}$$

since $a(I'') \leq 2(3\kappa_{I''} + \kappa + \theta)$.

Next we determine a sufficient condition that $\mathcal{R}_E$ contain a box sequence (consisting of a single box) that $(5, 4\theta)$-approximates τ'. Let $\mathcal{S}_i(I') = \mathcal{S}_i \cap \mathcal{I}'$. Interval $I'' \in \mathcal{S}_i(I')$ is a *winner for iteration* i if $\epsilon_i \geq a(I'')$. This set of winners is denoted by $\mathcal{W}_i(I')$. It suffices that during iteration i, the set of $c_1 \log^2 n$ samples taken in SSES includes a winner I''; then since $\Delta_{\text{edit}}(I'', J_i^\tau(I'')) \leq \epsilon_i$, the (adjusted) diagonal extension $I' \times \tilde{J}$ of $I'' \times J_i^\tau(I'')$ will be certified. By Proposition 17, $I' \times \tilde{J}$ has normalized cost at most $3\kappa + 2\epsilon_i w_1/w_2 \leq 3\kappa + 2\theta \leq 3\epsilon_i$ and it $(1-(\kappa+\theta))$-covers τ'. If $\kappa = 0$ then $(I' \times \tilde{J}, \text{ncost}(I' \times \tilde{J}) + \theta + 2^{-\log n})$ is in $\mathcal{R}_E$ by the behavior of SSES and it $(5, 4\theta)$-approximates τ'. Otherwise $\kappa \geq 1/n$; so set $k = \lfloor \log 1/\kappa \rfloor$. Thus, $k \leq \log n$ and $2^{-k} \in [\kappa, 2\kappa)$. Then $(I' \times \tilde{J}, \text{ncost}(I' \times \tilde{J}) + \theta + 2^{-k})$ is in $\mathcal{R}_E$ and it $(5, 4\theta)$-approximates τ'.

Under successful sampling if $|\mathcal{W}_i(I')| \geq \frac{1}{32}|\mathcal{S}_i(I') - \mathcal{W}_i(I')|$, at least one interval from $\mathcal{W}_i(I')$ will be included in our $c_1 \log^2 n$ samples during SSES and $\mathcal{R}_E$ will contain a $(5, 4\theta)$-approximation of τ' as above. So suppose this fails:

$$\text{For all } i,\ |\mathcal{W}_i(I')| < \frac{1}{32}|\mathcal{S}_i(I') - \mathcal{W}_i(I')|. \tag{2.5}$$

We show that this implies (2.4). Multiplying (2.5) by ϵ_i and summing on i yields:

$$\sum_{I'' \in \mathcal{I}'} \sum_{i: I'' \in \mathcal{W}_i(I')} \epsilon_i < \frac{1}{32} \sum_{I'' \in \mathcal{I}'} \sum_{i: I'' \in \mathcal{S}_i(I') - \mathcal{W}_i(I')} \epsilon_i. \tag{2.6}$$

$I'' \in \mathcal{S}_i(I') - \mathcal{W}_i(I')$ implies $\epsilon_i < a(I'')$. Summing the geometric series:

$$\sum_{i: I'' \in \mathcal{S}_i(I') - \mathcal{W}_i(I')} \epsilon_i \leq 2a(I''). \tag{2.7}$$

Either $a(I'') = b(I'')$ or $a(I'') < b(I'')$. If the latter, then $I'' \in \mathcal{W}_i(I')$ for $\epsilon_i = b(I'')/2$. So:

$$\begin{aligned}
\sum_{I'' \in \mathcal{I}'} b(I'') &\leq \sum_{I''} \left(a(I'') + \sum_{i: I'' \in \mathcal{W}_i(I')} 2\epsilon_i \right) \\
&< \sum_{I''} \left(a(I'') + \frac{1}{16} \sum_{i: I'' \in \mathcal{S}_i(I') - \mathcal{W}_i(I')} \epsilon_i \right) \\
&\leq \frac{9}{8} \sum_{I'' \in \mathcal{I}'} a(I'')
\end{aligned}$$

which implies Equation 2.4. (The second inequality follows from (2.6) and the last inequality from (2.7).) □

□

2.3.3 Time complexity of CA

We write $t(w, \epsilon)$ for the time of **SMALL-ED**(z_1, z_2, ϵ) on strings of length w. We assume $t(w, \epsilon) \geq w$, and that for $k \geq 1$, there is a constant $c(k)$ such that for all $\epsilon \in [0, 1]$ and all $w > 1$, $t(w, k\epsilon) \leq c(k) \cdot t(w, \epsilon) + c(k)$. As mentioned earlier, by [Ukk85], we can use $t(w, \epsilon) = O(w^2\epsilon)$.

Theorem 25. *Let n be a sufficiently large power of 2 and $\theta \in [1/n, 1]$ be a power of 2. Let x, y be strings of length n. Let $\log n \leq w_1 \leq w_2 \leq \theta n/4$, $1 \leq d \leq n$ be powers of 2, where $w_1 | w_2$ and $w_2 | n$, and $w_1/w_2 \leq \theta$. The size of the set $\mathcal{R}$ output by CA is $O((\frac{n}{w_1})^2 \log^2 n)$ and in any run that satisfies successful sampling, CA runs in time:*

$$O\left(|\mathcal{R}| + \sum_{\substack{k=\log 1/\theta,\ldots,0 \\ \epsilon=2^{-k}}} \left(\frac{\theta n^2 \log n}{d\epsilon w_1^2} \cdot t(w_1, \epsilon) + \frac{\theta n^2 \log^2 n}{w_1 w_2 \epsilon} \cdot t(w_1, \epsilon) + \frac{nd \log^2 n}{w_2 \epsilon} \cdot t(w_2, \epsilon)\right)\right)$$

Proof. To bound $|\mathcal{R}|$ note that for each choice of k, i in the outer and inner loops of CA, the set of candidate boxes of width w_1 has size $O(\frac{\theta n}{w_1} \frac{\theta n}{w_1 \epsilon_i})$. This upper bounds the number of boxes certified by DSR. The call to SSES constructs at most one diagonal extension for each such candidate box, and each diagonal extension gives rise to at most $O(\log n)$ certified boxes. Thus, for each (k, i) there are $O(\frac{\theta^2 n^2 \log n}{(w_1)^2 \epsilon_i})$ certified boxes. Summing the geometric series over i, noting that $\min(\epsilon_i) = \theta$, and summing over $O(1/\theta)$ values of k gives the required bound on $|\mathcal{R}|$.

The steps in the algorithm that actually construct certified boxes (13 of DSR, 11 of SSES, 13 of CA) cost $O(1)$ per box giving the first term in the time bound.

We next bound the other contributions to runtime. The outer loop of CA has $\frac{4}{\theta} + 1$ iterations on k's. The inner loop has $1 + \log \frac{1}{\theta}$ iterations on i. Each iteration invokes DSR and SSES on $I \times J$ with I and J of width at most $4\theta n$.

We bound the time of a call to DSR. To distinguish between local variables of DSR and global variables of CA, we denote local input variables as $\hat{G}, \hat{n}, \hat{w}, \hat{d}, \hat{\delta}, \hat{\epsilon}$. For $\mathcal{B}$ and $\mathcal{T}$ as in DSR, $|\mathcal{B}| \leq \frac{\mu(I_{\hat{G}})}{\hat{\delta}\hat{w}}$. since $\mu(I_{\hat{G}}) = \mu(J_{\hat{G}})$. The main while loop of DSR repeatedly picks intervals $I \in \mathcal{T}$ and samples $c_0 |\mathcal{B}| \frac{\log \hat{n}}{\hat{d}} \leq \frac{c_0 \mu(I_{\hat{G}}) \log \hat{n}}{\hat{d}\hat{\delta}\hat{w}}$ vertical intervals J and tests whether $\Delta_{\text{edit}}(x_I, y_J) \leq \hat{\epsilon}$. Each such test takes time $t(\hat{w}, \hat{\epsilon})$. This is done at most once for each of the $\mu(I_{\hat{G}})/\hat{w}$ horizontal candidates for a total time of $O(\frac{\mu(I_{\hat{G}})^2 \log \hat{n}}{\hat{w}^2 \hat{\delta}\hat{d}}) t(\hat{w}, \hat{\epsilon})$. We next bound the cost of processing a pivot I. This requires testing $\Delta_{\text{edit}}(x_I, y_J) \leq 3\hat{\epsilon}$ for $J \in \mathcal{B}$ and $\Delta_{\text{edit}}(x_I, x_{I'}) \leq 2\hat{\epsilon}$ for $I' \in \mathcal{T}$. Each test costs $O(t(\hat{w}, \hat{\epsilon}))$ (by our assumption on $t(\cdot, \cdot)$), and since $|\mathcal{T}| \leq |\mathcal{B}| = \frac{\mu(I_{\hat{G}})}{\hat{w}\hat{\delta}}$, I is processed in time $O(\frac{\mu(I_{\hat{G}})}{\hat{w}\hat{\delta}} t(\hat{w}, \hat{\epsilon}))$. This is multiplied by the number of intervals declared dense, which we now upper bound. If I is declared dense then at the end of processing I, $\mathcal{X}$ is removed from $\mathcal{T}$. This ensures $\Delta_{\text{edit}}(I, I') > 2\epsilon$ for any two intervals I, I' declared dense. By the triangle inequality the sets $\mathcal{B}(I) = \{J \in \mathcal{B};\ \Delta_{\text{edit}}(x_I, y_J) \leq \epsilon\}$ are disjoint for different pivots. By successful sampling, for each pivot I, $|\mathcal{B}(I)| \geq \frac{\hat{d}}{4}$, and thus at most $|\mathcal{B}|/(\hat{d}/4) = \frac{4\mu(I_{\hat{G}})}{\hat{d}\hat{\delta}\hat{w}}$ intervals are declared dense, so all intervals declared dense are processed in time $O(\frac{\mu(I_{\hat{G}})^2}{\hat{w}^2 \hat{d}\hat{\delta}^2}) t(\hat{w}, \hat{\epsilon})$.

The time for dense/sparse classification of intervals and for processing intervals declared dense is at most $O(\frac{\mu(I_{\hat{G}})^2 \log \hat{n}}{\hat{w}^2 \hat{d}\hat{\delta}^2})t(\hat{w}, \hat{\epsilon})$. During iteration i of the inner loop of CA, the local variables of DSR are set as $\hat{n} = n$, $\mu(I_{\hat{G}}) \leq 4\theta n$, $\hat{w} = w_1$, $\hat{d} = d/\epsilon_i$, $\hat{\delta} = \epsilon_i/8$. Substituting these parameters yields time $O(\frac{\theta^2 n^2 \log n}{(w_1)^2 d\epsilon_i})t(w_1, \epsilon_i)$. Multiplying by the $O(1/\theta)$ iterations on k gives the first summand of the theorem.

Next we turn to SSES. The local input variables $n, w_1, w_2, \mathcal{S}, \theta$ are set to their global values so we denote them without $\hat{}$. The other local input variables are denoted as $\hat{G}, \hat{d}, \hat{\delta}, \hat{\epsilon}$. The local variable $\mathcal{B}$ has size $\frac{\mu(I_{\hat{G}})}{\hat{\delta} w_1}$. By successful sampling, we assume that on every call, every interval in $\mathcal{S}$ is $(\hat{d}, \hat{\delta}, \hat{\epsilon})$- sparse. The outer loop enumerates the $\mu(I_{\hat{G}})/w_2$ intervals I' of $\mathcal{I}_{w_2}(I_{\hat{G}})$. We select $\mathcal{H}$ to be $c_1 \log^2 n$ random subsets from subsets of I' belonging to $\mathcal{S}$. For each $I \in \mathcal{H}$ and $J \in \mathcal{B}$, we call **SMALL-ED**$(x_I, y_J, \hat{\epsilon})$, taking time $t(w_1, \hat{\epsilon})$. The total time of all tests is $O(\frac{\mu(I_{\hat{G}})^2 \log^2 n}{\hat{\delta} w_1 w_2})t(w_1, \hat{\epsilon})$. Using $\hat{d} = d/\epsilon_i$, $\hat{\delta} = \epsilon_i/8$ and $\hat{\epsilon} = \epsilon_i$ from the ith call to SSES gives $O(\frac{\theta^2 n^2 \log^2 n}{\epsilon_i w_1 w_2})t(w_1, \epsilon_i)$. Multiplying by the $O(1/\theta)$ iterations on k gives the second summand in the theorem.

Assuming successful sampling, all intervals in the set $\mathcal{S}$ passed from DSR to SSES are $(\hat{d}, \hat{\delta}, \hat{\epsilon})$-sparse. Therefore, for each sampled I, at most $\hat{d}$ intervals J are within $\hat{\epsilon}$ of I. For each of these we do a diagonal extension of $I \times J$ to a w_2-box $I' \times J'$, and call **SMALL-ED**$(x_{I'}, y_{J'}, 3\hat{\epsilon})$ at cost $O(t(w_2, \hat{\epsilon}))$ for each call. The number of such calls is $O(\frac{\mu(I_{\hat{G}})\hat{d} \log^2 n}{w_2})$. Using the parameter $\hat{d} = d/\epsilon_i$ in the ith call of the inner iteration of CA, we get a cost of $O(\frac{\theta n d \log^2 n}{\epsilon_i w_2})t(w_2, \epsilon_i)$ and multiplying by the $O(1/\theta)$ gives the third summand in the theorem. □

Choosing the parameters to minimize the maximum term in the time bound, subject to the restrictions of the theorem and using $t(w, \epsilon) = O(\epsilon w^2)$ we have:

Corollary 26. *For all sufficient large n, and for $\theta \geq n^{-1/5}$ (both powers of 2) choosing w_1, w_2, and d to be the largest powers of two satisfying: $w_1 \leq \theta^{-2/7} n^{1/7}$, $w_2 \leq \theta^{1/7} n^{3/7}$, and $d \leq \theta^{3/7} n^{2/7}$, with probability at least $1 - n^{-1/7}$, CA runs in time $\tilde{O}(n^{12/7}\theta^{4/7})$, and outputs the set $\mathcal{R}$ of size at most $\tilde{O}(n^{12/7}\theta^{4/7})$.*

Proof. Set w_1, w_2, and d to be the largest powers of two satisfying: $w_1 \leq \theta^{-2/7} n^{1/7}$, $w_2 \leq \theta^{1/7} n^{3/7}$, and $d \leq \theta^{3/7} n^{2/7}$.

Use the algorithm of [Ukk85] that gives $t(w, \epsilon) = O(\epsilon w^2)$. It can be easily verified that these choices satisfy the requirements of Theorem 25, and also that all three terms in the time analysis, and the number of boxes are all bounded by the claimed bound. □

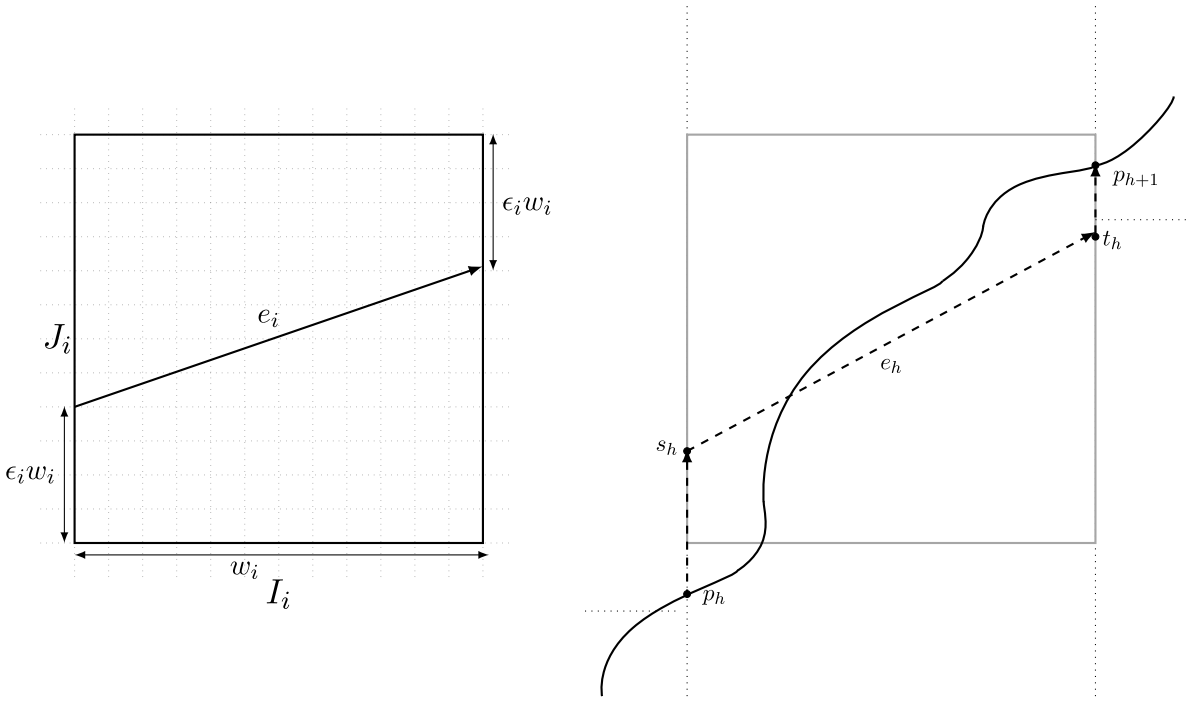
2.4 Min-cost Paths in Shortcut Graphs

We now describe the second phase of our algorithm, which uses the set $\mathcal{R}$ output by CA to upper bound $d_{\text{edit}}(x, y)$. A *shortcut graph* on vertex set $\{0, \ldots, n\} \times \{0, \ldots, n\}$ consists of the H and V edges of cost 1, together with an arbitrary collection of *shortcut* edges $(i, j) \to (i', j')$ where $i < i'$ and $j < j'$, also denoted by $e_{I,J}$ where $I = \{i, \ldots, i'\}$ and $J = \{j, \ldots, j'\}$, along with their costs. A *certified graph* (for x, y) is a shortcut graph where every shortcut edge $e_{I,J}$ has cost at least $d_{\text{edit}}(x_I, y_J)$. The min-cost path from $(0, 0)$ to (n, n) in a certified

graph upper bounds $d_{\text{edit}}(x, y)$. The second phase algorithm uses $\mathcal{R}$ to construct a certified graph, and computes the min cost path to upper bound on $d_{\text{edit}}(x, y)$.

A certified box $(I \times J, \kappa)$ corresponds to the $e_{I,J}$ with cost $\kappa\mu(I)$. (In the certified graph we use non-normalized costs.) However, the certified graph built from $\mathcal{R}$ in this way may not have a path of cost $O(d_{\text{edit}}(x, y) + \theta n)$. We need a modified conversion of $(I \times J, \kappa)$. If $\kappa \geq 1/2$ we add no shortcut. Otherwise $(I \times J, \kappa)$ converts to the edge $e_{I,J'}$ with cost $3\kappa\mu(I)$ where J' is obtained by shrinking J: $\min(J') = \min(J)+\ell$ and $\max(J') = \max(J')-\ell$ where $\ell = \lfloor \kappa\mu(I) \rfloor$. Call the resulting graph $\widetilde{G}$. We claim:

Lemma 5. *Let τ be a path from source to sink in $G_{x,y}$. If $\mathcal{R}$ contains a sequence σ that (k, θ)-approximates τ then there is a source-sink path τ' in $\widetilde{G}$ that consists of the shortcuts corresponding to σ together with some H and V edges with $cost_{\widetilde{G}}(\tau') \leq 5(k \cdot cost_{G_{x,y}}(\tau) + \theta n)$.*



(a) (b)

Figure 2.6: (a) The shortcut edge e_i is added for box $I_i \times J_i$. (b) An example of a path τ (in solid) passing through a box $I_h \times J_h$. The dashed path ${\tau'}_h$ is an approximation of τ between p_h and p_{h+1}. Here $s_h = (\min(I_h), \min(J'_h))$ and $t_h = (\max(I_h), \max(J'_h))$.

Proof. We will modify the path τ in $G_{x,y}$ to a path τ' in $\widetilde{G}$ of comparable cost. Let $\{(I_1 \times J_1, \epsilon_1), (I_2 \times J_2, \epsilon_2), \ldots, (I_m \times J_m, \epsilon_m)\}$ be the set of certified boxes that (k, θ)-approximates τ. Let $\ell_i = \mu(I_i) \cdot \epsilon_i$. Let L be the subset $[m]$ for which $\epsilon_i \leq 1/2$. For $i \in L$, let $e_i = e_{I_i, J_i'}$ be the shortcut edge with weight $3\epsilon_i$. We claim (1) there is a source-sink path in $\widetilde{G}$ that consists of $\{e_i : i \in L\}$ together with a horizontal path H_i whose projection to the x-axis is I_i for each $i \in [m] - L$, and a collection of (possibly empty) vertical paths $V_0, V_1, \ldots, V_m$ where the x-coordinate of V_i for $i > 0$ is $\max(I_i)$ and 0 for V_0, and (2) its cost satisfies the bound of the Lemma.

For the first claim, define for $h \in [m]$ $p_h = (i_h, j_h)$ to be the first point in τ_{I_h} and define $p_{m+1} = (n, n)$. We will define τ' to pass through all of the p_h. In preparation, observe that for $h \in L$, since $I_h \times J_h$ $(1 - \epsilon_h)$ covers τ, we have $\min(J_h') = \min(J_h) + \ell_h \geq j_h$ and $\max(J_h') = \max(J) - \ell_h \leq j_{h+1}$. Define the portion τ_h' between p_h and p_{h+1} by climbing vertically from p_h to $(i_h, \min(J_h'))$ and if $h \in L$ traversing $e_h = e_{I_h, J_h'}$ and climbing to p_{h+1} and if $h \notin L$ then move horizontally from $(i_h, \min(J_h'))$ to $(i_{h+1}, \min(J_h'))$ and then climb to p_{h+1}.

For the second claim, we upper bound $\text{cost}(\tau')$. For $h \in L$, e_{I_h, J_h} costs $3\ell_h$, and for $h \notin L$, the horizontal path that projects to I_h costs $\mu(I_h) \leq 2\ell_h$; the total is at most $\sum_h 3\ell_h$. The cost of vertical edges is $n - \sum_{h \in L} \mu(J_h') = \sum_{h \in L} (\mu(J_h) - \mu(J_h')) + \sum_{h \notin L} \mu(J_h) = \sum_{h \in L} 2\ell_h + \sum_{h \notin L} \mu(J_h) \leq \sum_h 2\ell_h$, since $\sum_h \mu(J_h) = \sum_h \mu(I_h) = n$. So $\text{cost}(\tau') \leq \sum_h 5\ell_h$. Since $\sum_{i=1}^m \ell_i \leq k \cdot \text{cost}_{G_{x,y}}(\tau) + \theta \cdot n$ by definition of (k, θ)-approximation, the lemma follows. □

The min-cost path algorithm.

We present an $O(n + m \log(mn))$ algorithm to find a min cost source-sink path in a shortcut graph $\widetilde{G}$ with m shortcuts. It's easier to switch to the max-benefit problem: Let $\widetilde{H}$ be the same graph with cost c_e of $e = (i, j) \to (i', j')$ replaced by *benefit* $b_e = (i' - i) + (j' - j) - c_e$, (so H and V edges have benefit 0). The min-cost path of $\widetilde{G}$ is $2n$ minus the max-benefit path of $\widetilde{H}$. To compute the max-benefit path of $\widetilde{H}$, we use a binary tree data structure with leaves $\{1, \ldots, n\}$, where each node v stores a number b_v, and a collection of lists $L_1, \ldots, L_n$, where L_i stores pairs $(e, q(e))$ where the head of e has x-coordinate i and $q(e)$ is the max benefit of a path that ends with e.

We proceed in $n - 1$ rounds. Let the set A_i consist of all the shortcuts whose tail has x-coordinate i. The preconditions for round i are: (1) for each leaf j, the stored value b_j is the max benefit path to (i, j) that includes a shortcut whose head has y-coordinate j (or 0 if there is no such path), (2) for each internal node v, $b_v = \max\{b_j : j \text{ is a leaf in the subtree of } v\}$. and (3) for every edge $e = (i', j') \to (i'', j'')$ with $i' < i$, the value $q(e)$ has been computed and $(e, q(e))$ is in list $L_{i''}$. During round i, for each shortcut $e = (i, j) \to (i', j')$ in A_i, $q(e)$ equals the max of $b_v + b_e$ over tree leaves v with $v \leq j$. This can be computed in $O(\log n)$ time as $\max b_v + b_e$, over $\{j\}$ union the set of left children of vertices on the root-to-j path that are not themselves on the path. Add $(e, q(e))$ to list $L_{i'}$. After processing A_i, update the binary tree: for each $(e, q(e)) \in L_{i+1}$, let j be the y-coordinate of the head of e and for all vertices v on the root-to-j path, replace b_v by $\max(b_v, q(e))$. The tree then satisfies the precondition for round $i + 1$. The output of the algorithm is b_n at the end of round $n - 1$. It takes $O(n)$ time to set up the data structure, $O(m \log m)$ time to sort the shortcuts,

and $O(\log n)$ processing time per shortcut (computing $q(e)$ and later updating the data structure).

2.5 Conclusion and Bibliographical Notes

To summarize, the algorithm **GAP-UB**$_\theta$ first runs the CoveringAlgorithm of Section 2.3 that outputs a set of *certifies boxes*. The algorithm then converts the output into a shortcut graph, and runs the min-cost path algorithm of Section 3.3 to find a min-cost source-sink path in the shortcut graph. By Corollary 26, and the quasilinear runtime (in the number of shortcuts) of the min-cost path algorithm, the algorithm **GAP-UB**$_\theta$ runs in time $\tilde{O}(n^{12/7}\theta^{4/7})$. Moreover, by Lemma 24 and Lemma 5, combined with the padding argument from Section 2.1, the final approximation factor is 610.

Further improvements:

Optimizing the approximation factor. In order to preserve the simplicity, we do not optimize the approximation factor in our algorithm. Our algorithms can be easily adjusted to guarantee an approximation factor $5+\epsilon$, for any fixed $\epsilon > 0$, while increasing the running time only by a constant factor. To do that one needs to chose various estimates (such as $\theta, \epsilon_i, \dots$) in smaller increments (instead of directly doubling) and align boxes with finer granularity. This yields a finer approximation to the shortest path in our covering algorithm. The factor comes from the triangle inequality used in the dense case. We omit the proof details from the current thesis.

Speeding up the algorithm. The runtime of **ED-UB** is dominated by the cost of **SMALL-ED**(z_1, z_2, ϵ) on pairs of strings of length $w \in \{w_1, w_2\}$. We use Ukkonen's algorithm [Ukk85] with $t(w, \epsilon) = O(\epsilon w^2)$. Replacing the Ukkonen's algorithm with **ED-UB**, we get a revised algorithm **ED-UB**$_1$. This worsens the approximation factor (roughly multiplying it by the approximation factor of **ED-UB**) but improves runtime. The internal parameters w_1, w_2, d are adjusted to maximize savings. One can iterate this process any constant number of times to get faster algorithms with worse (but still constant) approximation factors. Because of the dependence of the analysis on θ, we do not get a faster edit distance algorithm for all $\theta \in [0, 1]$ but only for θ close to 1. (This may be an artifact of our analysis rather than an inherent limitation.)

In the special case of one level of the recursion, using $t(w, \epsilon) = \tilde{O}(w^{12/7}\epsilon^{4/7})$ and choosing w_1, w_2, and d to be the largest powers of two satisfying: $w_1 \leq \theta^{-27/277} n^{49/277}$, $w_2 \leq \theta^{85/277} n^{399/831}$, and $d \leq \theta^{112/277} n^{84/277}$, we obtain an algorithm for the full range of $\theta \in [0, 1]$ that runs in time $\widetilde{O}(n^{2-98/277}\theta^{54/277}) = \widetilde{O}(n^{1.647}\theta^{0.195})$ if $d_{\text{edit}}(x, y) \leq \theta n$.

Open Problems. The most promising open problem after **[CDG+18]** is to get a constant approximation of edit distance in near linear time. Another question that arises is, to improve the approximation factor. In our algorithm, we can bring down the approximation factor to $5+\epsilon$. But due to triangular inequality 2-approximation seems to be a real bottleneck. Therefore it will be quite interesting

to investigate whether the approximation factor can be brought down below 2 or not. Apart from this, one can also try other variants of the standard edit distance problem, where the input strings follow some nice property using which one can hope for some better and efficient algorithm.

3. Approximate Pattern Matching

In this chapter we consider the approximate pattern matching problem. We study this problem in both offline and online setting. Both of our algorithms are randomized and produce correct answer with high probability. To the best of our knowledge, this is the first worst case sub-linear time (in the length of the pattern) and sub-linear succinct space algorithm for online pattern matching. In this chapter, the term "succinct space" has somewhat different interpretation than it's usual notion. By this we mean that, other than storing the pattern, the online algorithm requires space which is sub-linear in the length of the pattern. Our results are largely build over the techniques developed in Chapter 2, for computing constant approximation of edit distance in sub-quadratic time.

3.1 Preliminaries

The basic definitions and preliminary observations and claims used in this chapter are mostly taken or derived from Chapter 2. We recall and restate a few of them. Consider the text T of length n to be aligned along the horizontal axis and the pattern P of length w to be aligned along the vertical axis. For $i \in \{1, \ldots, n\}$, T_i denotes the i-th symbol of T and for $j \in \{1, \ldots, w\}$, P_j denotes the j-th symbol of P. $T_{s,t}$ is the substring of T starting by the s-th symbol and ending by the t-th symbol of T. For any interval $I \subseteq \{0, \ldots, n\}$, T_I denotes the substring of T indexed by $I \setminus \{\min(I)\}$ and for $J \subseteq \{0, \ldots, w\}$, P_J denotes the substring of P indexed by $J \setminus \{\min(J)\}$.

Edit distance and pattern matching graphs. For a text T of length n and a pattern P of length w, the *edit distance graph* $G_{T,P}$ is a directed weighted graph with vertex set $\{0, \cdots, n\} \times \{0, \cdots, w\}$ (see Definition 2.1.2). The *pattern matching graph* $\tilde{G}_{T,P}$ is the same as the edit distance graph $G_{T,P}$ except for the cost of horizontal edges $(i, 0) \to (i + 1, 0)$ which is zero.

For $I \subseteq \{0, \ldots, n\}$ and $J \subseteq \{0, \ldots, w\}$, $G_{T,P}(I \times J)$ is the subgraph of $G_{T,P}$ induced on $I \times J$. Clearly, $G_{T,P}(I \times J) \cong G_{T_I,P_J}$. We define the cost of a path τ in G_{T_I,P_J}, denoted by $\text{cost}_{G_{T_I,P_J}}(\tau)$, as the sum of the costs of its edges. We also define the cost of a graph G_{T_I,P_J}, denoted by $\text{cost}(G_{T_I,P_J})$, as the cost of the cheapest path from $(\min I, \min J)$ to $(\max I, \max J)$.

The following is well known in the literature (e.g. see [Sel80]).

Proposition 27. *Consider a pattern P of length w and a text T of length n, and let $G = \tilde{G}_{T,P}$. For any $t \in \{1, \ldots, n\}$, let $I = \{0, \cdots, t\}$, and $J = \{0, \cdots, w\}$. Then $k_t = cost(G(I \times J)) = \min_{i \le t} d_{edit}(T_{i,t}, P)$.*

A similar proposition is also true for the edit distance graph.

Proposition 28. *Consider a pattern P of length w and a text T of length n, and let $G = G_{T,P}$. For any $i_1 \le i_2 \in \{1, \cdots, n\}$, $j_1 \le j_2 \in \{1, \cdots, w\}$ let $I = \{i_1 - 1, \cdots, i_2\}$ and $J = \{j_1 - 1, \cdots, j_2\}$. Then $cost(G(I \times J)) = d_{edit}(T_{i_1,i_2}, P_{j_1,j_2})$.*

Let G be a grid graph on $I \times J$ and $\tau = (i_1, j_1), \ldots, (i_l, j_l)$ be a path in G. *Horizontal projection* of a path τ is the set $\{i_1, \ldots, i_l\}$. Let I' be a set contained in the horizontal projection of τ, then $\tau_{I'}$ denotes the (unique) minimal subpath of τ with horizontal projection I'. Let $G' = G(I' \times J')$ be a subgraph of G. For $\delta \in [0, 1]$ we say that $I' \times J'$ $(1 - \delta)$-*covers* the path τ if the initial and the final vertex of $\tau_{I'}$ are at a vertical distance of at most $\delta(|I'|-1)$ from $(\min(I'), \min(J'))$ and $(\max(I'), \max(J'))$, resp..

A *certified box* of G is a pair $(I' \times J', \ell)$ where $I' \subseteq I$, $J' \subseteq J$ are intervals, and $\ell \in \mathbb{N}$ such that $\text{cost}(G(I' \times J')) \leq \ell$. At high level, our goal is to approximate each path τ in G by a path via the corner vertices of certified boxes. For that we want that a substantial portion of the path τ goes via those boxes and that the sum of the costs of the certified boxes is not much larger than the actual cost of the path. The next definition makes our requirements precise. Let $\sigma = \{(I_1 \times J_1, \ell_1), (I_2 \times J_2, \ell_2), \ldots, (I_m \times J_m, \ell_m)\}$ be a sequence of certified boxes in G. Let τ be a path in $G(I \times J)$ with horizontal projection I. For any $k, \zeta \geq 0$, we say that σ (k, ζ)-*approximates* τ if the following three conditions hold:

1. $I_1, \ldots, I_m$ is a decomposition of I, i.e., $I = \bigcup_{i \in [m]} I_i$, and for all $i \in [m-1]$, $\min(I_{i+1}) = \max(I_i)$.
2. For each $i \in [m]$, $I_i \times J_i$ $(1 - \ell_i/(|I_i| - 1))$-covers τ.
3. $\sum_{i \in [m]} \ell_i \leq k \cdot \text{cost}(\tau) + \zeta$.

3.2 Offline Approximate Pattern Matching

Theorem 29. *There is a constant $c \geq 1$ and there is a randomized algorithm that computes c-approximation to approximate pattern matching in time $O(n \cdot w^{3/4})$ with probability at least $(1 - 1/n^3)$.*

3.2.1 Technique Overview

To prove the above theorem we design our algorithm that works as follows: For $k = 2^j$, $j = 0, \ldots, \log w^{3/4}$, we run the standard $O(kn)$ algorithm [GG88] to identify all t such that $k_t \leq k$. To identify positions with $k_t \leq k$ for $k > w^{3/4}$ where k is a power of two we will use the technique of Chapter 2 to compute $(O(1), O(w^{3/4}))$-approximation of $k_1, \ldots, k_n$. The obtained information can be combined in a straightforward manner to get a single $O(1)$-approximation to $k_1, \ldots, k_n$: For each t, if for some $2^j \leq w^{3/4}$, k_t is at most 2^j (as determined by the former algorithm) then output the smallest such 2^j as the approximation of k_t, otherwise output the approximation of k_t found by the latter algorithm. This way, for $k_t \leq w^{3/4}$ we will get 2-approximation, and for $k > w^{3/4}$ we will get a $O(1)$-approximation. We will now elaborate on the latter algorithm. The technique is primarily based on the edit distance algorithm of Chapter 2, which works in two phases and this is similar for our pattern matching algorithm as well. The first phase (*covering phase*) identifies a set of *certified boxes*, that is subgraphs of the pattern matching graph with good upper bounds on their cost. These certified boxes should adequately cover the min-cost paths of interest. Then the next phase runs a min-cost path algorithm on these boxes to obtain

the output sequence. We will show that both of these phases will take $\widetilde{O}(nw^{3/4})$ time so the overall running time of our algorithm will be $\widetilde{O}(nw^{3/4})$.

We next describe the two phases of the algorithm. The algorithm will use the following parameters: $w_1 = w^{1/4}$, $w_2 = w^{1/2}$, $d = w^{1/4}$, $\theta = w^{-1/4}$, which essentially mean the same as in Chapter 2 though their setting is different. Moreover similar to Chapter 2, we use the following simplifying assumptions: without loss of generality we assume that w_1 and w_2 are powers of two (by rounding them down to the nearest powers of two), $1/\theta$ is a reciproval of a power of two (by decreasing θ by at most a factor of two), $w_2|w$ (by chopping off a small suffix from P which will affect the approximation by a negligible additive error as $w^{3/4} \gg w_2$), and $w|n$ (if not we can run the algorithm twice: on the largest prefix of T of length divisible by w and then on the largest suffix of T of length divisible by w). Let $c_0, c_1 \geq 0$ be the large enough constants. The algorithm will not explicitly compute k_t for all t but only for t where t is a multiple of w_2, and then it will use the same value for each block of w_2 consecutive k_t's. Again, this will affect the approximation by a negligible additive error.

3.2.2 Covering phase

We describe the first phase (*covering phase*) of the algorithm now. First, we partition the text T into substrings $T_1^0, \ldots, T_{n_0}^0$ of length w, where $n_0 = n/w$. Then we process each of the parts independently. Let T' be one of the parts. We partition T' into substrings $T_1^1, T_2^1, \ldots, T_{n_1}^1$ of length w_1, and we also partition T' into substrings $T_1^2, T_2^2, \ldots, T_{n_2}^2$ of length w_2, where $n_1 = w/w_1$ and $n_2 = w/w_2$. For a substring u of v starting by i-th symbol of v and ending by j-th symbol of v, we let $\{i-1, i+1, \ldots, j, j\}$ be its *span*. Moreover for $\delta \in (0,1)$ we call u to be (δ)-aligned if both $i-1$ and $j-1$ are divisible by $\delta(j-i)$. The covering algorithm proceeds in phases $j = 0, \ldots, \lceil \log 1/\theta \rceil$ associated with $\epsilon_j = 2^{-i}$. Similar to the edit distance algorithm, here also each phase has two parts, namely the *dense case* and the *extension sampling*.

Dense case. In this part the algorithm aims to identify for each ϵ_j, a set of substrings T_i^1 which are similar to more than d $(\epsilon_j/8)$-aligned, w_1 length substrings of P . We identify each T_i^1 by testing a random sample of relevant substrings of P. If we determine with high confidence that there are at least $\Omega(d)$ substrings of P similar to T_i^1, we add T_i^1 into a set D_j of such strings, and we also identify all $T_{i'}^1$ that are similar to T_i^1. By triangle inequality we would also expect them to be similar to many relevant substrings of P. So we add these $T_{i'}^1$ to D_j as well as we will not need to process them anymore. We output the set of certified boxes of edit distance $O(\epsilon_j w_1)$ found this way. More formally:

For $j = \lceil \log 1/\theta \rceil, \ldots, 0$, the algorithm maintains sets D_j of substrings T_i^1. These sets are initially empty.

Step 1. For each $i = 1, \ldots, n_1$ and $j = \lceil \log 1/\theta \rceil, \ldots, 0$, if T_i^1 is in D_j then we continue with the next i and j. Otherwise we process it as follows.

Step 2. Set $\epsilon_j = 2^{-j}$. Independently at random, sample $8c_0 \cdot w \cdot (\epsilon_j w_1 d)^{-1} \cdot \log n$ many $(\epsilon_j/8)$-aligned substrings of P of length w_1. For each sampled substring u check if its edit distance from T_i^1 is at most $\epsilon_j w_1$. If less than $\frac{1}{2} \cdot c_0 \cdot \log n$ of

the samples have their edit distance from T_i^1 below $\epsilon_j w_1$ then we are done with processing this i and j and we continue with the next pair.

Step 3. Otherwise we identify all substrings $T_{i'}^1$ that are not in D_j and are at edit distance at most $2\epsilon_j w_1$ from T_i^1, and we let X to be the set of their spans relative to the whole T.

Step 4. Then we identify all $(\epsilon_j/8)$-aligned substrings of P of length w_1 that are are at edit distance at most $3\epsilon_j w_1$ from T_i^1, and we let Y to be the set of their spans. We might allow also some $(\epsilon_j w_1/8)$-aligned substrings of P of edit distance at most $6\epsilon_j w_1$ to be included in the set Y (as some might be misidentified to have the smaller edit distance from T_i^1 by our procedure that searches for them).

Step 5. For each pair of spans (I, J) from $X \times Y$ we output corresponding certified box $(I \times J, 8\epsilon_j w_1)$. We add substrings corresponding to X into D_j and continue with the next pair i and j.

Once we process all pairs of i and j, we proceed to the next phase: *extension sampling.*

Extension sampling. In this part for every $\epsilon_j = 2^{-j}$ and every substring T_i^2, which does not have all its substrings T_ℓ^1 contained in D_j we randomly sample a set of such T_ℓ^1's. For each sampled T_ℓ^1 we determine all $(\epsilon_j/8)$-aligned, w_1 length substrings of P at edit distance at most $\epsilon_j w_1$ from T_ℓ^1. There should be $O(d)$-many such substrings of P. We extend each such substring into a substring of size $|T_i^2|$ within P and we check the edit distance of the extended string from T_i^2. For each extended substring of edit distance at most $3\epsilon_j w_2$ we output a set of certified boxes.

Here we define the appropriate extension of substrings. Let u be a substring of T of length less than $|P|$, and let v be a substring of u starting by the i-th symbol of u. Let v' be a substring of P of the same length as v starting by the j-th symbol of P. The *diagonal extension u' of v' in P with respect to u and v*, is the substring of P of length $|u|$ starting at position $j - i$. If $(j - i) \leq 0$ then the extension u' is the prefix of P of length $|u|$, and if $j - i + |P| > |P|$ then the extension u' is the suffix of P of length $|u|$.

Step 6. Process all pairs $i = 1, \ldots, n_2$ and $j = \lceil \log 1/\theta \rceil, \ldots, 0$.

Step 7. Independently at random, sample $c_1 \cdot \log^2 n \cdot \log w$ substrings T_ℓ^1 that are part of T_i^2 and that are not in D_j. (If there is no such substring continue for the next pair of i and j.)

Step 8. For each T_ℓ^1, find all $(\epsilon_j/8)$-aligned substrings v' of P of length w_1 that are at edit distance at most $\epsilon_j w_1$ from T_ℓ^1.

Step 9. For each v' determine its diagonal extension u' with respect to T_i^2 and T_ℓ^1. Check if the edit distance of u' and T_i^2 is less than $3\epsilon_j w_2$. If so, compute it and denote the distance by c. Let I' be the span of T_i^2 relative to T, and J' be the span of u' in P. For all powers a and b of two, $w^{3/4} \leq a \leq b \leq w$, output the certified box $(I' \times J', c + a + b)$. Proceed for the next i and j.

This ends the covering algorithm which outputs various certified boxes.

To implement the above algorithm we will use Ukkonen's [Ukk85] $O(nk)$-time algorithm to check whether the edit distance of two strings of length w_1 is at most $\epsilon_j w_1$ in time $O(w_1^2 \epsilon_j)$. Given the edit distance is within this threshold the algorithm can also output its precise value. To identify all substrings of length w_1

at edit distance at most $\epsilon_j w_1$ of S from a given string R (where S is the pattern P of length w and R is one of the T_i^1 of length w_1) we use the $O(nk)$-time pattern matching algorithm of Galil and Giancarlo [GG88]. For a given threshold k, this algorithm determines for each position t in S, whether there is a substring of edit distance at most k from R ending at that position in S. If the algorithm reports such a position t then we know by the following proposition that the substring $S_{t-|R|+1,t}$ is at edit distance at most $2k$. At the same time we are guaranteed to identify all the substrings of S of length w_1 at edit distance at most k from R. Hence in Step 4, finding all the substrings at distance $3\epsilon_j w_1$ with perhaps some extra substrings of edit distance at most $6\epsilon_j w_1$ can be done in time $O(ww_1\epsilon_j)$.

Proposition 30. *For strings S and R and integers $t \in \{1, \ldots, |S|\}$, $k \geq 0$, if $\min_{i \leq t} d_{edit}(S_{i,t}, R) \leq k$ then $d_{edit}(S_{t-|R|+1,t}, R) \leq 2k$.*

Proof. Let $S_{i,t}$ be the best match for R ending by the t-th symbol of S. Hence, $k = d_{\text{edit}}(S_{i,t}, R)$. If $S_{i,t}$ is by ℓ symbols longer that R then $k \geq \ell$ and $d_{\text{edit}}(S_{t-|R|+1,t}, R) \leq k + \ell \leq 2k$ by the triangle inequality. Similarly, if $S_{i,t}$ is shorter by ℓ symbols. □

3.2.3 Correctness of the covering algorithm

Lemma 6. *Let $t \geq 1$ be such that t is a multiple of w_2. Let τ_t be the min-cost path between vertex $(t - w, 0)$ and (t, w) in the edit distance graph $G = G_{T,P}$ of T and P of cost at least $w^{3/4} \geq \theta w$. The covering algorithm outputs a set of weighted boxes $\mathcal{R}$ such that every $(I \times J, \ell) \in \mathcal{R}$ is correctly certified i.e., cost$(G(I \times J)) \leq \ell$ and there is a subset of $\mathcal{R}$ that $(O(1), O(k_t))$-approximates τ_t with probability at least $1 - 1/n^7$.*

It is clear from the description of the covering algorithm that it outputs only correct certified boxes from the edit distance graph of T and P, that is for each box $(I \times J, \ell)$, $\text{cost}(G(I \times J)) \leq \ell$.

The cost of τ_t corresponds to the edit distance between P and $T_{t-w+1,t}$ and it is bounded by $2k_t$ by Proposition 30. Let k'_t be the smallest power of two $\geq k_t$. We claim that by essentially the similar argument as in Proposition 23 and Theorem 24 of Chapter 2 the algorithm outputs with high probability a set of certified boxes that $(O(1), O(k'_t))$-approximates τ_t. Therefore instead of repeating the whole proof, here we sketch the differences between the current covering algorithm with that of Chapter 2 and argue about how to handle them.

The main substantial difference is that the edit distance algorithm in 2 searches for certified boxes located only within $O(k_t)$ diagonals along the main diagonal of the edit distance graph. (This rests on the observation of Ukkonen [Ukk85] that a path of cost $\leq k_t$ must pass only through vertices on those diagonals.) Here we process certified boxes in the whole matrix as each t requires a different "main" diagonal. Except for this difference and the order of processing various pieces the algorithms are the same.

The discovery of certified boxes depends on the number (*density*) of relevant substrings of P similar to a given T_i^1. In the edit distance algorithm in 2 this density is measured only in the $O(k_t)$-width strip along the main diagonal of the edit distance graphs whereas here it is measured within the whole P. (So the actual classification of substrings T_i^1 on *dense* (in D_j) and *sparse* (not in D_j) might

differ between the two algorithms.) Hence, One could think (though technically not quite correct) that the certified boxes output by the current algorithm form a superset of boxes output by the edit distance algorithm of Chapter 2. However, this difference is immaterial for the correctness argument in Theorem 24 of Chapter 2 .

Another difference is that in Steps 4 we use $O(ww_1\epsilon_j)$-time algorithm to search for all the similar substrings. This algorithm will report all the substrings we were looking for and additionally it might report some substrings of up to twice the required edit distance. This necessitates the upper bound $8\epsilon_j w_1$ in certified boxes in Step 5. It also means a loss of factor of at most two in the approximation guarantee as the boxes of interest are reported with the cost $8\epsilon_j w_1$ instead of the more accurate $5\epsilon_j w_1$ of the original algorithm in Chapter 2 which would give a $(45, 15\text{cost}(\tau_t))$-approximation. (In that theorem θw represents an (arbitrary) upper bound on the cost of τ_t provided it satisfies certain technical conditions requiring that θ is large enough relative to w. This is satisfied by requiring that $\text{cost}(\tau_t) \geq w^{3/4} \geq \theta w$.)

Another technical difference is that the path τ_t might pass through two edit distance graphs $G_{T^0_{\ell-1},P}$ and $G_{T^0_\ell,P}$, where $t \in [(\ell-1)w+1, \ell w]$. This means that one needs to argue separately about restriction of τ_t to $G_{T^0_{\ell-1},P}$ and $G_{T^0_\ell,P}$. However, the proof of Theorem 24 in Chapter 2 analyses approximation of the path in separate parts restricted to substrings of T of size w_2. As both t and w are multiples of w_2, the argument for each piece applies in our setting as well.

3.2.4 Time complexity of the covering algorithm

Claim 7. *The covering algorithm runs in time $\widetilde{O}(nw^{3/4})$ with probability at least $1 - 1/n^8$.*

We analyse the running time of the covering algorithm for each $T' = T^0_i$ separately. We claim that the running time on T' is $\widetilde{O}(w^{7/4})$ so the total running time is $\widetilde{O}((n/w)w^{7/4}) = \widetilde{O}(nw^{3/4})$.

In Step 1, for every $i = 1, \ldots, n_1$ and $j = 0, \ldots, \log w^{1/4}$, we might sample $O(\frac{w}{\epsilon_j w_1 d} \cdot \log n)$ substrings of P of length w_1 and check whether their edit distance from T^1_i is at most $\epsilon_j w_1$. This takes time at most $\widetilde{O}(\frac{w}{\epsilon_j w_1 d} \cdot \frac{w}{w_1} \cdot w_1^2 \epsilon_j) = \widetilde{O}(w^2/d) = \widetilde{O}(w^{7/4})$ in total.

We say that a bad event happens either if some substring T^1_i has more than d relevant substrings of P having distance at most $\epsilon_j w_1$ but we sample less than $\frac{1}{2} \cdot c_0 \log n$ of them, or if some substring T^1_i has less than $d/4$ relevant substrings of P having distance at most $\epsilon_j w_1$ but we sample more than $\frac{1}{2} \cdot c_0 \log n$ of them. By Chernoff bound, the probability of a bad event happening during the whole run of the covering algorithm is bounded by $\exp(-O(\log n)) \leq 1/n^8$, for sufficiently large constant c_0. Assuming no bad event happens we analyze the running time of the algorithm further.

Each substring T^1_i that reaches Step 3 can be associated with a set of its relevant substrings in P of edit distance at most $\epsilon_j w_1$ from it. The number of these substrings is at least $d/4$ many. These substrings must be different for different strings T^1_i that reach Step 3 as if they were not distinct then the two substrings T^1_i and $T^1_{i'}$ would be at edit distance at most $2\epsilon_j w_1$ from each other

and one of them would be put into D_j in Step 5 while processing the other one so it could not reach Step 3. Hence, we can reach Steps 3–5 for at most $\frac{8w}{\epsilon_j w_1} \cdot \frac{4}{d}$ strings T_i^1. For a given j and each T_i^1 that reaches Step 3, the execution of Steps 3 and 4 takes $O(ww_1\epsilon_j)$ time, hence we will spend in them $\widetilde{O}(w^2/d) = \widetilde{O}(w^{7/4})$ time in total.

Step 5 can report for each j at most $\frac{8w}{\epsilon_j w_1} \cdot \frac{w}{w_1}$ certified boxes, so the total time spent in this step is $\widetilde{O}(w^2/w_1) = \widetilde{O}(w^{7/4})$ as $\epsilon_j w_1 \geq 1/4$.

Step 7 takes order less time than Step 8. In Step 8 we use Ukkonen's [Ukk85] $O(nk)$-time edit distance algorithm to check the distance of strings of length w_1. We need to check $\widetilde{O}(n_2 \cdot \frac{w}{\epsilon_j w_1})$ pairs for the total cost $\widetilde{O}(\frac{w}{w_2} \cdot \frac{w}{\epsilon_j w_1} \cdot w_1^2 \epsilon_j) = \widetilde{O}(w^{7/4})$ per j.

As no bad event happens, for each T_ℓ^1, there will be at most $d/4$ strings v' processed in Step 9. We will spend $O(w_2^2\epsilon_j)$ time on each of them to check for edit distance and $O(\log^2 n)$ to output the certified boxes. Hence, for each j we will spend here $\widetilde{O}(\frac{w}{w_2} \cdot dw_2^2\epsilon_j)$ time, which is $\widetilde{O}(ww_2d)$ in total.

Thus, the total time spent by the algorithm in each of the steps is $\widetilde{O}(w^{7/4})$ as required.

3.3 Min-cost Path in a Grid Graph with Shortcuts

In this section we explain how we use certified boxes to calculate the approximation of k_t's. Consider any grid graph G. A *shortcut* in G is an additional edge $(i, j) \to (i', j')$ with cost ℓ, where $i < i'$ and $j < j'$.

Let $G_{T,P}$ be the edit distance graph for T and P. Let $(I \times J, \ell)$ be a certified box in $G_{T,P}$ with $|I| = |J|$. If $\ell < 1/2(|I| - 1)$ add a shortcut edge $e_{I,J}$ from vertex $(\min I, \min J + \ell)$ to vertex $(\max I, \max J - \ell)$ with cost 3ℓ. Do this for all certified boxes output by the covering algorithm to obtain a graph$G'_{T,P}$. Next remove all the diagonal edges (D-steps) of cost 0 or 1 from graph $G'_{T,P}$ and obtain graph graph $G''_{T,P}$.

Lemma 8. *If τ is a path from $(t - w, 0)$ to (t, w) in $G_{T,P}$ which is (k, ζ)-approximated by a subset of certified boxes σ by the covering algorithm then there is a path from $(t - w, 0)$ to (t, w) in $G''_{T,P}$ of cost at most $5 \cdot (k \cdot cost_{G_{T,P}}(\tau) + \zeta)$ consisting of shorcut edges corresponding to σ and H and V steps.*

The proof of the above lemma is exactly similar to the proof of Lemma 5 in Chapter 2. Hence, we omit this form the current thesis.

By Lemma 6 and Proposition 8, for t, where $w_2|t$, the cost of a shortest path from $(t - w, 0)$ to (t, w) in $G''_{T,P}$ is bounded by $O(k_t)$. At the same time, any path in $G''_{T,P}$ from $(i, 0)$ to (t, w), $i \leq t$, has cost at least k_t. So we only need to find the minimal cost of a shortest path from any $(i, 0)$ to (t, w) in $G''_{T,P}$ to get an approximation of k_t.

To find the minimal cost, we reset to zero the cost of all horizontal edges $(i, 0) \to (i + 1, 0)$ in $G''_{T,P}$ to get a graph G. The graph G corresponds to taking the pattern matching graph $\tilde{G}_{T,P}$, removing from it all its diagonal edges and adding the shortcut edges. The cost of a path from $(0, 0)$ to (t, w) in G is the minimum over $i \leq t$ of the cost of a shortest path from $(i, 0)$ to (t, w) in $G''_{T,P}$.

Hence, we want to calculate the cost of the shortest path from $(0,0)$ to (t,w) for all t.[1] For this we will use a simple algorithm that will make a single sweep over the shortcut edges sorted by their origin and calculate the distances for $t = 0, \ldots, n$. The algorithm will maintain a data structure that at time t will allow to answer efficiently queries about the cost of the shortest path from $(0,0)$ to (t,j) for any $j \in \{0, \ldots, w\}$.

The data structure will consist of a binary tree with $w+1$ leaves. Each node is associated with a subinterval of $\{0, \ldots, w\}$ so that the j-th leaf (counting from left to right) corresponds to $\{j\}$, and each internal node corresponds to the union of all its children. We denote by I_v the interval associated with a node v. The depth of the tree is at most $1 + \log(w+1)$. At time t, query to the node v of the data structure will return the cost of the shortest path from $(0,0)$ to $(t, \max I_v)$ that uses some shortcut edge $(i,j) \to (i',j')$, where $j' \in I_v$. Each node v of the data structure stores a pair of numbers (c_v, t_v), where c_v is the cost of the relevant shortest path from $(0,0)$ to $(t_v, \max I_v)$ and t_v is the time it was updated the last time. (Initially this is set to $(\infty, 0)$.) At time $t \geq t_v$, the query to the node v returns $c_v + (t - t_v)$.

At time t to find the cost of the shortest path from $(0,0)$ to (t,j) we traverse the data structure from the root to the leaf j. Let $v_1, \ldots, v_\ell$ be the left children of the nodes along the path in which we continue to the right child. We query nodes $v_1, \ldots, v_\ell$ to get answers $a_1, \ldots, a_\ell$. The cost of the shortest paths from $(0,0)$ to (t,j) is $a = \min\{j, a_1 + (j - \max I_{v_1}), a_2 + (j - \max I_{v_2}), \ldots, a_\ell + (j - \max I_{v_\ell})\}$. As each query takes $O(1)$ time to answer, computing the shortest path to (t,j) takes $O(\log w)$ time.

The algorithm that outputs the cheapest cost of any path from $(0,0)$ to (t,w) in G will process the shortcut edges $(i,j) \to (i',j')$ one by one in the order of increasing i. The algorithm will maintain lists $L_0, \ldots, L_n$ of updates to the data structure to be made before time t. At time t the algorithm first outputs the cost of the shortest path from $(0,0)$ to (t,w). Then it takes each shortcut edge $(t,j) \to (t',j')$ one by one, $t < t'$. (The algorithm ignores shortcut edges where $t = t'$.) Using the current state of the data structure it calculates the cost c of a shortest path from $(0,0)$ to (t,j) and adds $(c+d, j')$ to list $L_{t'}$, where d is the cost of the shortcut edge $(t,j) \to (t',j')$.

After processing all edges starting at $(t,\cdot)$ the algorithm performs updates to the data structure according to the list L_{t+1}. Update (c,j) consists of traversing the tree from the root to the leaf j and in each node v updating its current values (c_v, t_v) to the new values $(c'_v, t+1)$, where $c'_v = \min\{c_v + t + 1 - t_v, c + \max I_v - j\}$.

Then the algorithm increments t and continues with further edges.

If the number of shortcut edges is m then the algorithm runs in time $O(n + m(\log m + \log w))$. First, it has to set-up the data structure, sort the edges by their origin and then it processes each edge. Processing each edge will require $O(\log w)$ time to find the min-cost path to the originating vertex and then later at time t' it will require time $O(\log w)$ to update the data structure. As there are $\widetilde{O}(\frac{n}{w} \cdot \frac{w}{\theta w_1} \cdot \frac{w}{w_1}) \leq \widetilde{O}(nw^{3/4})$ certified boxes in total the running time of the algorithm is as required.

The correctness of the algorithm is immediate from its description.

[1] Although, we really care only about t, where $w_2 | t+1$ as for all the other values of t we will approximate k_t by $k_{t'}$ for the previous multiple $t'+1$ of w_2.

3.4 Online Approximate Pattern Matching

In this section we describe the online algorithm from Theorem 3. In the online setting the pattern P is given while the text T arrives in online fashion. The main challenge of this setting is that at any point of time (other than the pattern) we are allowed to store a substring of the text of length just sub-linear in w. To overcome this situation the online algorithm is based on interleaved execution of the cover and min-cost path algorithms from Sections 3.2.2 and 3.3. Moreover we need to maintain some extra datastructure in a clever manner for the covering algorithm. Also to get the required space bound we use a little modified tree data structure for the min-cost path algorithm. For the online setting we use the same parameters as the offline one, but we set their values slightly differently: $w_1 = w^{11/18}$, $w_2 = w^{20/27}$, $d = w^{7/54}$, $\theta = w^{-1/9}$. Next, we describe the data structure used in the covering algorithm and the modified tree data structure for the min-cost path algorithm.

Covering algorithm data structure. For each substring T_m^0 of w consecutive input symbols of text T, and $j = \lceil \log 1/\theta \rceil, \ldots, 0$ the algorithm will maintain a set D'_j that stores the content of strings T_i^1 that reached Step 3 of the covering algorithm during processing of T_m^0. Moreover for each of such string T_i^1 the algorithm will also store a set $Y_{i,j}$ that contains the spans obtained in Step 4 while processing T_i^1. This is done as the whole w length string T_m^0 can't be stored at once. Moreover to bound the size of D'_j and $Y_{i,j}$, before adding a new T_i^1 that reached Step 3 of the covering algorithm to D'_j, we first ensure that no string close to T_i^1 is already contained in D'_j. Also after finishing each T_m^0 we discard all the information associated with it.

Modified tree data structure. Here we describe the modified tree data structure used for the min-cost path algorithm. Notice, every shortcut edge corresponds to some certified box. Our covering algorithm has $\log 1/\theta$ rounds where in any round the total number of possible vertical positions, where the bottom left corner or the top right corner point of any certified box might lie is bounded by $\frac{w}{\theta w_1}$. Next, we round up all the edit distance estimates to powers of two, hence in any certified box there are at most $2 \log w$ positions from which a shortcut edge might start or end. Therefore, the number of distinct vertical positions where these shortcut edges might originate from or lead to is bounded by $q = \frac{2w}{\theta w_1} \cdot \log 1/\theta \cdot \log w$. Thus the tree data structure of the min-cost path algorithm will ever perform updates to at most $q \log w$ distinct nodes. We do not need to store the nodes that are never updated, so the tree data structure will occupy only space $\widetilde{O}(\frac{w}{\theta w_1})$.

3.4.1 The online algorithm

Now we explain how to interleave the two phases to achieve required time and space bound. The algorithm processes the input text T in batches of w_2 symbols. Upon receipt of the t-th symbol we buffer the symbol, if t is not divisible by w_2 then the algorithm outputs the previous value k_{t-1} as the current value k_t and waits for the next symbol. Otherwise we received batch T_ℓ^2 of next w_2 symbols, for $\ell = t/w_2$, and we will proceed as follows.

The covering algorithm in the online setting is almost similar to the covering

algorithm offline setting. However here, we will execute the covering algorithm twice on each T^2_ℓ where during the first execution the only thing that we will send to the min-cost path algorithm are the certified boxes produced at Step 9, all other modifications to data structures will be discarded. During the second run of the algorithm on T^2_ℓ, we will preserve all modifications to D'_j's and other data structures except we will discard the certified boxes produced at Step 9 (we will not send them to the min-cost path algorithm as they are already sent in the first pass).

Covering algorithm. We now describe how the covering algorithm executes on each T^2_ℓ. The algorithm maintain sets S_j, $j = \lceil \log 1/\theta \rceil, \ldots, 0$ that are empty at the beginning. We partition T^2_ℓ into $T^1_g, \ldots, T^1_h$ of length w_1, where $g = (\ell - 1) \cdot \frac{w_2}{w_1} + 1$ and $h = g + \frac{w_2}{w_1} - 1$. For $i = g, \ldots, h$ we do the following. For each $j = \lceil \log 1/\theta \rceil, \ldots, 0$, set $\epsilon_j = 2^{-j}$. Check, whether T^1_j is at edit distance at most $2\epsilon_j w_1$ from some string $T^1_{i'}$ in D'_j. If it is then send the set of all the certified boxes $(I, J, 8\epsilon_j w_1)$ to the min-cost path algorithm, where I is the span of T^1_i in T and $J \in Y_{i',j}$. If it is not close to any string in D'_j then sample the relevant substring in P as in Step 2 and see how many of them are at edit distance $\leq \epsilon_j w_1$ from T^1_i. If at most $\frac{1}{2} \cdot c_0 \cdot \log n$ of the samples have their edit distance from T^1_i below $\epsilon_j w_1$ then put index i into S_j and continue for another j and then the next i. Otherwise we execute Step 4 of the algorithm to find set Y. (We always skip Step 3.) We put T^1_i into D_j and set $Y_{i,j}$ to Y. During the first execution of the covering algorithm, upon processing all j and i we will directly proceed to the sparse extension sampling part whereas after the second execution of the covering algorithm, we send all the certified boxes $(I, J, 8\epsilon_j w_1)$ to the min-cost path algorithm, where I is the span of T^1_i and $J \in Y_{i,j}$.

In the extension sampling part for each $j = \lceil \log 1/\theta \rceil, \ldots, 0$, we sample from the set S_j the strings T^1_ℓ in Step 7, and we proceed for them as in Steps 8–9. During the first execution of the covering algorithm, for each certified box (I, J, ℓ') produced in Step 9 round up ℓ' to the nearest larger or equal power of two and send the box to the min-cost path algorithm.

Min-cost path algorithm. The min-cost path algorithm receives certified boxes from the covering algorithm and it converts them into corresponding shortcut edges. The algorithm receives the certified boxes at two distinct phases.

Shortcut edges generated after the first execution of the covering algorithm correspond to boxes that were produced at Step 9. These edges are sorted by their originating vertex, stored, and processed at appropriate time steps during the next phase.

During the next phase the algorithm receives boxes $(I, J, 8\epsilon_j w_1)$, where I is the span of some T^1_i and $J \in Y_{i,j}$. It converts them into edges and upon receiving all the edges for a particular T^1_i, it sorts them according to their originating vertex. Then the min-cost path algorithm proceeds for times steps $(i-1) \cdot w_1$ to $i \cdot w_1 - 1$, and processes all stored shortcut edges that originate in these time steps. During these time steps it also updates its tree data structure as in the offline case. Again we use lists for storing pending updates. At any moment of time, the number of unprocessed edges and updates is bounded by the number of edges produced in Step 9 and edges produced for a particular string T^1_i. This is at most $\widetilde{O}(\frac{w}{\theta w_1})$. We conclude by the following lemma:

Lemma 9. *Let n and w be large enough integers. Let P be the pattern of length w, T be the text of length n (arriving online one symbol at a time), $1/w \leq \theta \leq 1$ be a real. Let $\theta w_1 \geq 1$, $w_1 \leq \theta w_2$, $w_1 | w_2$ and $w_2 | n$. With probability at least $1 - 1/poly(n)$ the online algorithm for pattern matching runs in amortized time $\widetilde{O}(\frac{w}{d} + \frac{ww_1}{w_2} + dw_2 + \frac{w}{w_1})$ per symbol and in succinct space $\widetilde{O}(w_2 + \frac{w}{d\theta} + \frac{w}{w_1\theta} + \frac{w^2}{\theta^2 w_1^2 d} + d)$.*

Proof. The running time of the online algorithm can be analyzed in a similar manner as the offline algorithm. The only difference is that here the covering and the min-cost path algorithm is interleaved. For each batch of w_2 symbols we run the covering algorithm twice with the modification that instead of executing Step 3 for each T_j^1 we check whether it is at distance at most $2\epsilon_j w_1$ from some string in D'_j. But this step takes amortized time $\widetilde{O}(\frac{w}{\epsilon_j w_1 d} \cdot w_1^2 \epsilon_j \cdot \frac{1}{w_1}) = \widetilde{O}(\frac{w}{d})$. Moreover the total number of certified boxes send by the covering algorithm to the min-cost path algorithm is the same in both the offline and the online algorithm. Hence the online algorithm has the amortized time of $\widetilde{O}(\frac{w}{d} + \frac{ww_1}{w_2} + dw_2 + \frac{w}{w_1})$ per symbol.

To determine the space complexity of the online algorithm we analyse the space used by the covering algorithm and the min-cost path algorithm separately. At any time the covering algorithm stores a batch of w_2 symbols which takes space $O(w_2)$. Next for $j = \lceil \log 1/\theta \rceil, \ldots, 0$ it stores set D'_j of strings T_i^1 that reached Step 3 of the covering algorithm. Each of these strings is of length w_1, hence requires $O(w_1)$ space. Moreover for each such string the algorithm stores set $Y_{i,j}$ of spans obtained at Step 4 and this require space $O(\frac{w}{\epsilon_j w_1})$. For each such string (as it reached Step 3), there exist at least $d/4$ relevant substrings of P which are at distance at most $\epsilon_j w_1$, and for any two strings of D'_j (as they are at distance more than $2\epsilon_j w_1$) these sets of relevant substrings of P are disjoint. Hence D'_j stores contents of at most $\frac{4w}{\epsilon_j w_1 d}$ different strings and the total space used by all D'_j and $Y_{i,j}$ is $\widetilde{O}(\frac{w}{\theta w_1 d} \cdot w_1 + \frac{w}{\theta w_1 d} \cdot \frac{w}{\theta w_1}) = \widetilde{O}(\frac{w}{\theta d} + \frac{w^2}{\theta^2 w_1^2 d})$. Maintaining sets S_j does not require any extra space as we store the whole batch of w_2 symbols. As argued before the tree data structure stored by the min-cost path algorithm occupies space $\widetilde{O}(\frac{w}{\theta w_1})$ and the list of edges can be stored in $\widetilde{O}(\frac{w}{\theta w_1} + d)$ space. Hence total succinct space used by the online algorithm is $\widetilde{O}(w_2 + \frac{w}{d\theta} + \frac{w}{w_1\theta} + \frac{w^2}{\theta^2 w_1^2 d} + d)$. □

For example, we can instantiate the above proposition for the parameters: $w_1 = w^{11/18}$, $w_2 = w^{20/27}$, $d = w^{7/54}$, $\theta = w^{-1/9}$, to get the following:

Theorem 31. *There is a constant $c \geq 1$ so that there is a randomized online algorithm that computes $(c, w^{8/9})$-approximation to approximate pattern matching in amortized time $O(w^{1-(7/54)})$ and succinct space $O(w^{1-(1/54)})$ with probability at least $1 - 1/poly(n)$.*

3.5 Conclusion and Bibliographical Notes

For the online pattern matching algorithm, it can be noticed that there is a clear tradeoff among the runtime, the succinct space used by the algorithm and the additive part of the approximation factor. Keeping the runtime fixed, decreasing the additive part of the approximation factor would increase the succinct space used whereas, keeping the additive error part fixed, decreasing the runtime would increase the succinct space used.

Open Problem. The online algorithm presented in this chapter have non trivial time and space complexity only for the case when the edit distance between the pattern and the text is high. Therefore, it will be a nice idea to extend our online approximation algorithm for the full range of edit distance which will be interesting from both theoretical and practical perspectives.

4. Combinatorial Lower Bounds of Boolean Matrix Multiplication

In this chapter we propose two combinatorial models for the Boolean Matrix Multiplication (BMM) problem, and prove lower bounds on computing BMM in these models. First, we give a relatively relaxed combinatorial model and prove that the time required for any algorithm to compute BMM is at least $\Omega(n^3/2^{O(\sqrt{\log n})})$. Subsequently, we propose a more general model capable of simulating the "Four Russians Algorithm". We prove a lower bound of $\Omega(n^{7/3}/2^{O(\sqrt{\log n})})$ for the BMM under this model.

4.1 Combinatorial Models

The first combinatorial model for BMM was given by Angluin [Ang76]. To compute the product of two $n \times n$ matrices $\mathcal{P}$ and $\mathcal{Q}$, the model allows to take bit-wise OR (*union*) of rows of the matrix $\mathcal{Q}$, in order to generate the individual rows of the resulting product matrix $\mathcal{PQ}$. Formally, for any $i \in [n]$, to generate the ith row of the product matrix $\mathcal{PQ}$ denoted by $\mathcal{PQ}_i$, check the set of indices of l's in the ith row of $\mathcal{P}$. Let, this be $\mathcal{P}_{i'}$. Then, $\mathcal{PQ}_i = \bigvee_{j \in \mathcal{P}_{i'}} \mathcal{Q}_j$, where $\bigvee$ represents the bit-wise OR of rows. The cost in this model is, the total number of unions taken. By a counting argument, Angluin [Ang76] showed that there are matrices $\mathcal{P}$ and $\mathcal{Q}$ such that the number of unions required must be $\Omega(n^2/\log n)$. This bound matches the number of unions taken by the Four Russians Algorithm, and in that sense the Four Russians Algorithm is optimal.

If the cost of computing each row union is counted as n, the total cost becomes $\Theta(n^3/\log n)$. The Four Russians Algorithm improves this bound to $O(n^3/\log^2 n)$ by leveraging "word-level parallelism" that enables computing each row union in time $O(n/\log n)$.

A possible approach to speed-up the Four Russians Algorithm would be to lower the cost of each union operation even further. The above analysis ignores the fact that we might be taking the union of rows with identical content multiple times. For example if $\mathcal{P}$ and $\mathcal{Q}$ are random matrices (as in the lower bound of Angluin) then each row of the resulting product matrix is an all-one row. Such rows will appear after taking an union of merely $O(\log n)$ different rows from $\mathcal{Q}$. An entirely naive algorithm would be to take unions of an all-one row with n possible rows of $\mathcal{Q}$ after only few unions. Hence, there would be only $O(n \log n)$ different unions to take, that has a total cost of $O(n^2 \cdot poly(\log n))$. The repetitions of unions can be easily detected by maintaining a short fingerprint for each row evaluated.

Our first model takes repetitions into account. Similar to Angluin, our focus is on the number of unions taken by the algorithm but we charge for each union operation differently. The natural cost of a union of two rows with values $u, v \in \{0,1\}^n$ is the minimum of the number of ones in u and v. This charging scheme is meaningful as one could use sparse set representation of u and v. In addition to that if unions of the rows (vectors) with identical content are taken multiple times we charge all of them only once, i.e., we charge the first one the actual cost

and all the additional unions are charged just by a unit cost. As we have argued, on random matrices $\mathcal{P}$ and $\mathcal{Q}$, BMM will cost $O(n^2 \log n)$ in this model. Our first lower bound shows that even in this model, there are matrices for which the cost of BMM is almost cubic.

The next natural operation one might allow to the algorithm is to divide rows into pieces. This is indeed what the Four Russians Algorithm and many other algorithms do. In the Four Russians Algorithm, this corresponds to the "word-level parallelism". Hence we might allow the algorithm to break rows into pieces, take unions of the pieces, and concatenate the resultant pieces back. In our more general model we set the cost of the partition and concatenation to be a unit cost, and we only allow to split a piece into continuous segments. More complex partitions can be simulated by performing many two-sided partitions and paying proportionally to the complexity of the partition. The cost of a union operation is again proportional to the smaller number of ones in the pieces, while repeated unions are charged for a unit cost. In this model one can implement the Four Russians Algorithm for the cost $O(n^3/\log^2 n)$, matching its usual cost. In the model without partitions the cost of the Four Russians Algorithm is $\Theta(n^3/\log n)$. In this model we are able to prove super-quadratic lower bound when we restrict that all partitions happen first, then unions take place, and then concatenations.

Perhaps, the characteristic property of "combinatorial" algorithms is that, from the run of such an algorithm one can extract a combinatorial proof (*witness*) for the resulting product. This is how we interpret our models. For given $\mathcal{P}$ and $\mathcal{Q}$ we construct a witness circuit that mimics the work flow of the algorithm. The circuit operates on rows of $\mathcal{Q}$ to derive the rows of the resulting matrix $\mathcal{PQ}$. The values flowing through the circuit are bit-vectors representing the values of resultant rows together with information about which union of which submatrix of $\mathcal{Q}$ the row represents. The gates can partition the vectors in pieces, concatenate them and take their union. For our lower bound we require that unions take place only after all partitions and before all concatenations. This ordering among the operations seems to be a reasonable restriction since we do not have to emulate the run of an algorithm step by step but rather see what it eventually produces. Also allowing the partition, union and concatenation operations to occur in arbitrary order could perhaps lead to only quadratic cost on all matrices. The proper modelling of combinatorial algorithms is a significant issue here: one wants a model that is strong enough to capture known algorithms (and other conceivable algorithms) but not so strong that it admits unrealistic quadratic algorithms.

4.2 Technique Overview

Our lower bounds are primarily dependent on are graphs that are derived from (r, t)-graphs of Rusza and Szemeredi [RS78]. Our graphs are tripartite with vertices split into parts A, B, C, where $|A| = |C| = n$ and $|B| = n/3$. The key property of these graphs is that there are almost quadratically many pairs $(a, c) \in A \times C$ that are connected via a single (*unique*) vertex from B. In terms of the corresponding matrices $\mathcal{P}$ (adjacency matrix representing the bipartite graph between A and B) and $\mathcal{Q}$ (adjacency matrix representing the bipartite graph between B and C) this means that in order to evaluate a particular row of their product we must take a union of very specific rows in $\mathcal{Q}$. The number of rows in

the union must be almost linear. Since $\mathcal{Q}$ is dense this might lead to an almost cubic cost for the whole algorithm provided different vertices in A are connected to different vertices in B so that we are required to take different unions.

Though, this is not apriori the case for the (r,t)-derived graph, we can easily achieve this by removing edges between A and B at random, each independently with probability $1/2$. This indeed make the neighborhoods of different vertices in A very different from each other. We call such a graph *diverse* (see a later section for a formal definition). It turns out that for our lower bound we need a slightly stronger property, not only that for different vertices of A, we need to take unions of different rows of $\mathcal{Q}$ but, also that for different vertices the results of these unions are different as well. We call this stronger property *unhelpfulness*. Using unhelpfulness of graphs we are able to derive the almost cubic lower bound on the simpler model. Unhelpfulness is a much more subtle property than diversity, and we crucially depend on the combinatorial properties of our graphs to derive it.

Next we tackle the issue of lower bounds for the partition model. This turns out to be a substantially harder problem. One needs unhelpfulness on different pieces of rows (restrictions to columns of $\mathcal{Q}$), that is, making sure that the result of union of some pieces of rows does not appear (too often) as a result of union of other pieces of rows. This is impossible to achieve in full generality. Roughly speaking what we can achieve is that, different parts of *any* witness circuit cannot produce the same results of unions. The key lemma that formalizes it (Lemma 14) shows that the results of unions obtained for a particular interval of columns in $\mathcal{Q}$ can be used at most $O(\log n)$ times on average in the rest of the circuit. This is a property of the graph which we refer to as that the graph *admitting only limited reuse*. This key lemma is technically complicated and challenging to prove (albeit elementary). Putting all the pieces together turns out to be also quite non trivial.

4.3 Notation and Preliminaries

For any integer $k \geq 1$, $[k] = \{1, \ldots, k\}$. For a vertex a in a graph G and a subset S of vertices of G, $\Gamma(a)$ are the neighbors of a in G, and $\Gamma_S(a) = \Gamma(a) \cap S$. (To emphasize which graph G we mean we may write $\Gamma_{S,G}(a)$.) A *subinterval* of set $C = \{c_1, c_2, \ldots, c_n\}$ is any set $K = \{c_i, c_{i+1}, \ldots, c_j\}$, for some $1 \leq i \leq j \leq |C|$. We interpret $\min K$ as i and $\max K$ as j. For a subinterval $K = \{c_i, c_{i+1}, \ldots, c_{i+\ell-1}\}$ of C and a vector $v = \{v_1, \ldots, v_\ell\} \in \{0,1\}^\ell$, $K \restriction_v$ denotes the set $\{c_j \in K;\ v_{j-i+1} = 1\}$. For a vector $v \in \{0,1\}^n$, $v \restriction_K = v_i, v_{i+1}, \ldots, v_{i+\ell-1}$. For a binary vector v, $|v|$ denotes the number of ones in v. For two sets S_1 and S_2, their set-union is represented by $S_1 \cup S_2$ and for two vectors $v_1, v_2 \in \{0,1\}^n$, $v_1 \bigvee v_2$ represents their bit-wise Or.

4.3.1 Matrices

We will denote matrices by calligraphic letters $\mathcal{P}, \mathcal{Q}, \mathcal{R}$. All matrices we consider are binary matrices. For integers i, j, $\mathcal{P}_i$ represents the i-th row of $\mathcal{P}$ and $\mathcal{P}_{i,j}$ represents the (i,j)-th entry of $\mathcal{P}$. Let $\mathcal{P}$ be an $n_A \times n_B$ matrix and $\mathcal{Q}$ be an $n_B \times n_C$ matrix, for some integers n_A, n_B, n_C. We associate matrices $\mathcal{P}, \mathcal{Q}$ with a

tripartite graph G. The vertices of G is the set $A \cup B \cup C$ where $A = \{a_1, \ldots, a_{n_A}\}$, $B = \{b_1, \ldots, b_{n_B}\}$ and $C = \{c_1, \ldots, c_{n_C}\}$. The edges of G are (a_i, b_k) for each i, k such that $\mathcal{P}_{i,k} = 1$, and (b_k, c_j) for each k, j such that $\mathcal{Q}_{k,j} = 1$. In this chapter we only consider graphs of this form. For a set of indices $S \subseteq B$, $\mathit{row}(\mathcal{Q}_S) = \bigvee_{i \in S} \mathcal{Q}_i$ is the bit-wise Or of rows of $\mathcal{Q}$ given by S.

4.3.2 Model

Circuit. A *circuit* is a directed acyclic graph W where each node (*gate*) has in-degree either zero, one or two. The *degree* of a gate is its in-degree and the *fan-out* is its out-degree. Degree one gates are called *unary gates* and degree two gates are *binary gates*. Degree zero gates are called *input gates*. For each binary gate g, $\mathit{left}(g)$ and $\mathit{right}(g)$ are its two predecessor gates. A computation of a circuit proceeds by passing values along the edges, where each gate processes its incoming values to decide on the value passed along its outgoing edges. The input gates have some predetermined values. The output of the circuit is the output value of some designated gate or gates.

Witness. Let $\mathcal{P}$ and $\mathcal{Q}$ be matrices of dimension $n_A \times n_B$ and $n_B \times n_C$, resp., with G be their associated graph. A *witness* for the matrix product $\mathcal{P} \times \mathcal{Q}$ is a circuit consisting of input gates, unary *partition gates*, binary union gates and binary *concatenation gates*. The values passed along the edges are triples (S, K, v), where $S \subseteq B$ identifies a set of rows of the matrix $\mathcal{Q}$, the subinterval $K \subseteq C$ identifies a set of columns of $\mathcal{Q}$, and v is $\mathit{row}(\mathcal{Q}_S) \restriction_K$, the restriction of $\mathit{row}(\mathcal{Q}_S)$ to the columns of K. Each input gate outputs $(\{b\}, C, Q_b)$ for some assigned $b \in B$. A partition gate with an assigned subinterval $K' \subseteq C$ on input (S, K, v) outputs *undefined* if $K' \not\subseteq K$ and outputs (S, K', v') otherwise, where $v' \in \{0,1\}^{|K'|}$ is such that for each $j \in [|K'|]$, $v'_j = v_{j + \min K' - \min K}$. A union gate on inputs $(S_\mathrm{L}, K_\mathrm{L}, v_\mathrm{L})$ and $(S_\mathrm{R}, K_\mathrm{R}, v_\mathrm{R})$ from its children outputs *undefined* if $K_\mathrm{L} \neq K_\mathrm{R}$, and outputs $(S_\mathrm{L} \cup S_\mathrm{R}, K_\mathrm{L}, v_\mathrm{L} \bigvee v_\mathrm{R})$ otherwise. A concatenation gate, on inputs $(S_\mathrm{L}, K_\mathrm{L}, v_\mathrm{L})$ and $(S_\mathrm{R}, K_\mathrm{R}, v_\mathrm{R})$ where $\min K_\mathrm{L} \leq \min K_\mathrm{R}$, is *undefined* if $\max K_\mathrm{L} + 1 < \min K_\mathrm{R}$ or $S_\mathrm{L} \neq S_\mathrm{R}$ or $\max K_\mathrm{L} > \max K_R$ and outputs $(S_\mathrm{L}, K_\mathrm{L} \cup K_\mathrm{R}, v')$ otherwise, where v' is obtained by concatenating v_L with the last $(\max K_\mathrm{R} - \max K_\mathrm{L})$ bits of v_R.

It is straightforward that whether a gate is undefined depends solely on the structure of the circuit but not on the actual values of $\mathcal{P}$ or $\mathcal{Q}$. We will say that the circuit is *structured* if union gates do not send values into partition gates, and concatenation gates do not send values into partition and union gates. Such a circuit first breaks rows of $\mathcal{Q}$ into parts, computes union of compatible parts and then assembles resulting rows using concatenation.

We say that a witness W is a *correct witness* for $\mathcal{P} \times \mathcal{Q}$ if W is structured, no gate has undefined output, and for each $a \in A$, there is a gate in W with output $(\Gamma_B(a), C, v)$ for $v = \mathit{row}(\mathcal{Q}_{\Gamma_B(a)})$.

Cost. The *cost* of the witness W is defined as follows. For each union gate g with inputs $(S_\mathrm{L}, K_\mathrm{L}, v_\mathrm{L})$ and $(S_\mathrm{R}, K_\mathrm{R}, v_\mathrm{R})$ and an output (S, K, v) we define its *row-class* to be $\mathit{class}(g) = \{v, v_\mathrm{L}, v_\mathrm{R}\}$. If T is a set of union gates from W, $\mathit{class}(T) = \{\{u, v, z\},\ \{u, v, z\}$ is the row-class of some gate in $T\}$. The cost of a row-class $\{u, v, z\}$ is $\mathit{min}\{|u|, |v|, |z|\}$. The cost of set T is $\sum_{\{u,v,z\} \in \mathit{class}(T)}$ *cost of* $\{u, v, z\}$. The *cost of witness* W is the number of gates in W plus the cost of the set containing all union gates in W.

We can make the following simple observation.

Proposition 32. *If W is a correct witness for $\mathcal{P} \times \mathcal{Q}$, then for each $a \in A$, there exists a collection of subintervals $K_1, \ldots, K_\ell \subseteq C$ such that $C = \bigcup_i K_i$ and for each $i \in [\ell]$, there is a union gate in W which outputs $(\Gamma_B(a), K_i, row(\mathcal{Q}_{\Gamma_B(a)}) \restriction_{K_i})$.*

Union and resultant circuit. One can look at the witness circuit from two separate angles which are captured in the next definitions. A *union circuit over a universe B* is a circuit with gates of degree zero and two where each gate g is associated with a subset $set(g)$ of B so that for each binary gate g, $set(g) = set(left(g))) \cup set(right(g))$. For integer $\ell \geq 1$, a *resultant circuit* is a circuit with gates of degree zero and two where each gate g is associated with a vector $row(g)$ from $\{0,1\}^\ell$ so that for each binary gate g, $row(g) = row(left(g)) \vee row(right(g))$, where $\vee$ is a coordinate-wise OR.

For a vertex $a \in A$ and a subinterval $K = \{c_i, c_{i+1}, \ldots, c_{i+\ell-1}\}$ of C, a *union witness* for (a, K) is a union circuit W over B with a single output gate g_{out} where $set(g_{\text{out}}) = \Gamma_B(a)$ and for each input gate g of W, $set(g) = \{b\}$ for some $b \in B$ connected to a.

Induced union witness. Let W be a correct witness for $\mathcal{P} \times \mathcal{Q}$. Pick $a \in A$ and a subinterval $K \subseteq C$. Let there be a union gate g in W with output $(\Gamma_B(a), K, row(\mathcal{Q}_{\Gamma_B(a)}) \restriction_K)$. An *induced union witness* for (a, K) is a union circuit over B whose underlying graph consists of copies of the union gates that are predecessors of g, and a new input gate for each input or partition gate that feeds into one of the union gates. They are connected in the same way as in W. For each gate g in the induced witness we let $set(g) = S$ whenever its corresponding gate in W outputs (S, K', v) for some K' and v. From the correctness of W it follows that each such $K' = K$ and the resulting circuit is a correct union witness for (a, K).

4.3.3 (r,t)-graphs: The hard instance

We will use special type of graphs for constructing matrices which are hard for our combinatorial model of Boolean matrix multiplication. For integers $r, t \geq 1$, an (r,t)-graph is a graph whose edges can be partitioned into t pairwise disjoint induced matchings of size r. Somewhat counter-intuitively as shown by Rusza and Szemeredi [RS78] there are dense graphs on n vertices that are (r,t)-graphs for r and t close to n.

Theorem 33 (Rusza and Szemerédi [RS78])**.** *For all large enough integers n, for $\delta_n = 1/2^{\Theta(\sqrt{\log n})}$ there is a $(\delta_n n, n/3)$-graph $G_n^{r,t}$.*

A more recent work of Alon, Moitra Sudakov [AMS12] provides a construction of a (r,t)-graphs on n vertices with $rt = (1-o(1))\binom{n}{2}$ and $r = n^{1-o(1)}$. The graphs of Rusza and Szemerédi are sufficient for us.

Let $G_n^{r,t}$ be the graph from the previous theorem and let $M_1, M_2, \ldots, M_{n/3}$ be the disjoint induced matchings of size $\delta_n n$. We define a tripartite graph G_n as follows: G_n has vertices $A = \{a_1, \ldots, a_n\}$, $B = \{b_1, \ldots, b_{n/3}\}$ and $C = \{c_1, \ldots, c_n\}$. For each i, j, k such that $(i,j) \in M_k$ there are edges (a_i, b_k) and (b_k, c_j) in G_n. The following immediate lemma states one of the key properties of G_n.

Lemma 10. *If $(i,j) \in M_k$ in $G_n^{r,t}$ then there is a unique path between a_i and c_j in G_n via b_k.*

For the rest of the chapter, we will fix the graphs G_n. Additionally, we will also use a graph $\widetilde{G}_n$ which is obtained from G_n by removing each edge between A and B independently at random with probability $1/2$. (Technically, $\widetilde{G}_n$ is a random variable.) When n is clear from the context we will drop the subscript n.

Fix some large enough n. Let $\mathcal{P}$ be the $n \times n/3$ adjaceny matrix between A and B in G and $\mathcal{Q}$ be the $n/3 \times n$ adjacency matrix between B and C in G. The adjacency matrix between A and B in $\widetilde{G}$ will be denoted by $\widetilde{\mathcal{P}}$. ($\widetilde{\mathcal{P}}$ is also a random variable.) The adjacency matrix between B and C in $\widetilde{G}$ is $\mathcal{Q}$.

We say that $c \in C$ is *unique* for A if there is exactly one $b \in B$ such that (a,b) and (b,c) are edges in G. The previous lemma implies that on average a has many unique vertices c in G, namely $\delta_n n/3$. For $S \subseteq C$, let $S[a]$ denote the set of vertices from S that are unique for a in G. E.g., $C[a]$ are all vertices unique for a. Let $\beta_a(S)$ denote the set of vertices from B that are connected to a and some vertex in $S[a]$. Notice, $|\beta_a(S)| = |S[a]|$. Since $\beta_a(\cdot)$ and $\cdot[a]$ depend on edges in graph G, to emphasise which graph we have in mind we may subscript them by G: $\beta_{a,G}(\cdot)$ and $\cdot[a]_G$.

For the randomized graph $\widetilde{G}$ we will denote by $S[a]'_{\widetilde{G}}$ the set of vertices from S that are unique for a in G and that are connected to a via B also in $\widetilde{G}$. (Thus, vertices from S that are not unique for a in G but became unique for a in $\widetilde{G}$ are not included in $S[a]'_{\widetilde{G}}$.) Let $\beta'_{a,\widetilde{G}}(S)$ denotes $\beta_a(S[a]'_{\widetilde{G}})$

4.3.4 Diverse and unhelpful graphs

In this section we define two properties of $\widetilde{G}$ that capture the notion that one needs to compute many different unions of rows of $\mathcal{Q}$ to calculate $\widetilde{\mathcal{P}} \times \mathcal{Q}$. The simpler condition stipulates that neighborhoods of different vertices from A are quite different. The second condition stipulates that not only the neighborhoods of vertices from A are different but also the unions of rows from $\mathcal{Q}$ that correspond to these neighborhoods are different as well.

Let G and $\widetilde{G}$ and $\mathcal{P}, \mathcal{Q}, \widetilde{\mathcal{P}}$ be as in the previous section. For integers $k, \ell \geq 1$, we say $\widetilde{G}$ is (k,ℓ)-*diverse* if for every set $S \subseteq B$ of size at least ℓ, no k vertices in A are all connected to all the vertices of S.

Lemma 11. *Let $c, d \geq 4$ be integers. The probability that $\widetilde{G}$ is $(c \log n, d \log n)$-diverse is at least $1 - n^{-(cd/2)\log n}$.*

Proof. $\widetilde{G}$ is not $(c \log n, d \log n)$-diverse if for some set $S \subseteq B$ of size $\ell \geq d \log n$, and some k-tuple of distinct vertices $a_1, \ldots, a_k \in A$ for $k = c \log n$, each vertex a_i is connected to all vertices from S in $\widetilde{G}$. The probability that all vertices of a given k-tuple $a_1, \ldots, a_k \in A$ are connected to all vertices in a given set S of size $\ell \geq d \log n$ in $\widetilde{G}$ is at most $2^{-k\ell}$. (The probability is zero if some a_i is not connected to some vertex from S in G.) Hence, the probability that there is some set $S \subseteq B$ of size $\ell \geq d \log n$, and some k-tuple of distinct vertices $a_1, \ldots, a_k \in A$ where each vertex a_i is connected to all vertices from S in $\widetilde{G}$ is bounded by:

$$\sum_{\ell=d\log n}^{n}\binom{n}{\ell}\cdot\binom{n}{k}\cdot 2^{-\ell k} \le \sum_{\ell=d\log n}^{n} n^{\ell}\cdot n^{k}\cdot 2^{-\ell k}$$
$$\le \sum_{\ell=d\log n}^{n} 2^{(\ell+k)\log n-\ell k}$$
$$\le \sum_{\ell=d\log n}^{n} 2^{-\ell k/2}$$
$$\le \sum_{\ell=d\log n}^{n} \frac{1}{n^{(cd/2)\log n}}$$

where the second inequality follows from $c, d \geq 4$. □

For $S \subseteq B$, $a \in A$ and a subinterval $K \subseteq C$, we say that S is *helpful for a on K* if there exists a set $S' \subseteq \beta'_{a,\widetilde{G}}(K)$ such that $|S| \leq |S'|$ and $C[a]_G \cap (K \restriction_{row(\mathcal{Q}_S)}) = C[a]_G \cap (K \restriction_{row(\mathcal{Q}_{S'})})$. In other words, the condition means that $row(\mathcal{Q}_S)$ and $row(\mathcal{Q}_{S'})$ agree on coordinates in K that correspond to vertices unique for a in G. This is a necessary precondition for $row(\mathcal{Q}_S) \restriction_K = row(\mathcal{Q}_{S'}) \restriction_K$ which allows one to focus only on the *hard-core* formed by the unique vertices. In particular, if for some $S'' \subseteq \Gamma_{B,\widetilde{G}}(a)$ in $\widetilde{G}$, $row(\mathcal{Q}_S) \restriction_K = row(\mathcal{Q}_{S''}) \restriction_K$, then $S' = S'' \cap \beta'_{a,\widetilde{G}}(K)$ satisfies $C[a]_G \cap (K \restriction_{row(\mathcal{Q}_S)}) = C[a]_G \cap (K \restriction_{row(\mathcal{Q}_{S'})})$. (See the proof below.)

For integers $k, \ell \geq 1$, we say $\widetilde{G}$ is *(k,ℓ)-unhelpful on K* if for every set $S \subseteq B$ of size at least ℓ, there are at most k vertices in A for which S is helpful on K.

Lemma 12. *Let $c, d \geq 4$ be integers. Let and $K = \{c_i, c_{i+1}, \ldots, c_{i+\ell-1}\}$ be a subinterval of C. The probability that $\widetilde{G}_n$ is $(c\log n, d\log n)$-unhelpful on K is at least $1 - n^{-(cd/2)\log n}$.*

Proof. Take any set $S \subseteq B$ of size $\ell \geq d\log n$ and arbitrary vertices $a_1, \ldots, a_k \in A$ for $k = c\log n$. Consider $row(\mathcal{Q}_S) \restriction_K$ and some $i \in [k]$. Since edges between B and C are always the same in $\widetilde{G}$, $row(\mathcal{Q}_S) \restriction_K$ is always the same in $\widetilde{G}$. If S is helpful on K for a_i then there exists $S_i \subseteq \beta'_{a,\widetilde{G}}(K)$ such that $|S_i| \geq \ell$ and $C[a]_G \cap (K \restriction_{row(\mathcal{Q}_{S_i})}) = C[a]_G \cap (K \restriction_{row(\mathcal{Q}_S)})$. It turns out that given a_i, the possible S_i is uniquely determined by $row(\mathcal{Q}_S) \restriction_K$. Whenever $row(\mathcal{Q}_S) \restriction_K$ has one in a position c that corresponds to a unique vertex of a in G, $row(\mathcal{Q}_{S_i}) \restriction_K$ must have one there as well so the corresponding b must be in S_i. Conversely, whenever $row(\mathcal{Q}_S) \restriction_K$ has zero in a position c that corresponds to a unique vertex of a in G, $row(\mathcal{Q}_{S_i}) \restriction_K$ must have zero there as well so the corresponding b is not in S_i. The probability that $S_i \subseteq \beta'_{a,\widetilde{G}}(K)$ is $2^{-|S_i|}$.

Hence, the probability over choice of $\widetilde{G}$ that S is helpful for a_i on K is at most $2^{-\ell}$. For different a_i's this probability is independent as it only depends on edges between a_i and B. Thus the probability that S is helpful for $a_1, \ldots, a_k$ is at most $2^{-\ell k}$.

There are at most $\binom{n}{\ell}\cdot\binom{n}{k}$ choices for the set S of size ℓ and $a_1, \ldots, a_k$. Hence, the probability that $\widetilde{G}$ is not $(c\log n, d\log n)$-unhelpful on K is at most:

$$\sum_{\ell=d\log n}^{n}\binom{n}{\ell}\cdot\binom{n}{k}\cdot 2^{-\ell k} \leq \sum_{\ell=d\log n}^{n} n^{\ell}\cdot n^{k}\cdot 2^{-\ell k}$$
$$\leq \sum_{\ell=d\log n}^{n} 2^{(\ell+k)\log n-\ell k}$$
$$\leq \sum_{\ell=d\log n}^{n} 2^{-\ell k/2}$$
$$\leq \sum_{\ell=d\log n}^{n} \frac{1}{n^{(cd/2)\log n}}$$

where the third inequality follows from $c, d \geq 4$. □

4.4 Union Circuits

In this section we prove a cubic lower bound for our first model. Precisely our goal is to prove the following theorem:

Theorem 34. *There is a constant $c > 0$ such that for all n large enough there are matrices $\mathcal{P} \in \{0,1\}^{n\times n/3}$ and $\mathcal{Q} \in \{0,1\}^{n/3\times n}$ such that any correct witness for $\mathcal{P}\times\mathcal{Q}$ consisting of only union gates has cost at least $n^3/2^{c\sqrt{\log n}}$.*

Here by consisting of only union gates we mean consisting of union gates and input gates. Our almost cubic lower bound on the cost of union witnesses is an easy corollary to the following lemma.

Lemma 13. *Let n be a large enough integer and $\widetilde{G}_n$ be the graph from Section 4.3.3, and $\widetilde{\mathcal{P}}, \mathcal{Q}$ be its corresponding matrices. Let W be a correct witness for $\widetilde{\mathcal{P}}\times Q$ consisting of only union gates. Let $\widetilde{\mathcal{P}}$ have at least m ones. Let each row of $\mathcal{Q}$ have at least r ones. If $\widetilde{G}$ is (k,ℓ)-unhelpful on C for some integers $k, \ell \geq 1$ then any correct witness for $\widetilde{\mathcal{P}}\times\mathcal{Q}$ consisting of only union gates has cost at least $(mr/2k\ell) - nr/k$.*

Proof. Let W be a correct witness for $\widetilde{\mathcal{P}}\times\mathcal{Q}$ consisting of only union gates. For each gate g of W with output (S, C, v), for some v, define $set(g) = S$. Consider $a \in A$. Let g_a be a gate of W such that $set(g_a) = \Gamma_{B,\widetilde{G}}(a)$ (which equals $\beta'_{a,\widetilde{G}}(C)$). Take a maximal set D_a of gates from W, descendants of g_a, such that for each $g \in D_a$, $|set(g)| \geq \ell$ and either $|set(left(g))| < \ell$ or $|set(right(g))| < \ell$, also for $g \neq g' \in D_a$, $\{set(g), set(left(g)), set(right(g))\} \neq \{set(g'), set(left(g')), set(right(g'))\}$.

Notice, if $g \neq g' \in D_a$ then $class(g) \neq class(g')$. This is because for any sets $S \neq S' \subseteq set(g_a)$, $row(\mathcal{Q}_S) \neq row(\mathcal{Q}_{S'})$. (Say, $b \in S \setminus S'$, then there is 1 in $\mathcal{Q}_b$ which corresponds to a vertex c unique for a. Thus, $row(\mathcal{Q}_S)_c = 1$ whereas $row(\mathcal{Q}_{S'})_c = 0$.)

We claim that since D_a is maximal, $|D_a| \geq \lfloor |set(g_a)|/2\ell \rfloor$. We prove the claim. Assume $set(g_a) \geq 2\ell$ otherwise there is nothing to prove. Take any $b \in set(g_a)$ and consider a path $g_0, g_1, \ldots, g_p = g_a$ of gates in W such that $set(g_0) = \{b\}$. Since $|set(g_0)| = 1$, $|set(g_a)| \geq 2\ell$ and $set(g_{i-1}) \subseteq set(g_i)$, there is some g_i with $|set(g_i)| \geq \ell$ and $|set(g_{i-1})| < \ell$. By maximality of

D_a there is some gate $g \in D_a$ such that $\{set(g), set(left(g)), set(right(g))\} = \{set(g_i), set(left(g_i)), set(right(g_i))\}$. Hence, b is in $set(left(g))$ or $set(right(g))$ of size $< \ell$. Thus

$$set(g_a) \subseteq \bigcup_{g \in D_a;\ |set(left(g))| < \ell} set(left(g)) \cup \bigcup_{g \in D_a;\ |set(right(g))| < \ell} set(right(g))$$

Hence, $|set(g_a)| \leq 2\ell \cdot |D_a|$ and the claim follows.

For a given a, gates in D_a have different row-classes. Since $\widetilde{G}$ is (k, ℓ)-unhelpful on C, the same row-class can appear in D_a only for at most k different a's. (Say, there were $a_1, a_2, \ldots, a_{k+1}$ vertices in A and gates $g_1 \in D_{a_1}, \ldots, g_{k+1} \in D_{a_{k+1}}$ of the same row-class. For each $i \in [k+1]$, $set(g_i) \subseteq \Gamma_{B,\widetilde{G}}(a_i) = \beta'_{a_i,\widetilde{G}}(C)$ and $|set(g_i)| \geq \ell$. The smallest $set(g_i)$ would be helpful for $a_1, a_2, \ldots, a_{k+1}$ contradicting the unhelpfulness of $\widetilde{G}$.) Since

$$\sum_a |D_a| \geq \sum_a \lfloor |set(g_a)|/2\ell \rfloor \geq \frac{m}{2\ell} - n,$$

witness W contains gates of at least $(m/2k\ell) - n/k$ different row-classes. Since, each $\mathcal{Q}_b$ contains at least r ones, the total cost of W is as claimed. □

Proof of Theorem 34. Let $\widetilde{G}_n$ be the graph from Section 4.3.3, and $\widetilde{\mathcal{P}}, \mathcal{Q}$ be its corresponding matrices. Let $r = n\delta_n$. By Lemma 12, the graph $\widetilde{G}$ is $(5\log n, 5\log n)$-unhelpful on C with probability at least $1 - 1/n^{\log n}$, and by Chernoff bound, $\widetilde{\mathcal{P}}$ contains at least $nr/10$ ones with probability at least $1 - exp(n)$. So with probability at least $1/2$, $\widetilde{\mathcal{P}}$ has $m \geq nr/10$ ones while $\widetilde{G}$ is $(5\log n, 5\log n)$-unhelpful on C. By the previous lemma, any witness for $\widetilde{\mathcal{P}} \times \mathcal{Q}$ is of cost $(nr^2/25\log n) - nr/5\log n$. For large enough n, this is at least $nr^2/50\log n = n^3\delta_n^2/50\log n$, and the theorem follows. □

4.5 Circuits with Partitions

In this section, our goal is to prove the lower bound $\Omega(n^{7/3}/2^{O(\sqrt{\log n})})$ on the cost of a witness for matrix product when the witness is allowed to partition the columns of $\mathcal{Q}$. Namely we prove the following:

Theorem 35. *For all n large enough there are matrices $\mathcal{P} \in \{0,1\}^{n \times n/3}$ and $\mathcal{Q} \in \{0,1\}^{n/3 \times n}$ such that any correct witness for $\mathcal{P} \times \mathcal{Q}$ has cost at least $\Omega(n^{7/3}/2^{O(\sqrt{\log n})})$.*

We provide a brief overview of the proof first. The proof builds on ideas seen already in the previous section but also requires several additional ideas. Consider a correct witness for $\widetilde{\mathcal{P}} \times \mathcal{Q}$. We group its union gates based on their corresponding subinterval of C. If there are many vertices in A that use many different subintervals (roughly $\Omega(n^{4/3})$ in total) the lower bound follows by counting the total number of gates in the circuit using diversity of $\widetilde{G}$ (Lemma 17). If there are many vertices in A which use only few subintervals (less than roughly $O(n^{1/3})$ each) then these subintervals must be large in size on average (about $n^{2/3}$) and contain lots of vertices from C unique for their respective vertices from A.

In this case we divide the circuit (its union gates) based on the associated subinterval, and we calculate the contribution of each part separately. To do that we have to limit the amount of reuse of a given row-class within each part, and also among distinct parts. Within each part we limit the amount of reuse using a similar technique to Lemma 13 based on unhelpfulness of the graph (Lemma 16). However, for distinct parts we need a different tool which we call *limited reuse*. Limited reuse is somewhat different than unhelpfulness, specifically in the type of guarantee we get. The guarantee is weaker in some sense, as we are not able to limit the reuse of a row-class for each single gate but only the total reuse of row-classes of all the gates in a particular part of the circuit. On average the reuse is again roughly $O(\log n)$.

However, the number of gates in a particular part of the circuit might be considerably larger than the number of gates we are able to charge for work in that part. In general, we are able to charge only those gates that already made some non-trivial progress in the computation (as otherwise the gates could be reused heavily.) We overcome this obstacle by balancing the size (total number of gates) of the part against the number of *chargeable* gates in that part.

If the total number of gates in the part is at least $n^{1/3}$-times larger than the total number of chargeable gates, we charge the part for its size. Otherwise we charge it for work. Each chargeable gates contributes by about $n^{2/3}$ units of work or more, however this can be reused almost $n^{1/3}$-times elsewhere. Either way, approximately $\Omega(n^{7/3})$ of work must be done in total. Now we present the actual proof.

In order to prove the theorem we introduce few more definitions. Let G_n and $\widetilde{G}_n$ and $\mathcal{P}, \mathcal{Q}, \widetilde{\mathcal{P}}$ be as in the Section 4.3.3. All witness circuits in this section are with respect to $\widetilde{\mathcal{P}} \times \mathcal{Q}$ (i.e., $\widetilde{G}_n$). Let c_0 and c_1 be some constants that we will fix later.

The following definition aims to separate contribution from different rows within a particular subcircuit. A witness circuit may benefit from taking a union of the same row of $\mathcal{Q}$ multiple times to obtain a particular union. This could help various gates to attain the same row-class. In order to analyze the cost of the witness we want to effectively prune the circuit so that contribution from each row of $\mathcal{Q}$ is counted at most once. The following definition captures this prunning.

Let W be a union circuit over B with a single vertex g_{out} of out-degree zero (*output gate*). The *trimming* of W is a map that associates to each gate g of W a subset $\mathit{trim}(g) \subseteq \mathit{set}(g)$ such that $\mathit{trim}(g_{\text{out}}) = \mathit{set}(g_{\text{out}})$ and for each non-input gate g, $\mathit{trim}(g) = \mathit{trim}(\mathit{left}(g)) \dot{\cup} \mathit{trim}(\mathit{right}(g))$. For each circuit W, we fix a canonical trimming that is obtained from $\mathit{set}(\cdot)$ by the following process: For each $b \in \mathit{set}(g_{\text{out}})$, find the left-most path from g_{out} to an input gate g such that $b \in \mathit{set}(g)$, and remove b from $\mathit{set}(g')$ of every gate g' that is *not* on this path.

Given the trimming of a union circuit W we will focus our attention only on gates that contribute substantially to the cost of the computation. We call such gates *chargeable* in the next definition. For a vertex $a \in A$ and a subinterval $K \subseteq C$, let W be a union witness for (a, K) with its trimming. We say a gate g in W is (a, K)-*chargeable* if $|\mathit{trim}(g) \cap \beta'_{a,\widetilde{G}}(K)| \geq c_0 \log n$ and $\mathit{trim}(\mathit{left}(g)) \cap \beta'_{a,\widetilde{G}}(K)$ and $\mathit{trim}(\mathit{right}(g)) \cap \beta'_{a,\widetilde{G}}(K)$ are both different from $\mathit{trim}(g) \cap \beta'_{a,\widetilde{G}}(K)$. (a, K)-*Chargeable descendants of* g are (a, K)-chargeable gates g' in W where $\mathit{trim}(g') \cap$

$\beta'_{a,\widetilde{G}}(K) \subseteq \mathit{trim}(g) \cap \beta'_{a,\widetilde{G}}(K)$. Observe that the number of (a,K)-chargeable descendants of a gate g is at most $|\mathit{trim}(g) \cap \beta'_{a,\widetilde{G}}(K)| + 1 - c_0 \log n$.

From a correct witness for $\widetilde{\mathcal{P}} \times \mathcal{Q}$, we extract some induced union circuit W for (a,K) and some resultant circuit W'. We say that a gate g from W *is compatible* with a gate g' from W' if $\mathit{row}(\mathcal{Q}_{\mathit{set}(g)}) \restriction_K = \mathit{row}(g')$.

We want to argue that chargeable gates corresponding to gates of a given correct witness have many different row-classes. Hence, we want to bound the number of gates whose result is compatible with each other. This is akin to the notion of helpfulness. In the case of helpfulness we were able to limit the repetition of the same row-class for individual gates operating on the same subinterval of columns of $\mathcal{Q}$. In addition to that we need to limit the occurence of the same row-class for gates that operate on distinct subintervals. As opposed to the simpler case of helpfulness, we will need to focus on the global count of row-classes that can be reused elsewhere from gates operating on the same subinterval. The next definition encapsulates the desired property of $\widetilde{G}$.

For $a, a' \in A$ and subintervals K, K' of C, we say that (a,K) and (a',K') are *independent* if either $a \neq a'$ or $K \cap K' = \emptyset$. A resultant circuit W' over $\{0,1\}^\ell$ is consistent with $\mathcal{Q}$, if there exists a subinterval $K \subseteq C$ of size ℓ, such that for each input gate g of W', $\mathit{row}(g) = \mathcal{Q}_b \restriction_K$ for some $b \in B$. We say that $\widetilde{G}$ *admits only limited reuse* if for any resultant circuit W' of size at most n^3 which is consistent with $\mathcal{Q}$ and any correct witness circuit W for $\widetilde{\mathcal{P}} \times \mathcal{Q}$, the number of gates in any induced union witnesses $W_1, \ldots, W_s$ for any pairwise independent pairs $(a_1, K_1), \ldots, (a_s, K_s)$ that are chargeable and compatible with some gate in W' is at most $c_1 |W'| \log n$.

We will show that with high probability $\widetilde{G}$ admits only limited reuse.

Lemma 14. *Let $c_1 \geq 7$ and $c_0 \geq 20$ be constants. Let n be a large enough integer. Let $\widetilde{G}_n$ be the graph from Section 4.3.3, and $\widetilde{\mathcal{P}}, \mathcal{Q}$ be its corresponding matrices. The probability that $\widetilde{G}$ admits only limited reuse is at least $1 - 1/n$.*

To prove this lemma we will analyze individual pairs (a,K) and their induced union circuits.

Lemma 15. *Let $c_0 \geq 5$ be a constant. Let $1 \leq m, \ell \leq n$ be integers. Let W' be arbitrary resultant circuit over $\{0,1\}^\ell$ with at most n^3 gates. Let $a \in A$ and K be a subinterval of C of size ℓ. Let E_m be the event that there is a union witness W for (a,K) in which at least m (a,K)-chargeable gates are compatible with gates in W'. There exists another event E'_m that depends only on the presence or absence of edges between a and $\beta'_{a,\widetilde{G}}(K)$ in $\widetilde{G}$ such that E_m implies E'_m, and the probability $\Pr[E_m] \leq \Pr[E'_m] \leq 2^{-m-(c_0-5)\log n}$.*

For independent pairs (a,K) and (a',K'), the events E'_m from Lemma 15 are independent so we will be able to bound the probability of them occuring simultaneously.

Proof. We claim that if E_m occurs then there must be a t-tuple of gates $g'_1, \ldots, g'_t$ in W', where $t \leq n^3$, such that the set:

$$X = \bigcup_{j=1}^{t} \beta'_{a,\widetilde{G}}(K \restriction_{\mathit{row}(g'_j)})$$

satisfies

1. $|X| \geq m + t(c_0 - 1)\log n$, and
2. edges between a and X are all present in $\widetilde{G}$.

The existence of such a t-triple is our event E'_m. E'_m has probability at most

$$\sum_{t=1}^{n^3} |W'|^t \cdot 2^{-(m+t(c_0-1)\log n)} \leq 2^{-m} \cdot \sum_{t=1}^{n^3} 2^{-t(c_0-4)\log n)} \leq 2 \cdot 2^{-m-(c_0-4)\log n},$$

as there are $|W'|^t$ choices for the t-tuple $g'_1, \ldots, g'_t$, and the probability that all edges between a and X are present in $\widetilde{G}$ is $2^{-|X|}$. The lemma follows in such a case as E'_m only depends on the presence or absence of edges between a and $\beta'_{a,\widetilde{G}}(K)$ in G. So we only need to prove the existence of the t-tuple of required properties whenever E_m occurs.

Let S be a set of m (a, K)-chargeable gates in W which are compatible with some gate in W'. For each gate $g \in S$, let $\mathit{trim}'(g) = \mathit{trim}(g) \cap \beta'_{a,\widetilde{G}}(K)$. Let $g_1, \ldots, g_s$ be all the gates in S that are maximal with respect to inclusion of their sets $\mathit{trim}'(g_i)$. All the gates in S are among the chargable descendants of $g_1, \ldots, g_s$. Observe:

1. For any $i \neq j \in [s]$, $\mathit{trim}'(g_i) \cap \mathit{trim}'(g_j) = \emptyset$, and
2. for any $i \in [s]$, the number of (a, K)-chargeable descendants of g_i is at most $|\mathit{trim}'(g_i)| + 1 - c_0 \log n$.

The first item holds as $\mathit{trim}(g_i)$ are either related by inclusion or disjoint, the second item holds by the definition of (a, K)-chargeable gates. This implies:

$$|S| = m \leq \left(\sum_{i=1}^{s} |\mathit{trim}'(g_i)| \right) + s - sc_0 \log n.$$

Pick the smallest set of gates $g'_1, \ldots, g'_t$ in W' so that each of the gates $g_1, \ldots, g_s$ is compatible with at least one of them. Clearly, $t \leq s$. Let g_{out} be the top-most gate of W. By definition, $\mathit{set}(g_{\text{out}}) = \Gamma_{B,\widetilde{G}}(a)$. If g_i is compatible with g'_j then $\mathit{row}(\mathcal{Q}_{\mathit{set}(g_i)}) \restriction_K = \mathit{row}(g'_j)$. Hence, $\beta'_{a,\widetilde{G}}(K \restriction_{\mathit{row}(g'_j)}) \subseteq \mathit{set}(g_i) \subseteq \mathit{set}(g_{\text{out}})$, and $\mathit{trim}'(g_i) \subseteq \beta'_{a,\widetilde{G}}(K \restriction_{\mathit{row}(g'_j)})$ by properties of vertices unique for a. For the set $X = \bigcup_{j=1}^{t} \beta'_{a,\widetilde{G}}(K \restriction_{\mathit{row}(g'_j)})$, the former implies that all edges between a and X must be present in $\widetilde{G}$. The latter implies $|X| \geq \sum_{i=1}^{s} |\mathit{trim}'(g_i)| \geq m + s(c_0 - 1)\log n \geq m + t(c_0 - 1)\log n$. Hence, $g'_1, \ldots, g'_t$ is a tuple of required properties and the lemma follows. □

Proof of Lemma 14. Fix arbitrary resultant circuit W' of size at most n^3 consistent with $\mathcal{Q}$. Fix $s \in [n^3]$ and pairwise independent $(a_1, K_1), (a_2, K_2), \ldots, (a_s, K_s)$, where each $a_i \in A$ and K_i is a subinterval of C. Fix a sequence of positive integers $m_1, m_2, \ldots, m_s$ such that $\sum_{i \in [s]} m_i \geq c_1 |W_1| \log n$.

Take $\widetilde{G}$ at random. Let W be some correct witness for $\widetilde{\mathcal{P}} \times \mathcal{Q}$ which for each $i \in [s]$, contains an induced union witness W_i for (a_i, K_i) such that W_i contains

at least m_i (a_i, K_i)-chargeable gates compatible with gates in W'. W might not exist. Our goal is to estimate the probability that such a union witness W exists.

Let E_i be the event that there is some union witness W_i for (a_i, K_i) which contains at least m_i (a_i, K_i)-chargeable gates compatible with gates in W'. We can associate to E_i also an event E_i' from Lemma 15. Since $(a_1, K_1), (a_2, K_2), \ldots,$ (a_s, K_s) are pairwise independent, the events E_i' are mutually independent. Thus we can bound the probability of the existence of W by

$$\begin{aligned}
\Pr[W_1, W_2, \ldots, W_s \text{ exists}] &= \Pr[E_1 \cap E_2 \cdots \cap E_s] \\
&\leq \Pr[E_1' \cap E_2' \cdots \cap E_s'] \\
&= \prod_{i \in [s]} \Pr[E_i'] \\
&\leq \prod_{i \in [s]} 2^{-m_i - (c_0 - 5)\log n} \\
&\leq 2^{-c_1 |W'| \log n - s(c_0-5)\log n}
\end{aligned}$$

where the second equality follows from the independence and the second inequality follows from Lemma 15.

This probability is for a fixed choice of W, s, a_i's, K_i's, and m_i's. For a given size $t = |W'|$ there are at most $(t^2+n)^t n^2$ choices for W' consistent with $\mathcal{Q}$. There are also at most $(n^3)^s$ choices for $(a_1, K_1), \ldots, (a_s, K_s)$ and at most $(c_1 n^3 \log n)^s$ choices for $m_1, \ldots, m_s$.

Thus the probability that $\widetilde{G}$ does not admit only limited reuse is at most:

$$\sum_{t=1}^{n^3} \sum_{s=1}^{n^3} n^{3s+2} (t^2+n)^t \cdot (c_1 n^3 \log n)^s \cdot 2^{-c_1 |W'| \log n - s(c_0-5)\log n} \leq 1/n.$$

□

4.5.1 The cost of chargeable gates in a partition

For $\widetilde{\mathcal{P}}, \mathcal{Q}$ from Section 4.3.3, let W be a correct witness for $\widetilde{\mathcal{P}} \times Q$. We say that a gate g in W is (a, K)-chargeable if g corresponds to an (a, K)-chargeable gate in the lexicographically first induced union witness for (a, K) in W.

The next lemma lower bounds the contribution of chargeable gates to the total cost of the witness. It is similar in spirit to Lemma 13 and its proof is similar. It focuses on union gates dealing with a particular subinterval $K \subseteq C$.

Lemma 16 (Partition version)**.** *Let n be a large enough integer and $\widetilde{G}_n$ be the graph from Section 4.3.3, and $\widetilde{\mathcal{P}}, \mathcal{Q}$ be its corresponding matrices. Let $r, k > 1$ be integers and $\ell = c_0 \log n$. Let W be a correct witness for $\widetilde{\mathcal{P}} \times Q$. Let $K \subseteq C$ be a subinterval. Let $R \subseteq B$ be such that for each b in R, $\mathcal{Q}_b \restriction_K$ has at least r ones. Let $A' \subseteq A$ be such that for each $a \in A'$, $|R \cap \beta'_{a,\widetilde{G}}(K)| \geq 2\ell$. Let $m = \sum_{a \in A'} |R \cap \beta'_{a,\widetilde{G}}(K)|$. If $\widetilde{G}$ is (k, ℓ)-unhelpful on K then there is a set D of union gates in W such that*

1. *Each gate in D is (a, K)-chargeable for some vertex $a \in A$, and*

2. *The number of different row-classes of gates in D of cost $\geq r$ is at least $m/4k\ell$.*

Proof. Pick $a \in A'$ for which there is an induced union witness in W. Fix the lexicographically first union witness W_a for (a, K). Let $trim(\cdot)$ be its trimming. Define $trim'(g) = trim(g) \cap R \cap \beta'_{a,\widetilde{G}}(K)$. For the output gate g_a of W_a, $trim'(g_a) = R \cap \beta'_{a,\widetilde{G}}(K)$ as $\beta'_{a,\widetilde{G}}(K) \subseteq \Gamma_{B,\widetilde{G}}(a) = trim(g_a)$. Take a maximal set D_a of gates from W_a such that for each $g \in D_a$, $|trim'(g)| \geq \ell$, $trim'(left(g)), trim'(right(g)) \subsetneq trim'(g)$ and either $|trim'(left(g))| < \ell$ or $|trim'(right(g))| < \ell$, and furthermore for $g \neq g' \in D_a$, $\{trim'(g), trim'(left(g)), trim'(right(g))\} \neq \{trim'(g'), trim'(left(g')), trim'(right(g'))\}$. Clearly, gates in D_a are (a, K)-chargeable.

Notice, if $g \neq g' \in D_a$ then $class(g) \neq class(g')$. (Here we identify g with its corresponding gate in W.) This is because for any sets $S \neq S' \subseteq trim'(g_a)$, $row(\mathcal{Q}_S) \restriction_K \neq row(\mathcal{Q}_{S'}) \restriction_K$. (Say, $b \in S \setminus S'$, then there is 1 in $\mathcal{Q}_b \restriction_K$ which corresponds to a vertex c unique for a. Thus, $row(\mathcal{Q}_S)_c = 1$ whereas $row(\mathcal{Q}_{S'})_c = 0$.) Also, if $g \in D_a$ and $\{u, v, z\}$ is its row-class then $|u|, |v|, |z| \geq r$, since $trim'(g), trim'(left(g)), trim'(right(g))$ have non-empty intersection with R.

We claim that since D_a is maximal, $|D_a| \geq \lfloor |trim'(g_a)|/2\ell \rfloor$. We prove the claim. Assume $trim'(g_a) \geq 2\ell$ otherwise there is nothing to prove. Take any $b \in trim'(g_a)$ and consider a path $g_0, g_1, \ldots, g_p = g_a$ of gates in W_a such that $trim'(g_0) = \{b\}$. Since $|trim'(g_0)| = 1$, $|trim'(g_a)| \geq 2\ell$ and $trim'(g_{i-1}) \subseteq trim'(g_i)$, there is some g_i with $|trim'(g_i)| \geq \ell$ and $|trim'(g_{i-1})| < \ell$. Say $g_{i-1} = left(g_i)$. Since $b \in trim'(left(g_i)) \subsetneq trim'(g_i)$ and $trim'(left(g_i)) \dot\cup trim'(right(g_i)) = trim'(g_i)$, $trim'(right(g_i)) \neq trim'(g_i)$. By maximality of D_a there is some gate $g \in D_a$ such that $\{trim'(g), trim'(left(g)), trim'(right(g))\} = \{trim'(g_i), trim'(left(g_i)), trim'(right(g_i))\}$. Hence, b is in $trim'(left(g))$ or $trim'(right(g))$ of size $< \ell$. Thus

$$trim'(g_a) \subseteq \bigcup_{g \in D_a;\, |trim'(left(g))| < \ell} trim'(left(g)) \cup \bigcup_{g \in D_a;\, |trim'(right(g))| < \ell} trim'(right(g))$$

Hence, $|trim'(g_a)| \leq 2\ell \cdot |D_a|$ and the claim follows.

Set $D = \bigcup_a D_a$. For a given a, gates in D_a have different row-classes. Each of the row-classes is of cost at least r. Indeed, for each $g \in D_a$, $trim'(g), trim'(left(g))$ and $trim'(right(g))$ are all non-empty, so each of the $set(g), set(left(g))$ and $set(right(g))$ contains some b with $|\mathcal{Q}_b \restriction_K| \geq r$.

Since $\widetilde{G}$ is (k, ℓ)-unhelpful on K, the same row-class can appear in D_a only for at most k different a's. (Say, there were $a_1, a_2, \ldots, a_{k+1}$ in A and gates $g_1 \in D_{a_1}, \ldots, g_{k+1} \in D_{a_{k+1}}$ of the same row-class. For each $i \in [k+1]$, $trim'(g_i) \subseteq set(g_i) \cap \beta'_{a_i,\widetilde{G}}(K)$ so $|set(g_i) \cap \beta'_{a_i,\widetilde{G}}(K)| \geq \ell$. The smallest $set(g_i) \cap \beta'_{a_i,\widetilde{G}}(K)$ would be helpful for $a_1, a_2, \ldots, a_{k+1}$ contradicting the unhelpfulness of $\widetilde{G}$.)

Since

$$\sum_{a \in A'} |D_a| \geq \sum_{a \in A'} \lfloor |trim'(g_a)|/2\ell \rfloor \geq \frac{m}{4\ell},$$

D contains chargeable gates of at least $m/4k\ell$ different row-classes with cost $\geq r$. □

4.5.2 Large number of partitions

If the witness for $\widetilde{\mathcal{P}} \times Q$ involves many subintervals for many vertices we will apply the next lemma.

Let n be a large enough integer and $\widetilde{G}_n$ be the graph from Section 4.3.3 with associated matrices $\widetilde{\mathcal{P}}, \mathcal{Q}$. Let W be a witness for $\widetilde{\mathcal{P}} \times \mathcal{Q}$. By Proposition 32 each $a \in A$ is associated with distinct subintervals $K_{a,1}, \ldots, K_{a,\ell_a} \subseteq C$, for some ℓ_a, such that $C = \bigcup_{j \in [\ell_a]} K_{a,j}$ and there are union gates $g_{a,1}, \ldots, g_{a,\ell_a}$ in W such that $g_{a,j}$ outputs $(\Gamma_{B,\widetilde{G}}(a), K_{a,j}, v_{a,j})$ for some $v_{a,j} \in \{0,1\}^{|K_{a,j}|}$.

Lemma 17. *Let W, ℓ_a's, $K_{a,j}$'s, $g_{a,j}$'s be as above. Let $c, d \geq 4$ and $\ell, r \geq 1$ be integers where r is large enough. Let $L = \{a \in A, \ell_a \geq \ell \ \&\ |\Gamma_{B,\widetilde{G}}(a)| \geq r\}$. If $\widetilde{G}$ is $(c \log n, d \log n)$-diverse then the size of W is at least $r\ell \cdot |L|/(2cd \log^2 n)$.*

Proof. If two union gates g, g' have outputs (S, K, v) and (S', K', v'), resp., where $K \neq K'$, then g and g' cannot have a descendant union gate in common. (This follows from consistency of union gates.) Consider a union gate g in W that outputs (S, K, v), where $|S| \geq d \log n$. Let $T = \{(a,j) \in L \times [\ell], g_{a,j}$ has g among its descendants$\}$. Clearly, for all $(a,j) \in T$, $g_{a,j}$ outputs $(\Gamma_{B,\widetilde{G}}(a), K, v_{a,j})$ for some $v_{a,j} \in \{0,1\}^{|K|}$. Hence, $(a,j), (a,j') \in T$ implies $j = j'$. For each $(a,j) \in T$, $S \subseteq \Gamma_{B,\widetilde{G}}(a)$. By $(c \log n, d \log n)$-diversity of $\widetilde{G}$, $|T| \leq c \log n$.

For each $(a,j) \in L \times [\ell]$, $g_{a,j}$ has at least $\lfloor |\Gamma_{B,\widetilde{G}}(a)|/d \log n \rfloor \geq \lfloor r/d \log n \rfloor \geq r/2d \log n$ distinct descendant union gates g' with output $(S', K_{a,j}, v')$, where $|S'| \geq d \log n$ and v' is arbitrary. (Each such g' has distinct S'.) Each such g' can be descendant of at most $c \log n$ gates $g_{a,j}$ by the bound on T. Hence, there are at least $|L| \cdot \ell r/(2cd \log^2 n)$ distinct union gates in W. □

4.5.3 Density lemma

We state here an auxiliary *density* lemma. The proof is standard but we include it for completeness.

Lemma 18. *Let $n, r \geq 1$ be integers. Let $K_1, \ldots, K_r$ be a collection of (not necessarily distinct) subintervals of $[n]$. Let $u_1 \in K_1, u_2 \in K_2, \ldots, u_r \in K_r$ be distinct elements. Denote $U = \{u_1, \ldots, u_r\}$. There are at least $r/2$ sets K_i such that $|K_i \cap U| \geq |K_i| r/4n$.*

Proof. Any subinterval I of $[n]$ is called sparse if $|I \cap U| < |I| r/4n$. Let $I_1, \ldots, I_k$ be the set of all sparse subintervals of $[n]$. We want to prove $|\cup_{i \in [k]} (I_i \cap U)| < r/2$. Denote $S = \cup_{i \in [k]} I_i$. Suppose $I' = \{I'_1, \ldots, I'_\ell\}$ be the minimal set of sparse subintervals covering all sparse subintervals. Thus $\cup_{i \in [k]} I_i = \cup_{j \in [\ell]} I'_j$. We claim, any $u \in S$ is covered by at most two subintervals of I'. As otherwise assume there are more than two subintervals in I' which contain u. All these intervals must have some nontrivial intersection including u. Among them consider the two, having the left most starting point and right most end point in $[n]$. It can be easily seen that the union of these two intervals covers all the other intervals and hence the minimality of I' is violated. Therefore our claim follows. Now as $|S| \leq n$, from the previous claim we get $\sum_{j \in [\ell]} |I'_j| \leq 2n$. The construction also implies $\bigcup_{i \in [k]} (I_i \cap U) = \bigcup_{j \in [\ell]} (I'_j \cap U)$. By the sparsity of the intervals, there are at most $2n \times r/4n = r/2$ elements of U contained in $\bigcup_{j \in [\ell]} (I'_j \cap U)$ and therefore

in $\bigcup_{i\in[k]}(I_i \cap U)$. Thus each of the $r/2$ elements of $U \setminus S$ are contained only in subintervals which are not sparse. Hence each set K_i associated with these $r/2$ elements satisfies $|K_i \cap U| \geq |K_i|r/4n$. □

4.5.4 The main proof

In this section we prove the lower bound $\approx n^{7/3}$ on the cost of witnesses for matrix product.

Theorem 36. *For all n large enough there are matrices $\mathcal{P} \in \{0,1\}^{n\times n/3}$ and $\mathcal{Q} \in \{0,1\}^{n/3\times n}$ such that any correct witness for $\mathcal{P} \times \mathcal{Q}$ has cost at least $\Omega(n^{7/3}/2^{O(\sqrt{\log n})})$.*

Let n be large enough and let $\widetilde{G}_n$ be the graph from Section 4.3.3. Set $c = 5, d = 5, c_0 = 7, c_1 = 20$. Let $r = \delta_n n$, $s = n^{1/3}$, $\ell = n^{1/3}$. With probability at least $1/2$, $\widetilde{G}$ is simultaneously $(c\log n, d\log n)$-diverse (Lemma 11), $(c\log n, d\log n)$-unhelpful on each of the $\binom{n}{2}$ subintervals of C (Lemma 12), admits only limited reuse (Lemma 14), and $\sum_{a\in A} |C[a]'_{\widetilde{G}}| \geq nr/3$ (by Chernoff inequality).

Let W be a correct witness for $\widetilde{G}$, our goal is to lower bound its cost.

We will define a sequence of sets $T_6 \subseteq T_5 \subseteq \cdots \subseteq T_1 \subseteq A \times C$ of pairs of (a, c) where c is unique for a.

1. **(Unique pairs.)** $T_1 = \{(a, c), a \in A, c \in C[a]'_{\widetilde{G}}\}$ is the set of pairs of a and its unique vertices. By assumption, $|T_1| \geq nr/3$.

2. **(Removing sparse a's.)** Let $A_2 = \{a \in A, |\Gamma_{B,\widetilde{G}}(a)| \geq r/6\}$. Clearly, $|A_2| \geq r/6$. Let $T_2 = T_1 \cap (A_2 \times C) = \{(a, c) \in T_1, |\Gamma_{B,\widetilde{G}}(a)| \geq r/6\}$. By an averaging argument, $|T_2| \geq nr/6$.

3. **(Removing a's with many subintervals K.)** For each $a \in A_2$, let $K_{a,1}, \ldots, K_{a,\ell_a}$ be obtained from Proposition 32. Let $A_3 = \{a \in A_2, \ell_a \leq \ell\}$ and $A'_2 = A_2 \setminus A_3$. If $|A'_2| \geq r/12$ we apply Lemma 17 to conclude that the size of W is at least $\frac{r}{12} \cdot \frac{r}{6} \cdot \frac{\ell}{2cd\log^2 n} \geq r^2\ell/150\log^2 n$. In this case we are done.

   Otherwise consider the case $|A_3| \geq r/12$. Let $T_3 = T_2 \cap (A_3 \times C)$. Since $|A'_2| < r/12$, $|T_3| \geq nr/12$.

4. **(Removing small subintervals K.)** For each $a \in A_3$, let $K'_{a,1}, \ldots, K'_{a,\ell'_a}$ be the subsequence of $K_{a,1}, \ldots, K_{a,\ell_a}$ obtained by removing each $K_{a,j}$ of size smaller than $r/24\ell$. So $\ell'_a \leq \ell_a \leq \ell$, and each $|K'_{a,j}| \geq r/24\ell$.

   We remove pairs (a, c) from T_1 not covered by large $K_{a,j}$'s: Let $T_4 = T_3 \cap (\bigcup_{a\in A_3}\{a\} \times (\bigcup_{j\in[\ell'_a]} K'_{a,j}))$. By the size and number of the removed subintervals K, $|T_4| \geq nr/24$.

5. **(Removing overlapping subintervals K.)** For each $a \in A_3$ find a collection of disjoint subintervals $K''_{a,1}, \ldots, K''_{a,\ell''_a}$ such that $|T_4 \cap (\{a\} \times \bigcup_{j\in[\ell''_a]} K''_{a,j})| \geq |T_4 \cap (\{a\} \times \bigcup_{j\in[\ell'_a]} K'_{a,j})|/2$. (Such a collection exists: Take the smallest subcollection of $K'_{a,1}, \ldots, K'_{a,\ell'_a}$ which covers their entire union.

Each point from $T_4 \cap (\{a\} \times \bigcup_{j\in[\ell'_a]} K'_{a,j})$ is contained in at most two intervals of this subcollection. Order the subcollection by the smallest element in each interval. Either the subset of intervals on odd positions in this ordering or on even positions has the required property.)

Let $T_5 = T_4 \cap (\bigcup_{a\in A_3}(\{a\} \times \bigcup_{j\in[\ell''_a]} K''_{a,j}))$. By the choice of removed subintervals K, $|T_5| \geq nr/48$.

6. **(Disregarding sparse sub-rows of $\mathcal{Q}$.)** For $b \in B$, let $T_b = \{(a,c) \in T_5,\ \{b\} = \beta_a(\{c\})$, i.e. b is on the path between a and $c\}$. Let $K(a,c)$ denote $K''_{a,j}$ such that $c \in K''_{a,j}$. (This is uniquely defined as $K''_{a,j}$'s are disjoint.)

   Set $B_6 = \{b \in B,\ |T_b| \geq r/48\}$. For $b \in B_6$, $(a,c) \in T_b$, we say that the triple (b,a,c), is *dense* if $|\mathcal{Q}_b \restriction_{K(a,c)}| \geq \frac{r}{24\ell} \cdot \frac{r}{48} \cdot \frac{1}{4n}$. By Lemma 18, for at least half of the pairs $(a,c) \in T_b$, (b,a,c) is dense.

   Let $T_6 = \bigcup_{b\in B_6}\{(a,c) \in T_5,\ (b,a,c)$ is dense$\}$.

   There are at most $\frac{r}{48} \cdot \frac{n}{3}$ pairs removed from T_5 because $b \notin B_6$ and at most half of the remaining points afterwards. So $|T_6| \geq |T_5|/3 \geq nr/150$.

Given sets T_6 and A_3, B_6 obtained so far we proceed with the final calculation.

Consider a subinterval $K \subseteq C$. Let $A_K = \{a \in A_3,\ |T_6 \cap (\{a\} \times K)| \geq 2c_0 \log n\}$, and $R_K = \{b \in B,\ |\mathcal{Q}_b \restriction_K| \geq \frac{r^2}{4800\cdot n\ell}\}$.

Let $m_K = \sum_{a\in A_K} |R_K \cap \beta'_{a,\widetilde{G}}(K)|$ and $m'_K = \sum_{a\in A_K} |T_6 \cap (\{a\} \times K)|$. Since for any $a \in A_3$, $|T_6 \cap (\{a\} \times K)| \leq |R_K \cap \beta'_{a,\widetilde{G}}(K)|$, $m'_k \leq m_k$. Also, $\sum_K m_K \geq \sum_K m'_K \geq |T_6| - 2c_0 n\ell \log n \geq |T_6|/2$.

Let s_K be the number of union gates in W that correspond to K (i.e., that output (S,K,v) for some S and v.)

Consider subintervals $K \subseteq C$, where $sm_K \leq s_K$, $\mathcal{C} = \{K \subseteq C,\ K$ subinterval, $sm_K \leq s_K\}$. If $\sum_{K\in\mathcal{C}} m_K \geq |T_6|/4$ then the $|W| \geq snr/600$ so we are done. So consider the case when $\sum_{K\in\mathcal{C}'} m_K \geq |T_6|/4$, where $\mathcal{C}' = \{K \subseteq C,\ K$ subinterval, $sm_K > s_K\}$. For each $K \in \mathcal{C}'$, apply Lemma 16 ($R \leftarrow R_K, A' \leftarrow A_K, k \leftarrow c\log n, \ell \leftarrow c_0 \log n, r \leftarrow r^2/4800n\ell, D \to D_K$) to obtain the set D_K of gates with at least $m_K/4cc_0 \log^2 n$ row-classes of cost at least $r^2/4800n\ell$. As all the gates in D_K are (a,K)-chargeable for some $a \in A$, by definition of limited reuse, their row-class coincides with at most $c_1 s_K \log n \leq m_K s c_1 \log n$ other gates in $\bigcup_{K'\in\mathcal{C}'} D_{K'}$. Thus, $\bigcup_{K'\in\mathcal{C}'} D_{K'}$ contains gates of at least $\sum_{K'\in\mathcal{C}'} m_{K'}/4c_1 cc_0 s \log^3 n$ row-classes each of cost at least $r^2/4800n\ell$. This contributes to the cost of W by at least $\frac{r^2}{4800n\ell} \cdot \frac{nr}{2400c_1 cc_0 s \log^3 n} = \Theta(r^3/\ell s \log^3 n)$. The theorem follows.

4.6 Conclusion and Open Problems

In this chapter we have proposed combinatorial models for BMM and proved lower bounds for them. Though these models are not strong enough to simulate the recent combinatorial approaches, it provides a starting point for more comprehensive analysis of the limitations of combinatorial algorithms. Therefore it will be interesting to extend our current model to a more generalized one, capable of simulating all recent and naturally anticipated combinatorial algorithms and prove better lower bounds for them. Towards this direction one idea would be to allow arbitrary partitioning of the rows. Another interesting attempt will be to improve

the combinatorial upper bound. Though getting a truly sub-cubic one will be a break through, coming up with an upper bound of the form $\Omega(n^3/2^{O(\sqrt{\log n})})$ will be quite interesting as well.

5. Weight Tolerant Subgraph for Single Source Shortest Path

In this chapter we provide the construction of a sparse subgraph of a given directed weighted graph with a designated source vertex, such that the subgraph preserves the distance from the source to all other vertices as long as the total weight increment of all the edges is bounded by k.

5.1 Preliminaries

5.1.1 Definitions

We start with the following basic definitions. For any positive integer r, we denote the set $\{1, 2, \cdots, r\}$ by $[r]$. Throughout this chapter we use $\mathbb{N}$ to indicate the set of natural numbers including zero. For any k-dimensional vector σ and $i \in [k]$, $\sigma(i)$ denotes the value of the i-th coordinate of σ. Given a directed graph $G = (V, E)$ on $n = |V|$ vertices and $m = |E|$ edges with a weight function w defined on the set of edges and a source vertex $s \in V$, a destination vertex $t \in V$, we use the following notations throughout this chapter.

- $V(G), E(G)$: the set of vertices and edges of G respectively.
- $w(P)$: weight of any path P.
- $dist_{G,w}(x, y)$: the sum of the weight of the edges appearing on the shortest path between two vertices x and y in G when weight of each edge is defined by the weight function w.
- $G + (u, v)$: the graph obtained by adding an edge (u, v) to the graph G.
- $G \setminus F$: the graph obtained by removing the set of edges F from the graph G.
- $In(A)$: the set of all vertices in $V \setminus A$ having an outgoing edge to some vertex in set $A \subseteq V$.
- $Out(A)$: the set of all vertices in $V \setminus A$ having an incoming edge from $A \subseteq V$.
- $In\text{-}Edge(A)$: the set of edges incoming to $A \subseteq V$.
- $Out\text{-}Edge(A)$: the set of edges out of $A \subseteq V$.
- $P[x, y]$: the subpath of a path P from a vertex x to y.
- $P \circ Q$: the path formed by concatenating paths P and Q assuming the fact that last vertex of P is same as first vertex of Q.
- $E(f)$: the set of edges e such that under a given flow f, $f(e) \neq 0$.

- $MaxFlow(G, S, t)$: any maximum valued flow in G from a source set S to t.
- G_{short} : the shortest path subgraph of G, i.e., union of all shortest $s - t$ paths in G.
- $ShortMaxFlow(G, S, t)$: any maximum valued flow returned by $MaxFlow(G_{short}, S, t)$.

We now define the notion of k-FTRS (from [BCR16]) and k-WTSS with respect to a fixed vertex t.

Definition 37 (k-FTRS(t))**.** *[BCR16] Given a graph G, a source vertex $s \in V(G)$, another vertex $t \in V(G)$ and an integer $k \geq 1$, a subgraph $H_t = (V(G), E')$ where $E' \subseteq E(G)$ is said to be k-FTRS(t) of G if for any set F of k edge failures, the following holds: t is reachable from s in $G \setminus F$ if and only if t is reachable from s in $H_t \setminus F$.*

Definition 38 (k-WTSS(t))**.** *Given a graph G with weight function w, a source vertex $s \in V(G)$, another vertex $t \in V(G)$ and an integer $k \geq 1$, a subgraph $H_t = (V(G), E')$ where $E' \subseteq E(G)$ is said to be k-WTSS(t) of G if for any weight increment function $I : E(G) \to \mathbb{N}$ such that $\sum_{e \in E(G)} I(e) \leq k$, the following holds: for the weight function defined by $w'(e) = w(e) + I(e)$ for all $e \in E(G)$, $dist_{G,w'}(s, t) = dist_{H_t,w'}(s, t)$.*

Following are the alternative definitions of k-FTRS and k-WTSS in terms of k-FTRS(t) and k-WTSS(t) respectively.

Definition 39. *[BCR16] A subgraph H of G is a k-FTRS if and only if it is a k-FTRS(t) for all $t \in V(G)$.*

Definition 40. *A subgraph H of G is a k-WTSS if and only if it is a k-WTSS(t) for all $t \in V(G)$.*

5.1.2 Max-flow and farthest min-cut

Our algorithm for constructing the k-WTSS heavily exploits the connection between min-cut, max-flow and the number of edge disjoint paths present in a graph. We start with the following well known theorem.

Theorem 41. *In any graph with unit capacity on edges, there is a flow of value r from a source set S to a destination vertex t if and only if there exist r edge disjoint paths that originate from the set S and terminate at t.*

Now we define (S, t)-min-cut in a graph G.

Definition 42 ((S, t)-min-cut)**.** *In any graph G an (S, t)-cut is a set of edges $C \subseteq E(G)$ such that every path from any vertex $s \in S$ to t must pass through some edge in C. An (S, t)-cut is called (S, t)-min-cut if it has the smallest size among all other (S, t)-cuts.*

Any (S,t)-cut C partitions the vertex set $V(G)$ into two subsets $A(C)$ and $B(C)$ where $A(C)$ is the set of all the vertices reachable from S in $G \setminus C$ and $B(C) = V(G) \setminus A(C)$. Note that $S \subseteq A(C)$ and $t \in B(C)$. From now onwards, we assume this pair of vertex sets $(A(C), B(C))$ to be output of a function $Partition(G, C)$.

For our purpose we do not just consider any (S,t)-min-cut, instead we consider the farthest one.

Definition 43 (Farthest Min Cut)**.** *Let S be a source set and t be a destination vertex in any graph G and suppose for any (S,t)-min-cut C, $(A(C), B(C)) = Partition(G, C)$. Any (S,t)-min-cut C_{far} is called* farthest min-cut*, denoted by $FMC(G, S, t)$, if for any other (S,t)-min-cut C, it holds that $A(C) \subsetneq A(C_{far})$.*

The following lemma given by Ford and Fulkerson establishes the uniqueness of farthest min-cut and also provides an algorithm to compute it.

Lemma 19. *[FF] Suppose f be a max-flow in G from any source set S to t and G_f be the corresponding residual graph. If B be the set of vertices from which there is a path to t in G_f and $A = V(G) \setminus B$, then the set C of edges that start at A and terminate at B is the unique farthest (S,t)-min-cut.*

Now we state following three important properties of farthest min-cut from [BCR16].

Lemma 20. *[BCR16] For any graph G, a source set S and a destination vertex t, let $C = FMC(G, S, t)$ and $(A, B) = Partition(G, C)$ and for any edge $(s, b) \in (S \times B)$ define $G' = G + (s, b)$. Then the value of max-flow from S to t in G' is exactly one greater than that in G and $FMC(G', S, t) = C \cup \{(s, b)\}$.*

Lemma 21. *[BCR16] Consider a source vertex s and a destination vertex t in any graph G. Let $S \subseteq V(G)$ such that $s \in S$ and $t \notin S$ and f be a max-flow from S to t in G and $C = FMC(G, S, t)$, $(A, B) = Partition(G, C)$. Then we can always find a max-flow f_{max} from s to t such that $E(f_{max}) \subseteq E(A) \cup E(f)$.*

Lemma 22. *[BCR16] Consider a source vertex s and a destination vertex t in any graph G. Let $S \subseteq V(G)$ such that $s \in S$, $t \notin S$ and $(A, B) = Partition(G, FMC(G, s, t))$. Then for $(A', B') = Partition(G, FMC(G, S, t))$, $B' \subseteq B$.*

5.1.3 Overview of the construction

In this section we provide a overview of our construction of the k-WTSS of graph G. We first simplify the construction by concentrating just on a single vertex t and describe how to build a subgraph that preserves the distance between the source s and t under weight increment. We call such a subgraph a k-WTSS(t). We then argue that using Locality Lemma (see Section 5.3), these subgraphs for different vertices can be combined to construct the k-WTSS of graph G. Locality Lemma actually says slightly more, that if we can construct such a subgraph for a particular vertex t with an additional property that in-degree of t in the subgraph is bounded by some value c, then we can build a k-WTSS of size at most cn.

So from now onwards we focus on constructing k-WTSS(t) for some vertex t. Let us first consider a toy example which provides a motivation behind our

technique. Let the input graph be G with weight function w and $dist_{G,w}(s,t) = d$. Suppose G is such that, it can be decomposed into $k+1$ disjoint subgraphs $G_0, \cdots, G_k$ where for $0 \leq i \leq k-1$, G_i contains all the $s-t$ paths of weight $d+i$ present in G and any $s-t$ path in G_i has weight exactly $d+i$. In general such a decomposition might not exist. However, if it exists then it is not hard to get such a decomposition. Now given such a decomposition, we compute a k-FTRS(t) of G_0 and for $i \in [k-1]$, a $(k-i-1)$-FTRS(t) of G_i and then take the union of these subgraphs. We claim that the obtained subgraph will be a k-WTSS(t). Let, under the incremented weight function, the shortest distance between s and t is $d+j$ for $1 \leq j < k$. The bound on j is justified because for $j = 0, k$ it is trivial as we have included k-FTRS(t). Without loss of generality we further assume that no weight increment happens on the edges of the current shortest path. The justification of this assumption is provided later. But this indeed implies that the current shortest path is contained in subgraph G_j and any $s-t$ path contained in graph G_i, where $0 \leq i \leq k-1$ has weight at least $d+j+1$. Therefore, the total increase in weight on the edges of G_j is bounded by $k-(j+1)$. Now as our weight increment function is integer valued the previous bound implies that at most $k-(j+1)$ many edges of G_j are affected by weight increment. Note, this is the place where integer valued increment plays a crucial role. However by our construction, we include the $(k-j-1)$-FTRS(t) of G_j in our subgraph. Thus even if we consider the removal of those affected edges then also as there is a path in G_j on which there is no weight increment, by the definition of $(k-j-1)$-FTRS(t) there will be a surviving path included in our constructed subgraph. This proves the correctness. Also by the result of [BCR16], in-degree of t in each $(k-i-1)$-FTRS(t) of G_i is bounded by 2^{k-i-1} and hence total in-degree of t in the constructed k-WTSS(t) is bounded by 2^{k+1}. Hence we get a k-WTSS of size at most $2^{k+1}n$.

Now we consider the more general situation where for a given a graph, there does not exists any nice decomposition of above type. In general, if we consider a subgraph by taking all the $s-t$ paths upto some specific weight, then that particular subgraph may also contain a $s-t$ path with larger weight and at this point, all the above arguments fail. However, the nice thing is that instead of any arbitrary weight, if we build a subgraph containing all the shortest $s-t$ paths then the subgraph will not contain any $s-t$ path of larger weight. Now if we use the construction of k-FTRS on this shortest path subgraph, then we can guarantee the preservation of distances as long as the distances do not change even after the weight increment. Though if the distance changes, we can not say anything. This is the main challenge that we overcome in our algorithm. For that purpose we use the properties of the farthest min-cut of the shortest path subgraph.

In [BCR16] Baswana *et al.* used the concept of farthest min-cut to construct k-FTRS. In their work, they first compute a series of k farthest min-cuts by taking source sets in some nested fashion. Then they calculate a max-flow between the final source set and the vertex t and keep only those incoming edges of t that have non-zero flow. We further exploit their technique in order to design our algorithm. We consider the shortest path subgraph and then compute a series of farthest min-cuts similar to [BCR16]. However as mentioned in the last paragraph, in this way we can construct the k-FTRS(t) only for the shortest path subgraph. Now let us

take the farthest min-cut considering s as source. Since it is a (s,t)-cut of the shortest path subgraph, removal of it destroys all the shortest $s-t$ paths present in the original graph. Now if we again compute the shortest path subgraph, we will get a subgraph containing only $s-t$ paths of weight $d+i$, for some $i>0$. Then we can process this new subgraph as before to compute a sequence of k farthest min-cuts and remove the first one. We proceed in this way until we reach a point where all the $s-t$ paths have weight at least $d+k$.

Now let us compare the situation with what we have already discussed with our toy example. Removal of cut edges only helps us to generate some subgraph of each of G_i's. However computing k-FTRS(t) of just some subgraph of G_i may not be sufficient to get k-WTSS(t). Thus for each G_i, we try to consider a lot of subgraphs so that when we combine k-FTRS(t) of all of them, we get the same advantage that we got from computing $(k-i-1)$-FTRS(t) of G_i in the toy example. One way of getting a lot of subgraphs of G_i is to try out removal of different cuts (not just the farthest one). Obviously we cannot try for all possible cuts, because there can be too many. Moreover, each time to reach at a subgraph of weight $d+i$ we may have to remove a series of cuts. As a result we may end up with exponentially many choices on removal of cuts to get all possible subgraphs of G_i.

The good thing is that it suffices to use just a series of k farthest min-cuts computed before for the purpose of removal. This will reduce the number of choices to k^i for any fixed G_i. In our algorithm we establish even a slightly better bound on the number of subgraphs of G_i, needed to consider to construct a k-WTSS(t). In the proof we use k-dimensional vectors to efficiently enumerate all of these subgraphs. After getting those subgraphs we apply a construction similar to that of k-FTRS(t) from [BCR16] to get a bound on in-degree of t. Though our techniques are build on the algorithm of [BCR16], to achieve the claimed bound we need to do much more than just using the algorithm of [BCR16] in a black box fashion. Moreover in the algorithm, without loss of generality we assume the following.

Assumption 44. *The out degree of source vertex s is 1 and the out degree of all other vertices is bounded by 2.*

Explanations for Assumption 44. We are given a directed graph $G=(V,E)$ with an associated weight function $w: E(G)\to\mathbb{R}$ and a source vertex s. We claim that, for any vertex $t\in V(G)$, we can construct a new graph $G'=(V',E')$ with a weight function $w': E(G')\to\mathbb{R}$ and a source vertex s' such that, G' has $O(m)$ vertices and $O(m)$ edges, for any vertex $v\in V(G)$, $dist_{G,w}(s,v)=dist_{G',w'}(s',v)$, the out-degree of the source vertex s' is 1 and the out-degree of every other vertex $v'\in V(G')$ is bounded by 2, given a k-WTSS(t) of G' (say H') we can construct a k-WTSS(t) of G (say H) and in-degree of any vertex in H is same as that of in H'.

Now we describe given graph G how to construct graph G'. If $|Out(s)|>1$ then add a new vertex s' to G and add an edge (s',s) and set $w'(s',s)=0$. Then make this new vertex s' as the source vertex of G'. Otherwise, set $s'=s$. Next, for each $v\in V(G)$, we construct a binary tree T_v as follows: Define r_v to be the root of T_v. Suppose $d(v)$ denotes the out-degree of v in G. Then T_v contains exactly $d(v)$ many leaves, say $l_v^1,\cdots,l_v^{d(v)}$. Let $(v,u_1),\cdots,(v,u_{d(v)})$ are the out

edges of v in G. We delete all of them and in that place we insert the binary tree T_v by adding an edge from vertex v to r_v and adding edges from vertex l_v^i to u_i, for all $i \in [d(v)]$. We define the weight function w' for G' as follows: Set, $w'(v, r_v) = 0$, $w'(l_v^i, u_i) = w(v, u_i)$ for all $i \in [d(v)]$ and for rest of the edges $e \in T_v$ set $w'(e) = 0$.

Subsequently we observe the following properties of G'.

1. The source vertex s' has out-degree 1 and every other vertex of G' has out-degree at most 2 whereas in-degree is same as that of in the graph G.

2. Graph G' has $O(m)$ vertices and $O(m)$ edges.

3. Every edge (v, u_i) of G is represented by a path $P_{v \to u_i} = (v, r_v) \circ$(path from r_v to l_v^i in T_v)$\circ (l_v^i, u_i)$ in G' and $w(v, u_i) = w'(P_{v \to u_i})$. Hence for any vertex $v \in V(G)$, $dist_{G,w}(s, v) = dist_{G',w'}(s', v)$.

Now we show, given a k-WTSS(t), say H' of the graph G' how to construct a k-WTSS(t), say H for the graph G. We build H as follows: For each $u_i \in Out(v)$ of the graph G, we include an edge (v, u_i) in H if and only if the edge (l_v^i, u_i) is present in graph H'. Now the claim is that H is a k-WTSS(t) for the graph G. Let $I : E(G) \to \mathbb{Z}$ be any increment function on graph G such that $\sum_{e \in E(G)} I(e) \leq k$. Now define another increment function $I' : E(G') \to \mathbb{Z}$ for the graph G' as follows: For every edge (v, u_i), set $I'(l_v^i, u_i) = I(v, u_i)$. For all other edges $e \in G'$, set $I'(e) = 0$. Clearly,

$$\sum_{e \in E(G')} I'(e) = \sum_{e \in E(G)} I(e) \leq k.$$

Now from the construction we can observe that for any vertex t in G, $dist_{H,w+I}(s, t) = dist_{H',w'+I'}(s', t)$. Now as H' is a k-WTSS(t) for the graph G', we have that $dist_{H',w'+I'}(s', t) = dist_{G',w'+I'}(s', t)$ and therefore

$$dist_{H,w+I}(s, t) = dist_{H',w'+I'}(s', t) = dist_{G',w'+I'}(s', t) = dist_{G,w+I}(s, t).$$

Hence H is a k-WTSS(t) for the graph G and in-degree of any vertex in H is same as that of in H'. Hence for any vertex $t \in V(G)$, computing k-WTSS(t) of G is same as computing k-WTSS(t) for G' which has $O(m)$ vertices and edges, and out-degree of source vertex is one and out-degree of every other vertex is bounded by two.

5.2 Farthest Min-cut of Shortest Path Subgraph

5.2.1 Computing farthest min-cut of shortest path subgraph

In this section we give an algorithm to find farthest min-cut of the shortest path subgraph of a given graph. We are given a weighted directed graph G, where weight of each edge is defined by a weight function $w : E(G) \to \mathbb{R}$. Let s and t be two vertices in G and $dist_{G,w}(s, t) = d$. We denote the set of all $s - t$ paths of

weight d, under the weight function w, by $\mathcal{P}_d$ and the corresponding underlying subgraph (just the union of all paths in $\mathcal{P}_d$) of G by G_{short}, more specifically, $V(G_{short}) = V(G)$ and $E(G_{short}) = \{e \mid e \in P \text{ for some } P \in \mathcal{P}_d\}$.

Definition 45. *Let s and t be two vertices in any graph G. For any source set $S \subseteq V(G_{short})$, the* farthest min-cut of shortest path subgraph*, denoted as $FSMC(G, S, t)$ is defined by $FMC(G_{short}, S, t)$*

For any (S, t)-cut C of G_{short}, define the partition function by $ShortPartition(G, C) = Partition(G_{short}, C)$.

Next given any graph G and two vertices s and t, we generate the subgraph G_{short} as follows: For each edge $(u, v) \in E(G)$ include (u, v) in a new edge set E' if $dist_{G,w}(s, u) + w(u, v) + dist_{G,w}(v, t) = dist_{G,w}(s, t)$. Then output the graph $G' = (V(G), E')$. It can be easily observed that $G' = G_{short}$ as for some $P \in \mathcal{P}_d$, an edge $e = (u, v) \in P$ if and only if $dist_{G,w}(s, u) + w(u, v) + dist_{G,w}(v, t) = dist_{G,w}(s, t)$.

We can implement the above procedure by first storing the values of $dist_{G,w}(s, u)$ for all $u \in V(G)$ and $dist_{G,w}(v, t)$ for all $v \in V(G)$ which can be done in time $O(mn)$ [Bel58, For56] where $|V(G)| = n$ and $|E(G)| = m$. Hence the time complexity to output the subgraph $G' = G_{short}$ will be $O(mn)$.

Now we can simply apply well known Ford-Fulkerson algorithm [FF] on the subgraph G_{short} to find the $ShortMaxFlow(G, S, t)$ and $FSMC(G, S, t)$. The correctness of $FSMC(G, S, t)$ follows from Lemma 19 applying on the subgraph G_{short}.

5.2.2 Disjoint shortest path lemma

Choose any $r \in \mathbb{N}$ and consider the following: Set $S_1 = \{s\}$ and for $i \in [r]$, define $C_i = FSMC(G, S_i, t)$, $(A_i, B_i) = ShortPartition(G, C_i)$ and $S_{i+1} = (A_i \cup Out(A_i)) \setminus \{t\}$. Let $E' \subseteq E(G)$ be such that $E' = \{(u_1, v_1), \cdots, (u_r, v_r)\}$, where $(u_i, v_i) \in C_i$.

Now let us introduce an auxiliary graph $G' = G + (s, v_1) + \cdots + (s, v_r)$ and set $w(s, v_i) = dist_{G,w}(s, v_i)$ for $i \in [r]$. Suppose f be a max-flow from S_{r+1} to t in the shortest path subgraph of G and $\mathcal{E}(t)$ be the set of incoming edges of t having nonzero flow value assigned by f. Now consider a new graph $G^* = (G' \setminus In\text{-}Edge(t)) + \mathcal{E}(t)$.

Lemma 23. *There will be at least $r + 1$ disjoint paths in G^* each of weight equal to $dist_{G,w}(s, t)$.*

Note that a similar claim was shown in [BCR16]. However, our claim is slightly more general as all the edges of E' may not lie on a single $s - t$ path in G. Moreover we also comment on the weight of the disjoint paths. Both these requirements are crucial for the correctness proof in Section 5.4. Fortunately, the proof in [BCR16] does not rely on the fact that those edges (u_i, v_i)'s are part of a single $s - t$ path. For the sake of completeness we include the proof here.

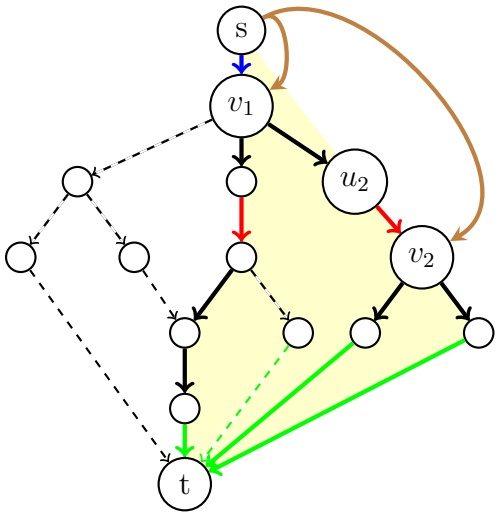


Figure 5.1: Suppose the yellow colored region represents G_{short}. The edges of C_1 and C_2 are colored with blue and red respectively. Brown colored edges are the edges added in the auxiliary graph G' and the edges colored with green constitute the set $\mathcal{E}(t)$. Paths represented by the thick edges are the 3 edge disjoint paths in G' when $r = 2$.

Let us denote the shortest path subgraph (union of all minimum weight $s-t$ paths) of G and G' by G_{short} and G'_{short} respectively. Now let us introduce a series of subgraphs G_i's as follows:

$$G_1 = G_{short},\ G_i = G_{short} + (s, v_1) + \cdots + (s, v_{i-1}) \text{ for } 2 \leq i \leq r+1.$$

Note that $G_{i+1} = G_i + (s, v_i)$. Since $w(s, v_i) = dist_{G,w}(s, v_i)$, the edge (s, v_i) must belong to G'_{short}. This is because (u_i, v_i) already lie on some shortest $s-t$ path, say P in G which is a subgraph of G'. Then $(s, v_i) \circ P[v_i, t]$ will be a minimum weight $s-t$ path. Hence $G_{r+1} = G'_{short}$. Now let us first prove the following.

Lemma 24. *For any two vertices $u, v \in V(G_{short})$, every $u-v$ path in G_{short} has weight $dist_{G,w}(u, v)$.*

Proof. For the sake of contradiction, suppose P be a $u-v$ path in G_{short} that has weight strictly greater than $dist_{G,w}(v, t)$. By the definition of G_{short}, for all $e \in P$, e must lie in some shortest $s-t$ path in G. If all the edges in P lies in the same shortest $s-t$ path then the claim is trivial. Otherwise there must exists two consecutive edges $e_1 = (x, y)$ and $e_2 = (y, z)$ such that they lie in two different shortest $s-t$ paths, say P_1 and P_2 respectively. Observe that $w(P_1[s, y]) = w(P_2[s, y])$, otherwise either of P_1 or P_2 is not a shortest $s-t$ path. Now consider the path $P_3 = P_1[s, y] \circ (y, z) \circ P_2[z, t]$. By construction $w(P_3) = w(P_1[s, y]) + w(y, z) + w(P_2[z, t]) = w(P_2)$ as $w(P_1[s, y]) = w(P_2[s, y])$. So we get another shortest $s-t$ path containing both e_1 and e_2. We can continue this process until we get a shortest $s-t$ path such that all $e \in P$ lie in it and this concludes the proof. □

Claim 25. $C_i = FMC(G_i, S_i, t)$.

Proof. We know that $C_i = FMC(G_{short}, S_i, t)$. Observe that for each $j < i$, we add $Out(A_j) \setminus \{t\}$ to S_{j+1}. Thus for each edge (s, v_j), $j < i$, both the endpoints lie inside the source set S_i. Hence C_i is also same as $FMC(G_i, S_i, t)$. □

Claim 26. *The value of a max-flow from s to t in G'_{short} is at least $r + 1$.*

Proof. We use induction to show that the value of a max-flow from s to t in G_i is at least i for $i \in [r + 1]$. The base case $i = 1$ is trivial because $G_1 = G_{short}$ and there is a $s - t$ path in G_{short}. Now for the induction argument, let us first take $(A, B) = ShortPartition(G, FMC(G_i, s, t))$. Then by applying Lemma 22 on the graph G_i, we say that $B_i \subseteq B$. Hence by Lemma 20, we can argue that the value of a max-flow from s to t in G_{i+1} is at least one more than that in G_i which is at least i by the induction argument, i.e., the value of a max-flow from s to t in G_{i+1} is at least $i + 1$. □

Now consider a new subgraph $G^*_{short} = (G_{r+1} \setminus In\text{-}Edge(t)) + \mathcal{E}(t)$. Note that G^*_{short} is a subgraph of G^*.

Proof of Lemma 23. As f be a max-flow from S_{r+1} to t in G_{short} and both the endpoints of the edge (s, v_i), for any $i \in [r]$ are inside S_{r+1}, the flow f is also a max-flow from S_{r+1} to t in $G_{r+1} = G'_{short}$. Now by applying Lemma 21 on the graph G'_{short}, we can get another max-flow f_{max} from s to t such that $E(f_{max}) \subseteq E(A_{r+1}) \cup E(f)$. As f_{max} terminates t using edges from $\mathcal{E}(t)$, so it is a valid flow also in G^*_{short}. So the value of a max-flow from s to t in G^*_{short} is also at least $r+1$. Now as by Lemma 24 every $s - t$ path in G^*_{short} is of weight equal to $dist_{G,w}(s, t)$ and G^*_{short} is a subgraph of G^*, the lemma follows.

□

5.3 Construction of k-WTSS and Locality Lemma

In this section we reduce the problem of finding k-WTSS to the problem of finding k-WTSS(t) for any fixed vertex $t \in V(G)$. The following lemma, a variant of which also appears in [BCR16], serves our purpose.

Lemma 27 (Locality Lemma)**.** *Let there be an algorithm $\mathcal{A}$ that given a directed weighted graph G and a vertex $t \in V(G)$, generates a subgraph H_t of G such that:*

- *H_t is a k-WTSS(t); and*
- *in-degree of t in H_t is bounded by a constant c_k.*

Then we can compute a k-WTSS of G such that it has at most $c_k \cdot n$ edges.

Proof. Let $(v_1, v_2, \cdots, v_n)$ be any arbitrary ordering of the vertices of G. Given the algorithm $\mathcal{A}$ we design another algorithm $\mathcal{A}'$ which generates k-WTSS in n rounds. The algorithm $\mathcal{A}'$ proceeds as follows: In the first round it starts with the graph $G_0 = G$ which is trivially a k-WTSS and computes another graph G_1 which is a k-WTSS and in G_1 in-degree of v_1 is bounded by c_k. Similarly in the i-th round, a graph G_i is computed such that G_i is a k-WTSS and in-degree of every vertex v_j for $j \leq i$ in G_i is bounded by c_k.

Now we describe round i in details. We start with a graph G_{i-1} which we know is a k-WTSS and in-degree of any vertex v_j for $j < i$ is bounded by c_k. Let H_i be the k-WTSS(v_i) for graph G_{i-1}, computed by algorithm $\mathcal{A}$. We define G_i to be a subgraph of G_{i-1} where the incoming edges of v_i is restricted to that present in H_i. Hence this process assures that in G_i the in-degree of vertices $v_1, \cdots, v_i$ are bounded by c_k.

Next we need to prove that for each $i \in [n]$, G_i is also a k-WTSS and we do this using induction. The base case is true as $G_0 = G$ is trivially a k-WTSS. Next assuming G_{i-1} is a k-WTSS we prove the same for G_i. Now consider any increment function I. Let F be the set of edges for which I has non zero value in G_{i-1} i.e., $F = \{e \in E(G_{i-1}) \mid I(e) > 0\}$. Suppose the new weight function is w' defined by $w'(e) = w(e) + I(e)$. Now consider any vertex t. Let $dist_{G,w'}(s,t) = d'$ and hence by the induction argument, $dist_{G_{i-1},w'}(s,t) = d'$. Suppose the corresponding path is P in G_{i-1}. We need to show there exist an $s-t$ path R of weight d' in G_i such that $w'(R) = d'$. If path P does not contain vertex v_i set $R = P$. Otherwise consider the segments $P[s, v_i]$ and $P[v_i, t]$. We have that $w'(P[s, v_i]) + w'(P[v_i, t]) = d'$. Observe that $w'(P[s, v_i]) = dist_{G_{i-1},w'}(s, v_i)$ otherwise P cannot be a shortest $s - t$ path under w'. Now as G_i differs from G_{i-1} only at the incoming edges of v_i, path segment $P[v_i, t]$ remains intact. As H_i is a k-WTSS(v_i) for G_{i-1}, there exist an $s - v_i$ path, say R' in H_i of weight $dist_{G_{i-1},w'}(s, v_i)$. By the construction G_i contains H_i and hence $R' \circ P[v_i, t]$ is a walk from s to t of weight $w'(R')+w'(P[v_i, t]) = dist_{G_{i-1},w'}(s, v_i)+w'(P[v_i, t]) = d'$ in G_i. Removing loops we get our desired path R of weight at most d'. Now as G_i is a subgraph of G_{i-1}, we can conclude that $w'(R) = d'$. Hence G_i is a k-WTSS. □

5.4 Construction of k-WTSS(t)

In this section we provide an algorithm to compute a k-WTSS(t) for any fixed vertex $t \in V(G)$ where source vertex is s.

5.4.1 Description of the algorithm

Before describing the algorithm we introduce some notations that we will use in our algorithm. Consider any k-dimensional vector $\sigma \in \{-1, 0, 1, \cdots, k\}^k$ such that, for any $i \in [k]$, if $\sigma(i) = -1$ then for all $i' \in [k]$ and $i' > i$, $\sigma(i') = -1$. Otherwise if, $\sigma(i) \neq -1$ then for all $i' < i$, $\sigma(i') \neq -1$. We use these vectors to efficiently enumerate all the subgraphs of G for which we want to calculate farthest min-cuts. Now for any such vector σ and $r \in [k]$, we recursively define the subgraph G_σ, set of source vertices $S_{\sigma,r}$ and edge set $C_{\sigma,r}$ as follows: if $\sigma = (-1, -1, \cdots, -1)$, G_σ is the union of all $s-t$ paths in G, starting with $S_{\sigma,1} = \{s\}$, for any $r \in [k]$ define $C_{\sigma,r} = FSMC(G_\sigma, S_{\sigma,r}, t)$, $S_{\sigma,r+1} = (A \cup Out(A)) \setminus \{t\}$ where $(A, B) = ShortPartition(G_\sigma, C_{\sigma,r})$. For $\sigma \neq (-1, \cdots, -1)$, G_σ is the union of all $s - t$ paths in $G_{\sigma'} \setminus C_{\sigma',\sigma(i)+1}$, where $\sigma' \in \{-1, 0, 1, \cdots, k\}^k$ such that,

$$\forall i' \in [k], \sigma'(i') = \begin{cases} \sigma(i') & \text{if } i' < i \\ -1 & \text{otherwise} \end{cases}$$

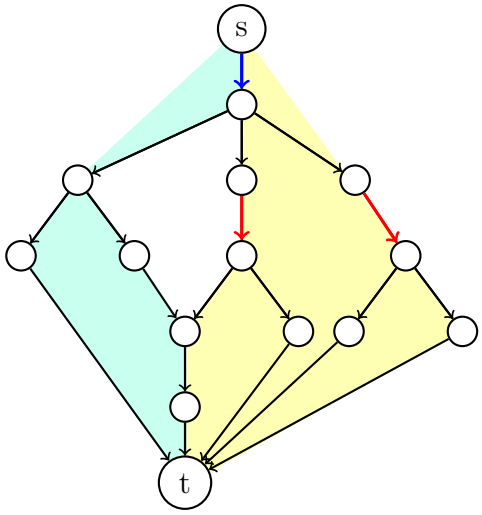


Figure 5.2: Region shaded with green color represents G_σ for $\sigma = (1, -1, \cdots, -1)$ whereas yellow colored region is the shortest path subgraph of G. The edges of $C_{(-1,\cdots,-1),1}$ and $C_{(-1,\cdots,-1),2}$ are colored with blue and red respectively. G_σ is obtained by removing red colored edges.

and, $i = \max\{i' \mid \sigma(i') \neq -1\}$. Now starting with $S_{\sigma,1} = \{s\}$, if there exists a $s - t$ path of weight $d + i$ then for any $r \in [k]$ define $C_{\sigma,r} = FSMC(G_\sigma, S_{\sigma,r}, t)$, $S_{\sigma,r+1} = (A \cup Out(A)) \setminus \{t\}$ where $(A, B) = ShortPartition(G_\sigma, C_{\sigma,r})$; else set $C_{\sigma,r} = \phi$. We refer the reader to Figure 5.2 for the better understanding about the graph G_σ.

We are given a weighted directed graph G with weight function w and a source vertex s and a destination vertex t. The weight of each edge of G is defined by the weight function $w : E(G) \to \mathbb{Z}$. Overall our algorithm (Algorithm 4) performs the following tasks: For different values of $\sigma \in \{-1, 0, \cdots, k\}^k$ it computes the sets $C_{\sigma,i}$ and $S_{\sigma,i}$ for $i \in [k]$. Then for each such σ, it computes max-flow in the shortest path subgraph of G_σ by considering $S_{\sigma,k}$ as source and add the edges incident on t with non-zero flow to a set $\mathcal{E}(t)$. At the end, our algorithm returns the subgraph $H_t = (G \setminus In\text{-}Edge(t)) + \mathcal{E}(t)$.

Our algorithm performs the above tasks in the recursive fashion. Starting with $\sigma = (-1, \cdots, -1)$, it first considers the shortest path subgraph of $G_\sigma = G$ and performs k iterations on it. At each iteration it computes the farthest min-cut $C_{\sigma,i}$ by considering $S_{\sigma,i}$ as source and t as sink starting with $S_{\sigma,1} = \{s\}$. Then it updates the graph by removing the edges present in $C_{\sigma,i}$ and passes this new graph in the next recursive call. Before the recursive call it also updates the σ by incrementing the value of $\sigma(j)$ by one and passes the updated value of σ to the recursive call. Here j is a parameter which denotes that the smallest coordinate of σ that has value -1. Initially j was set to 1 and before the next recursive call we increment its value by one. At the end of each iteration our algorithm updates the source set to $S_{\sigma,i+1}$ by including end points of all the edges present in the cut $C_{\sigma,i}$ in the set $S_{\sigma,i}$. At the end of k iterations, the algorithm computes max-flow in the shortest path subgraph of G_σ by considering $S_{\sigma,k}$ as source and add the edges incident on t with non-zero flow to a set $\mathcal{E}(t)$.

Algorithm 4 Algorithm for computing k-WTSS(t)

Input: A graph G with weight function w and two vertices s and t
Output: A subgraph H_t

Initialization: For all $\sigma \in \{-1, 0, \cdots, k\}^k$ and $r \in [k]$, set $C_{\sigma,r}$ to be ϕ;
Set $\sigma_{curr} = (-1, \cdots, -1)$;
$RecursiveWTSS(G, \sigma_{curr}, 1)$;
Return $H_t = (G \setminus In\text{-}Edge(t)) + \mathcal{E}(t)$;

Procedure 5 $RecursiveWTSS(G_{curr}, \sigma, j)$

if there exists an $i \in [j-1]$ such that $\sigma(i) \geq k - j + i - 1$ **then**
  return ;
end if
Define σ_{curr} by setting $\sigma_{curr}(j) = 0$ and $\sigma_{curr}(i) = \sigma(i)$ for all $i \neq j$;
if $dist_{G_{curr},w}(s, t) = d + j - 1$ **then**
  $S_1 \leftarrow \{s\}$;
  for $i = 1, \cdots, k$ **do**
    $C_{\sigma,i} \leftarrow FSMC(G_{curr}, S_i, t)$;
    $RecursiveWTSS((G_{curr} \setminus C_{\sigma,i}), \sigma_{curr}, j + 1)$;
    $\sigma_{curr}(j) \leftarrow \sigma_{curr}(j) + 1$;
    $(A_i, B_i) \leftarrow ShortPartition(G_{curr}, C_{\sigma,i})$;
    $S_{i+1} \leftarrow (A_i \cup Out(A_i)) \setminus \{t\}$;
  end for
  $f \leftarrow ShortMaxFlow(G_{curr}, S_{\sigma,k+1}, t)$;
  Add incoming edges of t present in $E(f)$ in $\mathcal{E}(t)$;
else
  Define σ_{curr} by setting $\sigma_{curr}(j) = 0$ and $\sigma_{curr}(i) = \sigma(i)$ for all $i \neq j$;
  $RecursiveWTSS(G_{curr}, \sigma_{curr}, j + 1)$;
end if

5.4.2 Analysis

Correctness:
In this section we first show that, for any vertex $t \in V(G)$, the output graph H_t is indeed a $k - WTSS(t)$. We start with the following simple observation.

Observation 46. *For any $\sigma \in \{-1, 0, \cdots, k\}^k$, any $s-t$ path in G_σ must have weight at least $d + i - 1$ where $i = \min\{i' \mid \sigma(i') = -1\}$.*

Proof. Now for any σ, let us consider a sequence of vectors $\alpha_1, \cdots, \alpha_i \in \{-1, 0, \cdots, k\}^k$ where $i = \min\{i' \mid \sigma(i') = -1\}$ as follows: for any $1 \leq j \leq i$,

$$\alpha_j(i') = \begin{cases} \sigma(i') & \text{if } i' < j \\ -1 & \text{otherwise} \end{cases}$$

Note that $\alpha_i = \sigma$. Now we use induction on j to show that any $s-t$ path in G_{α_j} must have weight at least $d + j - 1$ and that will prove our claim.

As a base case when $j = 1$, as $\alpha_1 = (-1, \cdots, -1)$, $G_{\alpha_j} = G$ and hence the claim is trivially true. Now suppose the claim is true for all $j \in [i-1]$ and we need to prove it for $j+1$. By definition, $G_{\alpha_{j+1}}$ is a subgraph of G_{α_j}. By induction hypothesis all the $s-t$ paths in G_{α_j} have weight at least $d + j - 1$. If there is no $s-t$ path of weight $d + j - 1$ in G_{α_j} then we are done because of our integer valued weight function. Otherwise any such path of weight $d + j - 1$ must pass through the cut set $C_{\alpha_j, \sigma(j)+1}$. Now by definition, $G_{\alpha_{j+1}}$ is build by removing the edge set $C_{\alpha_j, \sigma(j)+1}$ from the graph G_{α_j}. Hence there will be no $s-t$ path of weight $d + j - 1$ in $G_{\alpha_{j+1}}$. Since our weight function is integer valued, the claim follows. □

Note that the above observation is true only because we consider the range of our weight function w to be $\mathbb{Z}$. Otherwise it will trivially be false.

Now consider any increment function $I : E(G) \to \mathbb{N}$ such that $\sum_{e \in E(G)} I(e) \leq k$ and then denote the set of edges with non-zero value of the function I by F, i.e., $F = \{e \in E(G) | I(e) > 0\}$. So clearly $|F| \leq k$. Now suppose $dist_{G,w'}(s,t) = d' = d + j$ for some $0 \leq j \leq k$ where $w'(e) = w(e) + I(e)$. Thus we need to show that there also exists an $s-t$ path of weight d' in the subgraph H_t under the new weight function w'.

Suppose P be an $s-t$ path in G such that $w'(P) = d' = d + j$. For simplicity let us assume the following.

Assumption 47. *For all $e \in P$, $I(e) = 0$.*

In other words we are assuming that $w'(P) = w(P)$. We will now argue to justify our assumption.

Explanation for Assumption 47. Suppose P be one of the shortest paths from s to t in G under the new weight w', i.e., $w'(P) = d'$. Then consider the following set $S = \{e \in P | I(e) > 0\}$ and suppose $\sum_{e \in S} I(e) = k' \leq k$. Now define the following new weight function:

$$w''(e) = \begin{cases} w(e) & \text{if } e \in P \\ w'(e) & \text{otherwise} \end{cases}$$

Now $w''(P) = d' - k'$. Then use the argument same as before to show that there exists a path, say R in H_t of weight at most $d' - k'$ under this new weight function w''. Clearly, $w'(R) \leq w''(R) + k' = d'$.

Next we claim the following.

Lemma 28. *One of the following three cases must satisfy.*

1. *There exists a σ such that P belongs to the subgraph G_σ where $\sigma(j) = -1$ and for some positive integer r, the last edge of P belongs to the edge set $C_{\sigma,r}$.*

2. *There exists a σ such that P belongs to the subgraph G_σ where $\sigma(j+1) = -1$, $\sigma(j) \neq -1$ and there is no $i \in [j-1]$ such that $\sigma(i) \geq k - j + i - 1$.*

3. *There exists a σ such that P belongs to the subgraph G_σ where if $i = \min\{i' \mid \sigma(i') = -1\}$ then $i \leq j$ and for any $i' \leq i$, $\sigma(i') < k - j + i' - 1$ and P passes through all the cut sets $C_{\sigma,1}, \cdots, C_{\sigma,k-j+i-1}$.*

Proof. Here we describe a procedure to find desired σ for the path P. Let us initialize $\sigma = (-1, \cdots, -1)$. So $G_\sigma = G$ and thus trivially P belongs to G_σ. Suppose P pass through edges of the cut sets $C_{\sigma,1}, \cdots, C_{\sigma,r_1}$, but does not pass through any edge of C_{σ,r_1+1}. Note that r_1 will be equal to 0 if P does not pass through any of the cut sets. Update σ by setting $\sigma(1) = r_1$. By the definition of G_σ, P belongs to it. Now suppose P passes through edges of the cut sets $C_{\sigma,1}, \cdots, C_{\sigma,r_2}$, but does not pass through any edge of C_{σ,r_2+1}. Then update σ by setting $\sigma(2) = r_2$. Now proceed in this way until $\sigma(j)$ is set or we reach at a point where for some $i \in [j-1]$, P passes through all the cut sets $C_{\sigma,1}, \cdots, C_{\sigma,k-j+i-1}$. This process may stop prematurely if P reaches t before satisfying either of above two conditions, but in that case the last edge, say (v,t) of P must belong to some cut set $C_{\sigma,r}$. Hence we will be in the first case and this completes the proof. □

Now let us call the path P is of type-1, type-2 and type-3 respectively depending on which of the above three cases it satisfies.

Type-1: This case is the simplest among the three.

Lemma 29. *If P is a type-1 path then P is contained in the subgraph H_t.*

Proof. Suppose (v,t) is the last edge of the path P. Now as $(v,t) \in C_{\sigma,r}$ for some σ and r, $(v,t) \in \mathcal{E}(t)$. Thus by the construction of the subgraph H_t, the edge (v,t) belongs to H_t. Also by the construction of the subgraph H_t, for all the vertices $u \neq t$, $In\text{-}Edge(u)$ belong to H_t. Hence P must lie completely inside H_t. □

Type-2: As path P belongs to the subgraph G_σ where $\sigma(j) \neq -1$ and $\sigma(j+1) = -1$, by Observation 46, $w(P) \geq d + j$. However by our Assumption 47, $w(P) = d + j$ and so it must pass through an edge $(u_r, v_r) \in C_{\sigma,r}$ for all $r \in [k]$. Now consider an auxiliary graph $G'_\sigma = G_\sigma + (s, v_1) + \cdots + (s, v_k)$ and extend the weight function w as $w(s, v_r) = w(P[s, v_r])$. Then define another graph $G^*_\sigma = (G'_\sigma \setminus In\text{-}Edge(t)) + \mathcal{E}(t)$. By Lemma 23 we can claim the following.

Corollary 48. *There will be $k+1$ edge disjoint paths in G^*_σ each of weight $w(P)$ under weight function w.*

Now we use the above corollary to conclude the following.

Lemma 30. *If P is a type-2 path then there exists an $s-t$ path of weight d' in the subgraph H_t under the new weight function w'.*

Proof. By Corollary 48, we get $k+1$ edge disjoint paths $P_1, \cdots, P_{k+1}$ each of weight $w(P) = d + j$. Since $|F| \leq k$ where $F = \{e \in E(G) | I(e) > 0\}$, at least one of the $k+1$ many edge disjoint paths, say P_1 must survive in $G^*_\sigma \setminus F$. If P_1 also belongs to the subgraph H_t then we are done. Otherwise P_1 must take some of the (s, v_r)'s as the first edge and the remaining portion $P_1[v_r, t]$ lies inside H_t. Now consider the following path $R = P[s, v_r] \circ P_1[v_r, t]$. By the construction of G'_σ, $w'(R) = w'(P_1) = w(P)$ and this completes the proof. □

Type-3: Suppose P is a type-3 path and thus belongs to G_σ for some σ where if $i = \min\{i' \mid \sigma(i') = -1\}$ then $i \leq j$ and for any $i' \leq i$, $\sigma(i') < k - j + i' - 1$ and P passes through all the cut sets $C_{\sigma,1}, \cdots, C_{\sigma,k-j+i-1}$. P passes through an edge $(u_r, v_r) \in C_{\sigma,r}$ for all $r \in [k - j + i - 1]$. Now if there exists a positive integer $r \in [k - j + i - 1]$ such that $w(P[v_{r-1}, u_r]) > dist_{G_\sigma,w}(v_{r-1}, u_r)$, replace the portions of path $P[v_{r-1}, u_r]$ by the $v_{r-1} - u_r$ path of weight $dist_{G_\sigma,w}(v_{r-1}r, u_r)$. We do this until there is no such r and after that we call this new path as P'.

Now consider an auxiliary graph $G'_\sigma = G_\sigma + (s, v_1) + \cdots + (s, v_{k-j+i-1})$ and extend the weight function w as $w(s, v_r) = w(P[s, v_r])$. Next define another graph $G^*_\sigma = (G'_\sigma \setminus In\text{-}Edge(t)) + \mathcal{E}(t)$. Now we use a slightly different argument than that used previously.

Let us now analyze by considering the following two cases separately.

Case 1: $[w(P[s, u_1]) = dist_{G_\sigma,w}(s, u_1)]$

Claim 31. *There will be at least $k-j+i$ edge disjoint paths in G^*_σ each of weight at most $d + j$ under weight function w. Moreover, at least one path among them will be of weight $d + i - 1$.*

Proof. Let us consider a new weight function w_1 as follows:

$$w_1(e) = \begin{cases} dist_{G_\sigma,w}(s, v_r) & \text{if } e = (s, v_r) \text{ for some } r \in [k - j + i - 1] \\ w(e) & \text{otherwise} \end{cases}$$

Then by Lemma 23, G^*_σ has $(k - j + i)$ edge disjoint paths, say $P_1, \cdots, P_{k-j+i}$ each of weight $d + i - 1$ under the new weight function w_1 where the weight bound follows from Observation 46. Now since by Assumption 44 the out degree of s is 1, so $|C_{\sigma,1}| = 1$. As both P and P' pass through the edge (u_1, v_1) and $w(P[s, u_1]) = dist_{G_\sigma,w}(s, u_1)$, so from the construction of P' it can be observed that $w_1(s, v_1) = w(s, v_1)$. Now consider the path that takes (s, v_1) as the first edge and say it is P_1. Then

$$w(P_1) = w(s, v_1) + w(P_1[v_1, t]) = w_1(s, v_1) + w_1(P_1[v_1, t]) = w_1(P_1) = d + i - 1.$$

For any other path, say P_2, clearly $w(P_2) \leq w_1(P_2) + (j - i + 1) = d + j$ because for any $2 \leq r \leq k - j + i - 1$, $w(s, v_r) - w_1(s, v_r) \leq j - i + 1$. □

So we get $k-j+i$ edge disjoint paths $P_1, \cdots, P_{k-j+i}$ each of weight at most $d+j$ and suppose P_1 has weight $d+i-1$. Let us also extend the weight function w' by setting $w'(s, v_r) = w(s, v_r)$ and extend I by setting $I(s, v_r) = 0$. If any one of these $k-j+i$ edge disjoint paths, say Q satisfies that $w'(Q) \leq d+j$, then we are done. This is because in that case either Q lies inside H_t which makes Q to be our desired path or for some $r \in [k-j+i-1]$, Q must take (s, v_r)'s as the first edge and the remaining portion $Q[v_r, t]$ lies inside H_t. In the second case, we consider the path $R = P[s, v_r] \circ Q[v_r, t]$. Note that $w'(R) = w'(Q) \leq d+j$.

Now we argue that there must exists one path among $k-j+i$ edge disjoint paths such that it will have weight at most $d+j$ under the weight function w'. Otherwise for all $r \in [k-j+i]$, $w'(P_r) \geq d+j+1$. Hence $I(P_1) \geq j-i+2$ and $I(P_r) \geq 1$ for all $2 \leq r \leq k-j+i$. Thus

$$\sum_{e \in E(G^*_\sigma)} I(e) \geq (j-i+2) + (k-j+i-1) \geq (k+1).$$

However as $I(s, v_1) = I(s, v_2) = \cdots = I(s, v_{\sigma(i)}) = 0$,

$$\sum_{e \in E(G^*_\sigma)} I(e) \leq \sum_{e \in E(G)} I(e) \leq k$$

which leads to a contradiction.

Case 2: $[w(P[s, u_1]) > dist_{G_\sigma, w}(s, u_1)]$

In this case also by the argument used in the first part of the proof of Claim 31, we can claim the following.

Claim 32. *There will be at least $k-j+i$ edge disjoint paths in G^*_σ each of weight at most $d+j$ under weight function w.*

Note that on the contrary to Claim 31, now we do not have the extra guarantee that at least of the edge disjoint paths must have weight $d+i-1$. Now just like the previous case, we only need to argue that there must exists one path among $k-j+i$ edge disjoint paths, say $P_1, \cdots, P_{k-j+i}$ such that it will have weight at most $d+j$ under the weight function w' and we will be done.

Now suppose $w(P[s, u_1]) = dist_{G_\sigma, w}(s, u_1) + l$, for $l > 0$. Consider the path that takes (s, v_1) as the first edge and say it is P_1. Then by the argument used in the proof of Claim 31, one can show that

$$w(P_1) = w(s, v_1) + w(P_1[v_1, t]) = (dist_{G_\sigma, w}(s, v_1) + l) + w_1(P_1[v_1, t]) = d+i+l-1.$$

Let Q be a shortest $s-v$ path in G_σ under weight w. Now since $w(P[s, u_1]) > dist_{G_\sigma, w}(s, u_1)$ and P is a shortest $s-t$ path under the weight w' (recall that $w' = w + I$), $I(Q) \geq l+1$. Moreover,

$$\sum_{e \in Q \text{ and } e \notin P} I(e) \geq l+1.$$

Now if for all $r \in [k-j+i]$, $w'(P_r) \geq d+j+1$, it must satisfy that $I(P_1) \geq j-i-l+2$ and $I(P_r) \geq 1$ for all $2 \leq r \leq k-j+i$.

$$\sum_{e \in E(G^*_\sigma)} I(e) \geq (l+1) + (j-i-l+2) + (k-j+i-2) \geq (k+1).$$

However as $I(s, v_1) = I(s, v_2) = \cdots = I(s, v_{\sigma(i)}) = 0$,

$$\sum_{e \in E(G^*_\sigma)} I(e) \leq \sum_{e \in E(G)} I(e) \leq k$$

which again leads to a contradiction.

Now from the above we can conclude the following.

Lemma 33. *If P is a type-3 path then there exists an $s-t$ path of weight $w(P) = d'$ in the subgraph H_t under the new weight function w'.*

Bounding the size of $\mathcal{E}(t)$:
In this part we establish an upper bound on the size of the set of edges $\mathcal{E}(t)$. We define $C_{\sigma,k+1} = FSMC(G_\sigma, S_{\sigma,k+1}, t)$ for any $\sigma \in \{-1, 0, \cdots, k\}^k$. Now as $FSMC(G_\sigma, S_{\sigma,i+1}, t) = FMC(G^{short}_\sigma, S_{\sigma,i+1}, t)$ for any $i \in [k]$, where G^{short}_σ is the shortest path subgraph of G_σ, so we can restate Lemma 6.6 from [BCR16] in the following form.

Lemma 34. *For any $i \in [k]$, $|C_{\sigma,i+1}| \leq 2 \times |C_{\sigma,i}|$.*

Reader may note that the proof of the above lemma in [BCR16] crucially relies on the Assumption 44. Next, we prove the following.

Lemma 35. $|\mathcal{E}(t)| \leq e(k-1)!2^k$.

Proof. In our algorithm for each $\sigma \in \{-1, 0, \cdots, k\}^k$ we compute the cut sets $C_{\sigma,1}, \cdots, C_{\sigma,k}$ and add $|C_{\sigma,k+1}|$ many edges in the set $\mathcal{E}(t)$ if for all $i' \leq i$, $\sigma(i') < k - i + i' - 1$ where $i = \min\{j \mid \sigma(j) = -1\}$; otherwise we do not compute anything. So the total number of σ for which we add edges in $\mathcal{E}(t)$ is bounded by $1+(k-1)+(k-1)(k-2)+\cdots+(k-1)! = (k-1)![1/0!+1/1!+\cdots+1/(k-1)!] \leq e \cdot (k-1)!$

Now by applying Lemma 34, we get that for each such σ, $|C_{\sigma,k+1}| \leq 2^k$ and this proves the claimed bound. □

Time Complexity:
Now we analyze the running time of our algorithm to find k-WTSS(t) for some given vertex $t \in V(G)$. We first preprocess the input graph to generate a new graph in a way so that the new graph satisfies Assumption 44. This takes $O(m)$ time. Next we apply Algorithm 4 on this new graph which has $O(m)$ many vertices and edges. By the argument in the proof of Lemma 35 we see that our algorithm computes k farthest min-cuts on shortest path subgraphs of G_σ for $e(k-1)!$ many different σ's. Now from the discussion in Section 5.2.1, assuming we have G_σ explicitly, to generate each such shortest path subgraph on this new transformed graph we need $O(m^2)$ time and then to compute k farthest min-cuts takes total $O(\sum_{i=1}^{k} |C_{\sigma,i}| \times m) = O(2^k m)$ time (see [FF]). Finally, one can get k-WTSS(t) of the original graph from that of the transformed graph in $O(m)$ time.

So overall time needed to compute k-WTSS(t) of any given graph with n vertices and m edges is $O((k-1)!2^k m^2) = O((k)^k m^2)$ (by Stirling's approximation). Now since by the Locality Lemma (Lemma 27), finding k-WTSS requires n rounds where in each round we find k-WTSS(v) for some $v \in V(G)$, computing k-WTSS takes total $O((k)^k m^2 n)$ time.

5.5 Lower Bound on the Size of k-WTSS

In this section we give explicit construction of a graph that will establish a lower bound on the size of a k-WTSS. Formally we prove the following.

Theorem 49. *For any positive integer $k \geq 2$, there exists an positive integer n' such that for all $n > n'$, there exists a directed graph G with n vertices and a weight function $w : E(G) \to \mathbb{Z}$, such that its k-WTSS must contain $c \cdot 2^k n$ many edges for some constant $c \geq 5/4$.*

Proof. Consider the largest ℓ such that $\sum_{j=2}^{\ell} j \leq k$. Next, take ℓ full binary trees $T_1, \ldots, T_\ell$ such that for each $\ell \leq i \leq l$, T_i has height $h_i = k - \sum_{j=2}^{i} j$ with root r_i. Let L_i be the set of leaves of tree T_i and thus $|L_i| = 2^{h_i}$. Next consider $L = \cup_i L_i$ and another set X containing n vertices. Finally define a graph G with $V(G) = \{s\} \cup (\cup_i V(T_i)) \cup X$ and $E(G) = \{(s, r_i) | 1 \leq i \leq \ell\} \cup \{(u, v) | u \in L, v \in X\} \cup (\cup_i E(T_i))$. Now let us consider the following weight function $w : E(G) \to \mathbb{N}$,

$$w(e) = \begin{cases} \sum_{j=2}^{i} j + i & \text{if } e = (s, r_i) \\ 1 & \text{otherwise} \end{cases}$$

Clearly, $|E(G)| = l + \sum_{i=1}^{\ell}(2|L_i| - 1) + |L| \times |X| = c \cdot 2^k n$ for some constant $c \geq 5/4$.

It only remains to show that any k-WTSS of G must contain all the edges of G. It can be easily observed that any edge of the form (s, r_i) and all the edges present in tree T_i, where $1 \leq i \leq \ell$ must be contained in k-WTSS of G. Now consider an edge $(t, t') \in L \times X$. Suppose $t \in L_i$. Now let P be the path from s to t. Naturally, P consists of edge (s, r_i) and a path from r_i to t that is contained in tree T_i. consider the set $S = \{(u, v) | (u, v) \in T_i,\ u \in P \text{ but } v \notin P\}$. Let us now consider the following increment function $I : E(G) \to \{0, \cdots, k\}$,

$$I(e) = \begin{cases} i + 1 - j & \text{if } e = (s, r_j) \text{ and } 1 \leq j < i \\ 1 & e \in S \\ 0 & \text{otherwise} \end{cases}$$

Clearly, $\sum_{e \in E(G)} I(e) \leq k$ due to the choice of ℓ. Also $P \circ (t, t')$ will be the only shortest path from s to t' in G under the new weight function $w'(e) = w(e) + I(e)$, $\forall_{e \in E(G)}$ because all the $s - t'$ paths whose first edge is (s, r_j) for $j < i$ and all the $s - t'$ paths except P, whose first edge is (s, r_i) will now have weight $k + i + 1$. Therefore any k-WTSS of G must contain the edge (t, t'). This completes the proof. □

5.6 Conclusion and Bibliographical Notes

In this work we initiate the study of single source shortest path problem in a model where weight of any edge can be increased. This model is motivated from congestion in any network and is simpler than the edge fault model. For this model we provide an efficient algorithm to compute a sparse subgraph that preserves the distances from any designated source vertex and is also resilient under bounded weight increment. When the weight increment is bounded by k then the subgraph computed by our algorithm will have size at most $O(k^k n)$. We also show a lower bound of $\frac{5}{4}2^k n$ on the size of such a subgraph.

In another variation, instead of edges, it is also possible to take weights on the vertices and performing increment over them. However, one can directly apply our result by splitting each vertex v into two vertices v_i and v_o where all the incoming and outgoing edges of v are respectively directed into v_i and directed out of v_o, and then considering an edge (v_i, v_o) with the weight equal to that on the vertex v.

Open Problems. Our upper and lower bound results show that our construction is tight up to some constant factor as long as k is bounded by some constant. Consequently, it will be interesting to further study this problem to close the gap between the upper and lower bound. Another open problem can be to improve the runtime of the construction.

Part II

Efficient Construction of Quasi-Gray Codes

6. Introduction

One of the fundamental question in the domain of combinatorics and algorithm design concerns with the efficient enumeration of all the elements of a specific combinatorial class. The aim is to generate a list such that an element in the list can be obtained by a small amount of change to the element that precedes it. One of the classic examples is, the binary *Gray code* introduced by Gray [Gra53], initially used in pulse code communication. Gray codes have found numerous applications in a wide variety of areas, such as information storage and retrieval [CCC92], processor allocation [CS90], computing the permanent [NW78], circuit testing [RC81], data compression [Ric86], graphics and image processing [ASD90], signal encoding [Lud81], modulation schemes for flash memories [JMSB09, GLSB11, YS12] and many more. Interested reader may refer to an excellent survey by Savage [Sav97] for a comprehensive treatment on this subject.

The original idea of a Gray code was to list down all binary strings of length n, i.e, all the elements of $\mathbb{Z}_2^n$, such that any two successive strings differ by exactly one bit. Later, the domain was generalised from $\mathbb{Z}_2^n$ to $\mathbb{Z}_m^n$. Formally, a *code* over domain $\mathbb{Z}_m^n$ is a cyclic sequence of all distinct elements from $\mathbb{Z}_m^n$. As another generalization researchers studied (e.g. [Fre78, RM10, BCJ$^+$10, BGPS14]) codes where instead of enumerating all the elements of the sequence, the aim is to list down ℓ distinct of them where each two consecutive elements differ in one position. We refer to such codes as Gray codes of length ℓ [Fre78]. A code is called *space-optimal*, if it enumerates all the elements of the domain. It is often required that the last and the first strings appearing in the list also differ in one position. Such codes are called *cyclic Gray codes*. In another variation, we study codes where any two consecutive codewords in the sequence differs at most in $c > 0$ (where c is a constant) positions. Such codes are called *quasi-Gray-codes* [BCJ$^+$10].

In our work [CDKS18], We study the problem of constructing quasi-Gray codes over $\mathbb{Z}_m^n$ in the cell probe model [Yao81], where each cell stores an element from $\mathbb{Z}_m$. The efficiency of a construction is measured using three parameters. First, we want the length of a quasi-Gray code to be as large as possible. Ideally, we want space-optimal codes. Second, we want to minimize the number of coordinates of the input string the algorithm reads in order to generate the next (or, previous) string in the code. Finally, we also want the number of cells written in order to generate the successor (or, predecessor) string to be as small as possible. Since our focus is on quasi-Gray codes, the number of writes will always be bounded by a universal constant. We are interested in the worst-case behavior and we use *decision assignment trees* (DAT) of Fredman [Fre78] to measure these complexities.

Related Works: The construction of Gray codes is central to the design of algorithms for many combinatorial problems [Sav97]. Frank Gray [Gra53] first came up with a construction of Gray code over binary strings of length n, where to generate the successor or predecessor strings one needs to read n bits in the worst-case. The type of code described in [Gra53] is known as *binary reflected*

Reference	Value of m	length	Worst-case cell read	Worst-case cell write
[Gra53]	2	2^n	n	1
[Fre78]	2	$2^{\Theta(n)}$	$O(\log n)$	$O(1)$
[FMS97]	2	$\Theta(2^n/n)$	$\log n + 1$	$\log n + 1$
[RM10]	2	2^{n-1}	$\log n + 4$	4
[BCJ+10]	2	$2^n - O(2^n/n^t)$	$O(t \log n)$	3
[BGPS14]	2	$2^n - 2^{n-t}$	$\log n + t + 3$	2
[BGPS14]	2	$2^n - 2^{n-t}$	$\log n + t + 2$	3
[Fre16]	2	2^n	$n - 1$	1
Theorem 51	2	$2^n - O(n)$	$\log n + 4$	2
[Coh63]	any m	m^n	n	1
Theorem 50	any odd m	m^n	$4 \log_m n + 3$	2

Table 6.1: Taxonomy of construction of Gray/quasi-Gray codes over $\mathbb{Z}_m^n$

Gray code. Later Bose *et al.* [BCJ+10] provided a different type of Gray code construction, namely *recursive partition Gray code* which attains $O(\log n)$ average case read complexity while having the same worst-case read requirements. The read complexity we referred here is in the bit-probe model. It is easy to observe that any space-optimal binary Gray code must read $\log n + 1$ bits in the worst-case [Fre78, NRR13, Fre16]. Recently, this lower bound was improved to $n/2$ in [Ras17]. An upper bound of even $n - 1$ was not known until very recently [Fre16]. This is also the best known so far.

Fredman [Fre78] extended the definition of Gray codes by considering codes that may not enumerate all the strings (though presented in a slightly different way in [Fre78]) and also introduced the notion of *decision assignment tree* (DAT) to study the complexity of any code in the bit-probe model. He provided a construction that generates a Gray code of length $2^{c \cdot n}$ for some constant $c < 1$ while reducing the worst-case bit-read to $O(\log n)$. Using the idea of Lucal's modified reflected binary code [Luc59], Munro and Rahman [RM10] got a code of length 2^{n-1} with worst-case read complexity only $4 + \log n$. However in their code two successive strings differ by 4 coordinates in the worst-case, instead of just one and we refer to such codes as quasi-Gray codes following the nomenclature used in [BCJ+10]. Brodal et al. [BGPS14] extended the results of [RM10] by constructing a quasi-Gray code of length $2^n - 2^{n-t}$ for arbitrary $1 \leq t \leq n - \log n - 1$, that has $t + 3 + \log n$ bits ($t + 2 + \log n$ bits) worst-case read complexity and any two successive strings in the code differ by at most 2 bits (3 bits). In contrast to the Gray codes over binary alphabets, Gray codes over non-binary alphabets received much less attention. The construction of binary reflected Gray code was generalized to the alphabet $\mathbb{Z}_m$ for any $m \in \mathbb{N}$ in [Flo56, Coh63, JWW80, Ric86, Knu11, HR16]. However, each of those constructions reads n coordinates in the worst-case to generate the next element. As mentioned before, we measure the read complexity in the well studied cell probe model [Yao81] where we assume that each cell stores an element of $\mathbb{Z}_m$. The argument of Fredman in [Fre78] implies a lower bound of $\Omega(\log_m n)$ on the read complexity of quasi-Gray code on $\mathbb{Z}_m^n$. To the best of our knowledge, for non-binary alphabets, there is nothing known similar to the results of Munro and Rahman or Brodal et al. [RM10, BGPS14]. We summarize the previous results along with ours in Table 6.1.

Despite of an extensive research conducted so far, no construction is known for a quasi-Gray code of length $2^n - 2^{\epsilon n}$, (for some constant $\epsilon < 1$), with sub-

linear read complexity. Rather recently Raskin [Ras17] showed that, any space-optimal quasi-Gray code over the domain $\mathbb{Z}_2^n$ (which can be easily generalized for any domain $\mathbb{Z}_m^n$, where m is even) must have the read complexity $n/2$. The immediate question that pops up naturally is: Is Raskin's lower bound true for quasi-Gray code over odd sized alphabet as well?

Our Contribution: In this thesis, we refute this by giving the construction of a space optimal quasi-Gray code over $\mathbb{Z}_m^n$ (where m is odd) with read complexity $4\log_m n$ and write complexity 2. As a consequence we get an exponential separation between the read complexity of space-optimal quasi-Gray code over $\mathbb{Z}_2^n$ and that over $\mathbb{Z}_3^n$. Formally we show the following.

Theorem 50. *Let $m \in \mathbb{N}$ be odd and $n \in \mathbb{N}$ be such that $n \geq 15$. Then, there is a space-optimal quasi-Gray code C over $\mathbb{Z}_m^n$ for which, the two functions $next(C, w)$ and $prev(C, w)$ can be implemented by inspecting at most $4\log_m n$ cells while writing only 2 cells.*

In the statement of the above theorem, $next(C, w)$ denotes the element appearing after w in the cyclic sequence of the code C, and analogously, $prev(C, w)$ denotes the preceding element. Using the argument as in [Fre78, NRR13] it is easy to see a lower bound of $\Omega(\log_m n)$ on the read complexity when the domain is $\mathbb{Z}_m^n$. Hence our result is optimal up to some small constant factor.

Raskin showed $\Omega(n)$ lower bound on the read complexity of space-optimal binary quasi-Gray codes. The existence of binary quasi-Gray codes with sub-linear read complexity of length $2^n - 2^{\epsilon n}$, for some constant $\epsilon < 1$, was open. Using a different technique than that used in the proof of Theorem 50 we get a quasi-Gray code over the binary alphabet which enumerates all but $O(n)$ many strings.

Theorem 51. *Let $n \geq 15$ be any natural number. Then, there is a quasi-Gray code C of length at least $2^n - 20n$ over $\mathbb{Z}_2^n$, such that the two functions $next(C, w)$ and $prev(C, w)$ can be implemented by inspecting at most $6 + \log n$ cells while writing only 2 cells.*

6.1 Organisation

Prat II of the thesis is organised as follows: In Chapter 7 we present the preliminary notions, definitions and tools used for the construction of quasi-Gray codes. The chapter also presents a brief overview of the construction. In Chapter 8 we present space-optimal quasi-Gray code having logarithmic read complexity over odd sized alphabet set. Next, in Chapter 9 we give quasi-Gray code over even sized alphabet set, that misses only linearly many code words and has logarithmic read complexity. We provide the concluding remark along with a brief discussion on prospective open problem in Chapter 10. Part II is based on [CDKS18].

7. Preliminaries and Overview

In this chapter we first provide the preliminary notations, definitions and the computational model used for building the quasi-Gray codes. Next we give a brief technical overview of the construction. Then we describe the required tools and the underlying structure behind all of our constructions of quasi-Gray codes.

7.1 Preliminaries

Notations: We use the standard notions of groups and fields, and mostly we use only elementary facts about them (see [DF04, LN96] for background.). By $\mathbb{Z}_m$ we mean the set of integers modulo m, i.e., $\mathbb{Z}_m := \mathbb{Z}/m\mathbb{Z}$. Throughout this chapter whenever we use addition and multiplication operation between two elements of $\mathbb{Z}_m$, then we mean the operations within $\mathbb{Z}_m$ that is modulo m. For any $m \in \mathbb{N}$, we let $[m]$ denote the set $\{1, 2, \ldots, m\}$. Unless stated otherwise explicitly, all the logarithms we consider throughout this paper are based 2.

Now we define the notion of counters used in our work.

Definition 52 (Counter)**.** *A* counter *of length ℓ over a domain $\mathcal{D}$ is any cyclic sequence $C = (w_1, \ldots, w_\ell)$ such that $w_1, \ldots, w_\ell$ are distinct elements of $\mathcal{D}$. With the counter C we associate two functions $next(C, w)$ and $prev(C, w)$ that give the successor and predecessor element of w in C, that is for $i \in [\ell]$, $next(C, w_i) = w_j$ where $j - i = 1 \bmod \ell$, and $prev(C, w_i) = w_k$ where $i - k = 1 \bmod \ell$. If $\ell = |\mathcal{D}|$, we call the counter a* space-optimal counter*.*

Often elements in the underlying domain $\mathcal{D}$ have some "structure" to them. In such cases, it is desirable to have a counter such that consecutive elements in the sequence differ by a "small" change in the "structure". We make this concrete in the following definition.

Definition 53 (Gray Code)**.** *Let $\mathcal{D}_1, \ldots, \mathcal{D}_n$ be finite sets. A* Gray code *of length ℓ over the domain $\mathcal{D} = \mathcal{D}_1 \times \cdots \times \mathcal{D}_n$ is a counter $(w_1, \ldots, w_\ell)$ of length ℓ over $\mathcal{D}$ such that any two consecutive strings w_i and w_j, $j - i = 1 \bmod \ell$, differ in exactly one coordinate when viewed as an n-tuple. More generally, if for some constant $c \geq 1$, any two consecutive strings w_i and w_j, $j - i = 1 \bmod \ell$, differ in at most c coordinates such a counter is called a* c-Gray Code*.*

By a quasi-Gray code we mean c-Gray code for some unspecified fixed $c > 0$. In the literature sometimes people do not place any restriction on the relationship between w_ℓ and w_1 and they refer to such a sequence a (quasi)-Gray code. In their terms, our codes would be *cyclic* (quasi)-Gray codes. If $\ell = |\mathcal{D}|$, we call the codes *space-optimal* (quasi-)Gray codes.

Decision Assignment Tree: The computational model we consider in our work is called *Decision Assignment Tree* (DAT). The definition we provide below is a generalization of that given in [Fre78]. It is intended to capture random access machines with small word size.

Let us fix an underlying domain $\mathcal{D}^n$ whose elements we wish to enumerate. In the following, we will denote an element in $\mathcal{D}^n$ by $\langle x_1, x_2, \ldots, x_n \rangle$. A decision

assignment tree is a $|\mathcal{D}|$-ary tree such that each internal node is labeled by one of the variables $x_1, x_2, \ldots, x_n$. Furthermore, each outgoing edge of an internal node is labeled with a distinct element of $\mathcal{D}$. Each leaf node of the tree is labeled by a set of assignment instructions that set new (fixed) values to chosen variables. The variables which are not mentioned in the assignment instructions remain unchanged.

The execution on a decision assignment tree on a particular input vector $\langle x_1, x_2, \ldots, x_n \rangle \in \mathcal{D}^n$ starts from the root of the tree and continues in the following way: at a non-leaf node labeled with a variable x_i, the execution queries x_i and depending on the value of x_i the control passes to the node following the outgoing edge labeled with the value of x_i. Upon reaching a leaf, the corresponding set of assignment statements is used to modify the vector $\langle x_1, x_2, \ldots, x_n \rangle$ and the execution terminates. The modified vector is the output of the execution.

Thus, each decision assignment tree computes a mapping from $\mathcal{D}^n$ into $\mathcal{D}^n$. We are interested in decision assignment trees computing the mapping $next(C, \langle x_1, x_2, \ldots, x_n \rangle)$ for some counter C. When C is space-optimal we can assume, without loss of generality, that each leaf assigns values only to the variables that it reads on the path from the root to the leaf. (Otherwise, the decision assignment tree does not compute a bijection.) We define the *read complexity* of a decision assignment tree T, denoted by $\mathrm{READ}(T)$, as the maximum number of non-leaf nodes along any path from the root to a leaf. Observe that any mapping from $\mathcal{D}^n$ into $\mathcal{D}^n$ can be implemented by a decision assignment tree with read complexity n. We also define the *write complexity* of a decision assignment tree T, denoted by $\mathrm{WRITE}(T)$, as the maximum number of assignment instructions in any leaf.

Instead of the domain $\mathcal{D}^n$, sometime we also use domains that are a Cartesian product of different domains. The definition of a decision assignment tree naturally extends to this case of different variables having different domains.

For any counter $C = (w_1, \ldots, w_\ell)$, we say that C is computed by a decision assignment tree T if and only if for $i \in [\ell]$, $next(C, w_i) = T(w_i)$, where $T(w_i)$ denotes the output string obtained after an execution of T on w_i. Note that any two consecutive strings in the cyclic sequence of C differ by at most $\mathrm{WRITE}(T)$ many coordinates.

For a small constant $c \geq 1$, some domain $\mathcal{D}$, and all large enough n, we will be interested in constructing cyclic counters on $\mathcal{D}^n$ that are computed by decision assignment trees of write complexity c and read complexity $O(\log n)$. By the definition such cyclic counters will necessarily be c-Gray codes.

7.1.1 Construction of Gray codes

For our construction of quasi-Gray codes on a domain $\mathcal{D}^n$ with decision assignment trees of small read and write complexity we will need ordinary Gray codes on a domain $\mathcal{D}^{O(\log n)}$. Several constructions of space-optimal binary Gray codes are known where the oldest one is the binary reflected Gray code [Gra53]. This can be generalized to space-optimal (cyclic) Gray codes over non-binary alphabets (see e.g. [Coh63, Knu11]).

Theorem 54 ([Coh63, Knu11])**.** *For any $m, n \in \mathbb{N}$, there is a space-optimal (cyclic) Gray code over $\mathbb{Z}_m^n$.*

7.2 Overview of the construction

Our construction of Gray codes relies heavily on the notion of s-functions defined by Coppersmith and Grossman [CG75]. An s-function is a permutation τ on $\mathbb{Z}_m^n$ defined by a function $f : \mathbb{Z}_m^s \to \mathbb{Z}_m$ and an $(s+1)$-tuple of indices $i_1, i_2, \ldots, i_s, j \in [n]$ such that $\tau(\langle x_1, x_2, \ldots, x_n\rangle) = (\langle x_1, x_2, \ldots, x_{j-1}, x_j + f(x_{i_1}, \ldots, x_{i_s}), x_{j+1}, \ldots, x_n\rangle)$, where the addition is inside $\mathbb{Z}_m$. Each s-function can be computed by some decision assignment tree that given a vector $x = \langle x_1, x_2, \ldots, x_n\rangle$, inspects $s+1$ coordinates of x and then it writes into a single coordinate of x.

A counter C (quasi-Gray code) on $\mathbb{Z}_m^n$ can be thought of as a permutation on $\mathbb{Z}_m^n$. Our goal is to construct some permutation α on $\mathbb{Z}_m^n$ that can be written as a composition of 2-functions $\alpha_1, \ldots, \alpha_k$, i.e., $\alpha = \alpha_k \circ \alpha_{k-1} \circ \cdots \circ \alpha_1$.

Given such a decomposition, we can build another counter C' on $\mathbb{Z}_m^{r+n}$, where $r = \lceil \log_m k \rceil$, for which the function $next(C', x)$ operates as follows. The first r-coordinates of x serve as an *instruction pointer* $i \in [m^r]$ that determines which α_i should be executed on the remaining n coordinates of x. Hence, based on the current value i of the r coordinates, we perform α_i on the remaining coordinates and then we update the value of i to $i+1$. (For $i > k$ we can execute the identity permutation which does nothing.)

We can use known Gray codes on $\mathbb{Z}_m^r$ to represent the instruction pointer so that when incrementing i we only need to write into one of the coordinates. This gives a counter C' which can be computed by a decision assignment tree that reads $r+3$ coordinates and writes into 2 coordinates of x. (A similar composition technique is implicit in Brodal et al. [BGPS14].) If C is of length $\ell = m^n - t$, then C' is of length $m^{n+r} - tm^r$. In particular, if C is space-optimal then so is C'.

Hence, we reduce the problem of constructing 2-Gray codes to the problem of designing large cycles in $\mathbb{Z}_m^n$ that can be decomposed into 2-functions. Coppersmith and Grossman [CG75] studied precisely the question of, which permutations on $\mathbb{Z}_2^n$ can be written as a composition of 2-functions. They show that a permutation on $\mathbb{Z}_2^n$ can be written as a composition of 2-functions if and only if the permutation is even. Since $\mathbb{Z}_2^n$ is of even size, a cycle of length 2^n on $\mathbb{Z}_2^n$ is an odd permutation and thus it cannot be represented as a composition of 2-functions. However, their result also implies that a cycle of length $2^n - 1$ on $\mathbb{Z}_2^n$ *can* be decomposed into 2-functions.

We want to use the counter composition technique described above in connection with a cycle of length $2^n - 1$. To maximize the length of the cycle C' in $\mathbb{Z}_2^{n+r}$, we need to minimize k, the number of 2-functions in the decomposition. By a simple counting argument, most cycles of length $2^n - 1$ on $\mathbb{Z}_2^n$ require k to be exponentially large in n. This is too large for our purposes. Luckily, there are cycles of length $2^n - 1$ on $\mathbb{Z}_2^n$ that can be decomposed into polynomially many 2-functions, and we obtain such cycles from linear transformations.

There are linear transformations $\mathbb{Z}_2^n \to \mathbb{Z}_2^n$ which define a cycle on $\mathbb{Z}_2^n$ of length $2^n - 1$. For example, the matrix corresponding to the multiplication by a fixed generator of the multiplicative group $\mathbb{F}_{2^n}^*$ of the Galois field $GF[2^n]$ is such a matrix. Such matrices are full rank and they can be decomposed into $O(n^2)$ elementary matrices, each corresponding to a 2-function. Moreover, there

are matrices derived from primitive polynomials that can be decomposed into at most $4n$ elementary matrices.[1] We use them to get a counter on $\mathbb{Z}_2^{n'}$ of length at least $2^{n'} - 20n'$ whose successor and predecessor functions are computable by decision assignment trees of read complexity $\leq 6 + \log n'$ and write complexity 2. Such counter represents 2-Gray code of the prescribed length. For any prime q, the same construction yields 2-Gray codes of length at least $q^{n'} - 5q^2n'$ with decision assignment trees of read complexity $\leq 6 + \log_q n'$ and write complexity 2.

The results of Coppersmith and Grossman [CG75] can be generalized to $\mathbb{Z}_m^n$ as stated in Richard Cleve's thesis [Cle89]. For odd m, if a permutation on $\mathbb{Z}_m^n$ is even then it can be decomposed into 2-functions. Since m^n is odd, a cycle of length m^n on $\mathbb{Z}_m^n$ is an even permutation and so it can be decomposed into 2-functions. If the number k of those functions is small, so the $\log_m k$ is small, we get the sought after counter with small read complexity. However, for most cycles of length m^n on $\mathbb{Z}_m^n$, k is exponential in n.

We show there is a cycle α of length m^n on $\mathbb{Z}_m^n$ that can be decomposed into $O(n^3)$ 2-functions. This in turn gives space-optimal 2-Gray codes on $\mathbb{Z}_m^{n'}$ with decision assignment trees of read complexity $O(\log_m n')$ and write complexity 2.

We obtain the cycle α and its decomposition in two steps. First, for $i \in [n]$, we consider the permutation α_i on $\mathbb{Z}_m^n$ which maps each element $0^{i-1}ay$ onto $0^{i-1}(a+1)y$, for $a \in \mathbb{Z}_m$ and $y \in \mathbb{Z}_m^{n-i}$, while other elements are mapped to themselves. Hence, α_i is a product of m^{n-i} disjoint cycles of length m. We show that $\alpha = \alpha_n \circ \alpha_{n-1} \circ \cdots \circ \alpha_1$ is a cycle of length m^n. In the next step we decompose each α_i into $O(n^2)$ 2-functions.

For $i \leq n-2$, we can decompose α_i using the technique of Ben-Or and Cleve [BC92] and its refinement in the form of catalytic computation of Buhrman et al. [BCK+14]. We can think of $x \in \mathbb{Z}_m^n$ as content of n memory registers, where $x_1, \ldots, x_{i-1}$ are the input registers, x_i is the output register, and $x_{i+1}, \ldots, x_n$ are the working registers. The catalytic computation technique gives a program consisting of $O(n^2)$ instructions, each being equivalent to a 2-function, which performs the desired adjustment of x_i based on the values of $x_1, \ldots, x_{i-1}$ without changing the ultimate values of the other registers. (We need to increment x_i iff $x_1, \ldots, x_{i-1}$ are all zero.) This program directly gives the desired decomposition of α_i, for $i \leq n-2$. (Our proof in Chapter 8 uses the language of permutations.)

The technique of catalytic computation fails for α_{n-1} and α_n as the program needs at least two working registers to operate. Hence, for α_{n-1} and α_n we have to develop entirely different technique. This is not trivial and quite technical but it is nevertheless possible, thanks to the specific structure of α_{n-1} and α_n.

7.3 The Key Tools

In this section we talk about the key tools used to construct quasi-Gray codes over $\mathbb{Z}_m^n$ for $m \in \mathbb{N}$. First we show how to compose decision assignment trees over different domains to get a decision assignment tree for a larger mixed domain. We use the Chinese remainder theorem for this purpose.

[1] Primitive polynomials were previously also used in a similar problem, namely to construct shift-register sequences (see e.g. [Knu11]).

7.3.1 Chinese Remainder Theorem for Counters

Theorem 55 (Chinese Remainder Theorem for Counters)**.** *Let $r, n_1, \ldots, n_r \in \mathbb{N}$ be integers, and let $\mathcal{D}_{1,1}, \ldots, \mathcal{D}_{1,n_1}, \mathcal{D}_{2,1}, \ldots, \mathcal{D}_{r,n_r}$ be some finite sets of size at least two. Let $\ell_1 \geq r-1$ be an integer, and $\ell_2, \ldots, \ell_r$ be pairwise co-prime integers. For $i \in [r]$, let C_i be a counter of length ℓ_i over $\mathcal{D}_i = \mathcal{D}_{i,1} \times \cdots \times \mathcal{D}_{i,n_i}$ computed by a decision assignment tree T_i over n_i variables. Then, there exists a decision assignment tree T over $\sum_{i=1}^{r} n_i$ variables that implements a counter C of length $\prod_{i=1}^{r} \ell_i$ over $\mathcal{D}_1 \times \cdots \times \mathcal{D}_r$. Furthermore,* $\mathrm{READ}(T) = n_1 + \max\{\mathrm{READ}(T_i)\}_{i=2}^{r}$, *and* $\mathrm{WRITE}(T) = \mathrm{WRITE}(T_1) + \max\{\mathrm{WRITE}(T_i)\}_{i=2}^{r}$.

Proof. For any $i \in [r]$, let the counter $C_i = (w_{i,1}, \ldots, w_{i,\ell_i})$. Let $x_1, \ldots, x_r$ be variables taking values in $\mathcal{D}_1, \ldots, \mathcal{D}_r$, respectively. The following procedure, applied repeatedly, defines the counter C:

If $x_1 = w_{1,i}$ for some $i \in [r-1]$ **then**
$x_{i+1} \leftarrow next(C_{i+1}, x_{i+1})$
$x_1 \leftarrow next(C_1, x_1)$
else
$x_1 \leftarrow next(C_1, x_1)$.

It is easily seen that the above procedure defines a valid cyclic sequence when starting at $w_{1,i_1}, \ldots, w_{r,i_r}$ for any $\langle i_1, i_2, \ldots, i_r \rangle \in [\ell_1] \times \cdots \times [\ell_r]$. That is, every element has a unique predecessor and a unique successor, and that the sequence is cyclic. It can easily be implemented by a decision assignment tree, say T. First it reads the value of x_1. Since $x_1 \in \mathcal{D}_1 = \mathcal{D}_{1,1} \times \cdots \times \mathcal{D}_{1,n_1}$, it queries n_1 components. Then, depending on the value of x_1, it reads and updates another component, say x_j. This can be accomplished using the decision assignment tree T_j. We also update the value of x_1, and to that end we use the appropriate assignments from decision assignment tree T_1. Observe that irrespective of how efficient T_1 is, we read x_1 completely to determine which of the remaining $r-1$ counters to update. Hence, $\mathrm{READ}(T) = n_1 + \max\{\mathrm{READ}(T_i)\}_{i=2}^{r}$, and $\mathrm{WRITE}(T) = \mathrm{WRITE}(T_1) + \max\{\mathrm{WRITE}(T_i)\}_{i=2}^{r}$.

Now it only remains to show that the counter described above is indeed of length $\prod_{i=1}^{r} \ell_i$. Thus, it suffices to establish that starting with the string $\langle w_{1,1}, \ldots, w_{r,1} \rangle$, we can generate the string $\langle w_{1,i_1}, \ldots, w_{r,i_r} \rangle$ for any $\langle i_1, \ldots, i_r \rangle \in [\ell_1] \times \cdots \times [\ell_r]$. Let us assume $i_1 = 1$. At the end of the proof we will remove this assumption. Suppose the string $\langle w_{1,1}, w_{2,i_2}, \ldots, w_{r,i_r} \rangle$ is reachable from $\langle w_{1,1}, w_{2,1}, \ldots, w_{r,1} \rangle$ in t steps. As our procedure always increment x_1, t must be divisible by ℓ_1. Let $d = t/\ell_1$. Furthermore, the procedure increments a variable x_i, $i \neq 1$, exactly after ℓ_1 steps. Thus, $\langle w_{1,1}, w_{2,i_2}, \ldots, w_{r,i_r} \rangle$ is reachable if and only if d satisfies the following equations:

$$\begin{aligned} d &\equiv i_2 - 1 \pmod{\ell_2} \\ d &\equiv i_3 - 1 \pmod{\ell_3} \\ &\vdots \\ d &\equiv i_r - 1 \pmod{\ell_r}. \end{aligned}$$

Since $\ell_2, \ldots, \ell_r$ are pairwise co-prime, Chinese Remainder Theorem (for a reference, see [DPS96]) guarantees the existence of a unique integral solution d such that $0 \leq d < \prod_{i=2}^{r} \ell_i$. Hence, $\langle w_{1,1}, w_{2,,i_2}, \ldots, w_{r,i_r} \rangle$ is reachable from $\langle w_{1,1}, w_{2,1}, \ldots, w_{r,1} \rangle$ in at most $\prod_{i=1}^{r} \ell_i$ steps.

Now we remove the assumption $i_1 = 1$, i.e., $w_{1,i_1} \neq w_{1,1}$. Consider the string $\langle w_{1,1}, w_{2,i'_2}, \ldots, w_{r,i'_r} \rangle$ where $w_{j,i'_j} = w_{j,i_j - 1}$ for $2 \leq j \leq \min\{i_1, r\}$, and $w_{j,i'_j} = w_{j,i_j}$ for $j > \min\{i_1, r\}$. From the arguments in the previous paragraph, we know that this tuple is reachable. We now observe that the next $i_1 - 1$ steps increment $w_{1,1}$ to w_{1,i_1} and w_{j,i'_j} to w_{j,i_j} for $2 \leq j \leq \min\{i_1, r\}$, thus, reaching the desired string $\langle w_{1,i_1}, \ldots, w_{r,i_r} \rangle$. □

Remark. *We remark that if C_i's are space-optimal in Theorem 55, then so is C.*

In the above proof, we constructed a special type of a counter where we always read the first coordinate, incremented it, and further depending on its value, we may update the value of another coordinate. From now on we refer to such type of counters as *hierarchical counters*. One can note that the above theorem is similar to the well known Chinese Remainder Theorem and has similar type of application for constructing of space-optimal quasi-Gray codes over $\mathbb{Z}_m^n$ for arbitrary $m \in \mathbb{N}$.

Lemma 36. *Let $n, m \in \mathbb{N}$ be such that $m = 2^k o$, where o is odd and $k \geq 0$. Given decision assignment trees T_1 and T_2 computing space-optimal (quasi-)Gray codes over $(\mathbb{Z}_{2^k})^{n-1}$ and $\mathbb{Z}_o^{n-1}$, respectively, there exists a decision assignment tree T implementing a space-optimal quasi-Gray code over $\mathbb{Z}_m^n$ such that* $\mathrm{READ}(T) = 1 + \max\{\mathrm{READ}(T_1), \mathrm{READ}(T_2)\}$, *and* $\mathrm{WRITE}(T) = 1 + \max\{\mathrm{WRITE}(T_1), \mathrm{WRITE}(T_2)\}$.

Proof. We will view $\mathbb{Z}_m^n$ as $Z_m \times (\mathbb{Z}_{2^k})^{n-1} \times (\mathbb{Z}_o)^{n-1}$ and simulate a decision assignment tree operating on $Z_m \times (\mathbb{Z}_{2^k})^{n-1} \times (\mathbb{Z}_o)^{n-1}$ on $\mathbb{Z}_m^n$. From the Chinese Remainder Theorem (see [DPS96]), we know that there exists a bijection (in fact, an isomorphism) $f \colon \mathbb{Z}_m \to \mathbb{Z}_{2^k} \times \mathbb{Z}_o$. We denote the tuple $f(z)$ by $\langle f_1(z), f_2(z) \rangle$. From Theorem 55 we know that there exists a decision assignment tree T' over $\mathbb{Z}_m \times (\mathbb{Z}_{2^k})^{n-1} \times (\mathbb{Z}_o)^{n-1}$ computing a space-optimal quasi-Gray code such that $\mathrm{READ}(T') = 1 + \max\{\mathrm{READ}(T_1), \mathrm{READ}(T_2)\}$, and $\mathrm{WRITE}(T') = 1 + \max\{\mathrm{WRITE}(T_1), \mathrm{WRITE}(T_2)\}$.

We can simulate actions of T' on an input $\mathbb{Z}_m^n$ to obtain the desired decision assignment tree T. Indeed, whenever T' queries x_1, T queries the first coordinate of its input. Whenever T' queries the i-th coordinate of $(\mathbb{Z}_{2^k})^{n-1}$, T queries the $(i+1)$-th coordinate of its input and makes its decision based on the $f_1(\cdot)$ value of that coordinate. Similarly, whenever T' queries the j-th coordinate of $(\mathbb{Z}_o)^{n-1}$, T queries the $(j+1)$-th coordinate and makes its decision based on the $f_2(\cdot)$ value of that coordinate. Assignments by T are handled in similar fashion by updating only the appropriate part of $\langle f_1(z), f_2(z) \rangle$. (Notice, queries made by T might reveal more information than queries made by T'.) □

Before proceeding further, we would also like to point out that to get a space-optimal decision assignment tree over $\mathbb{Z}_{2^k}$, it suffices to a get space-optimal decision assignment trees over $\mathbb{Z}_2$ for arbitrary dimensions. Thus, to get a decision assignment tree implementing space-optimal quasi-Gray codes over $\mathbb{Z}_m$, we only

need decision assignment trees implementing space-optimal quasi-Gray codes over $\mathbb{Z}_2$ and $\mathbb{Z}_o$. This also justifies our sole focus on construction of space-optimal decision assignment trees over $\mathbb{Z}_2$ and $\mathbb{Z}_o$ in the later chapters.

Lemma 37. *If, for all $n \in \mathbb{N}$, there exists a decision assignment tree T implementing a space-optimal (quasi-)Gray code over $\mathbb{Z}_2^n$, then for any k and $n \in \mathbb{N}$, there exists a decision assignment tree T' implementing a space-optimal (quasi-)Gray code over $(\mathbb{Z}_{2^k})^n$ such that the read and write complexity remain the same.*

Proof. Consider any bijective map $f : \mathbb{Z}_{2^k} \to \mathbb{Z}_2^k$. For example, one can take standard binary encoding of integers ranging from 0 to $2^k - 1$ as the bijective map f. Next, define another map $g : (\mathbb{Z}_{2^k})^n \to \mathbb{Z}_2^{kn}$ as follows: $g(x_1, \ldots, x_n) = \langle f(x_1), \ldots, f(x_n) \rangle$. Now consider T that implements a space-optimal (quasi-)Gray code over $\mathbb{Z}_2^{kn}$. We fix a partition of the variables $\{1, \ldots, k\} \uplus \cdots \uplus \{(n-1)k + 1, \ldots, nk\}$ into n blocks of k variables each.

We now construct a decision assignment tree T' over $(\mathbb{Z}_{2^k})^n$ using T and the map f. As in the proof of Lemma 36, our T' follows T in the decision making. That is, if T queries a variable, then T' queries the block in the partition where the variable lies. (Again, as noted before, T' may get more information than required by T.) Upon reaching a leaf, using f, T' updates the blocks depending on T's updates to the variables.

□

7.3.2 Permutation Group and Decomposition of Counters

We start this part with some basic notation and facts about the permutation group which we will use heavily in our construction. The set of all permutations over a domain $\mathcal{D}$ forms a group under the composition operation, denoted by $\circ$, which is defined as follows: for any two permutations σ and α, $\sigma \circ \alpha(x) = \sigma(\alpha(x))$, where $x \in \mathcal{D}$. The corresponding group, denoted $\mathcal{S}_N$, is the *symmetric group* of order $N = |\mathcal{D}|$. We say, a permutation $\sigma \in \mathcal{S}_N$ is a *cycle* of length ℓ if there are distinct elements $a_1, \ldots, a_\ell \in [N]$ such that for $i \in [\ell - 1]$, $a_{i+1} = \sigma(a_i)$, $a_1 = \sigma(a_\ell)$, and for all $a \in [N] \setminus \{a_1, a_2, \ldots, a_\ell\}$, $\sigma(a) = a$. We denote such a cycle by $(a_1, a_2, \cdots, a_\ell)$. Below we state few simple facts about composition of cycles.

Proposition 56. *Consider two cycles $C_1 = (t, a_1, \cdots, a_{\ell_1})$ and $C_2 = (t, b_1, \cdots, b_{\ell_2})$ where for any $i \in [\ell_1]$ and $j \in [\ell_2]$, $a_i \neq b_j$. Then, $C = C_2 \circ C_1$ is the cycle $(t, a_1, \cdots, a_{\ell_1}, b_1, \cdots, b_{\ell_2})$ of length $\ell_1 + \ell_2 + 1$.*

Proposition 57. *If $\sigma \in \mathcal{S}_N$ is a cycle of length ℓ, then for any $\alpha \in \mathcal{S}_N$, $\alpha \circ \sigma \circ \alpha^{-1}$ is also a cycle of length ℓ. Moreover, if $\sigma = (a_1, a_2, \cdots, a_\ell)$, then $\alpha \circ \sigma \circ \alpha^{-1} = (\alpha(a_1), \alpha(a_2), \cdots, \alpha(a_\ell))$.*

The permutation $\alpha \circ \sigma \circ \alpha^{-1}$ is called the *conjugate* of σ with respect to α. The above proposition is a special case of a well known fact about the cycle structure of conjugates of any permutation and can be found in any standard text book on Group Theory (e.g., Proposition 10 in Chapter 4.3 of [DF04].).

Roughly speaking, a counter of length ℓ over $\mathcal{D}$, in the language of permutations, is nothing but a cycle of the same length in $\mathcal{S}_{|\mathcal{D}|}$. We now make this

correspondence precise and give a construction of a decision assignment tree that implements such a counter.

Lemma 38. *Let $\mathcal{D} = \mathcal{D}_1 \times \cdots \times \mathcal{D}_r$ be a domain. Suppose $\sigma_1, \ldots, \sigma_k \in \mathcal{S}_{|\mathcal{D}|}$ are such that $\sigma = \sigma_k \circ \sigma_{k-1} \circ \cdots \circ \sigma_1$ is a cycle of length ℓ. Let $T_1, \ldots, T_k$ be decision assignment trees that implement $\sigma_1, \ldots, \sigma_k$ respectively. Let $\mathcal{D}' = \mathcal{D}'_1 \times \cdots \times \mathcal{D}'_{r'}$ be a domain such that $|\mathcal{D}'| \geq k$, and let T' be a decision assignment tree that implements a counter C' of length k' over $\mathcal{D}'$ where $k' \geq k$.*

Then, there exists a decision assignment tree T that implements a counter of length $k'\ell$ over $\mathcal{D}' \times \mathcal{D}$ such that $\mathrm{READ}(T) = r' + \max\{\mathrm{READ}(T_i)\}_{i=1}^{k}$, and $\mathrm{WRITE}(T) = \mathrm{WRITE}(T') + \max\{\mathrm{WRITE}(T_i)\}_{i=1}^{k}$.

Proof. Suppose $C' = (a_1, \ldots, a_{k'})$. Now let us consider the following procedure P: on any input $\langle x_1, x_2 \rangle \in \mathcal{D}' \times \mathcal{D}$,

If $x_1 = a_j$ for some $j \in [k]$ **then**
$x_2 \leftarrow \sigma_j(x_2)$
$x_1 \leftarrow next(C', x_1)$
else
$x_1 \leftarrow next(C', x_1)$.

Now using a similar argument as in the proof of Theorem 55, the above procedure is easily seen to be implementable using a decision assignment tree T of the prescribed complexity. Each time we check the value of $x_1 \in \mathcal{D}' = \mathcal{D}'_1 \times \cdots \times \mathcal{D}'_{r'}$. Thus, we need to read r' components. Depending on the value of x_1, we may apply σ_j on x_2 using the decision assignment tree T_j. Then we update the value of x_1. Hence, $\mathrm{READ}(T) = r' + \max\{\mathrm{READ}(T_i)\}_{i=1}^{k}$, and $\mathrm{WRITE}(T) = \mathrm{WRITE}(T') + \max\{\mathrm{WRITE}(T_i)\}_{i=1}^{k}$.

Let $(w_1, w_2 \cdots, w_\ell)$ be the cycle of length ℓ given by σ. We now argue that the procedure P generates a counter of length $k'\ell$ over $\mathcal{D}' \times \mathcal{D}$ starting at $\langle a_1, w_1 \rangle$. Without loss of generality, let us assume that $\sigma = \sigma_{k'} \circ \cdots \circ \sigma_{k+1} \circ \sigma_k \circ \sigma_{k-1} \circ \cdots \circ \sigma_1$, where for $j \geq k+1$, σ_j is the identity map. Fix $j \in [k']$. Define $\alpha_j = \sigma_{j-1} \circ \cdots \circ \sigma_1$, and $\tau_j = \alpha_j \circ \sigma \circ \alpha_j^{-1} = \sigma_{j-1} \circ \cdots \circ \sigma_1 \circ \sigma_{k'} \circ \cdots \circ \sigma_j$. For $i = 0, 1, \ldots, \ell$, let $\langle g_i, e_i \rangle = P^{ik'}(\langle a_j, \alpha_j(w_1) \rangle)$ where $P^{ik'}$ denotes ik' invocations of P. Since P increments x_1 in every invocation, for $i = 1, 2, \ldots, \ell$, $g_i = a_j$ and $e_i = \tau_j(e_{i-1})$.

By Proposition 57, τ_j is a cycle $(\alpha_j(w_1)\alpha_j(w_2) \cdots \alpha_j(w_\ell))$ of length ℓ. Hence, $e_1, \ldots, e_\ell$ are all distinct and $e_\ell = e_0$.

As a consequence we conclude that for any $x \in \mathcal{D}' \times \mathcal{D}$ and $1 \leq j_1 \neq j_2 \leq k'\ell$, $P^{j_1}(x) \neq P^{j_2}(x)$ and $P^{k'\ell}(x) = x$. This completes the proof. □

In the following two chapters we describe the construction of $\sigma_1, \cdots, \sigma_k \in \mathcal{S}_N$ where $N = m^n$ for some $m, n \in \mathbb{N}$ and show how the value of k depends on the length of the cycle $\sigma = \sigma_k \circ \sigma_{k-1} \circ \cdots \circ \sigma_1$.

8. Space-optimal Quasi-Gray Codes Over Odd Sized Alphabets

This chapter deals with space-optimal counter over $\mathbb{Z}_m^n$ for any Odd m. We start by recalling Theorem 50 in terms of decision assignment tree complexity.

Theorem 58 (Restatement of Theorem 50)**.** *For any odd $m \in \mathbb{N}$ and any positive integer $n \geq 15$, there is a space-optimal 2-Gray code over $\mathbb{Z}_m^n$ that can be computed by a decision assignment tree T such that* $\mathrm{READ}(T) \leq 4 \log_m n$.

Before providing the details, we give a short plan of how we proceed with the construction of the quasi-Gray code. First we set $n' = n - c \cdot \log n$ for some constant $c > 0$ that will be fixed later. Then we define suitable permutations $\alpha_1, \ldots, \alpha_{n'} \in \mathcal{S}_N$ where $N = m^{n'}$ such that their composition $\alpha_{n'} \circ \cdots \circ \alpha_1$ is a cycle of length $m^{n'}$. Next we show that each α_i can be further decomposed into $\alpha_{i,1}, \ldots, \alpha_{i,j} \in \mathcal{S}_N$ for some j, such that each $\alpha_{i,r}$ for $r \in [j]$ can be computed using a decision assignment tree with read complexity 3 and write complexity 1. Finally to complete the construction we use Lemma 38 with $\alpha_{i,r}$'s playing the role of $\sigma_1, \ldots, \sigma_k$ in the lemma.

We recall the notion of r-functions over $\mathbb{Z}_m^n$ that was introduced by Coppersmith and Grossman [CG75] for $m = 2$. Below we generalize that definition for any $m \in \mathbb{N}$.

Definition 59. *For any $r \in [n-1]$, an* r-function *on $\mathbb{Z}_m^n$ is a permutation τ over $\mathbb{Z}_m^n$ identified by a subset $\{i_1, \ldots, i_r, j\} \subseteq [n]$ of size $r+1$ and a function $f : \mathbb{Z}_m^r \to \mathbb{Z}_m$ such that for any $\langle a_1, \ldots, a_n \rangle \in \mathbb{Z}_m^n$,*

$$\tau(\langle a_1, \ldots, a_n \rangle) = \langle a_1, \ldots, a_{j-1}, a_j + f(a_{i_1}, \ldots, a_{i_r}), a_{j+1}, \ldots, a_n \rangle.$$

Observe that any r-function can be implemented using a decision assignment tree T that queries $x_{i_1}, \ldots, x_{i_r}$ and x_j at internal nodes, and at leaves it assigns value only to the variable x_j. Thus, $\mathrm{READ}(T) = r + 1$ and $\mathrm{WRITE}(T) = 1$.

Claim 39. *Any r-function on $\mathbb{Z}_m^n$ can be implemented using a decision assignment tree T with n variables such that* $\mathrm{READ}(T) = r + 1$ *and* $\mathrm{WRITE}(T) = 1$.

8.1 Construction of the Counter

We are now ready to provide details of our construction of a space-optimal quasi-Gray code over $\mathbb{Z}_m^n$ for any odd m. Define $n' := n - c \cdot \log_m n$ for some constant $c > 0$ that will be fixed later.

Step 1: Construction of $\alpha_1, \ldots, \alpha_{n'}$.

We consider specific permutations $\alpha_1, \ldots, \alpha_{n'}$ over $\mathbb{Z}_m^{n'}$ such that $\alpha = \alpha_{n'} \circ \cdots \circ \alpha_1$ is a cycle of length $m^{n'}$. We define them below.

Definition 60. *Let m and n' be natural numbers. For $i \in [n']$, we define α_i to be the permutation given by the following map: for any $\langle x_1, \ldots, x_i, \ldots, x_{n'} \rangle \in \mathbb{Z}_m^{n'}$,*

$$\alpha_i\left(\langle x_1, \ldots, x_i, \ldots, x_{n'} \rangle\right) = \begin{cases} \langle x_1, \ldots, x_i + 1, \ldots, x_{n'} \rangle & \text{if } x_j = 0,\ \forall j \in [i-1] \\ \langle x_1, \ldots, x_i, \ldots, x_{n'} \rangle & \text{otherwise.} \end{cases}$$

The addition operation in the mapping $x_i \leftarrow x_i + 1$ is within $\mathbb{Z}_m$.

The following observation is easily seen from the definitions of r-functions and α_i.

Claim 40. *For any $i \in [n']$, α_i is an $(i-1)$-function on $\mathbb{Z}_m^{n'}$. Furthermore, each α_i is composed of disjoint cycles of length m over $\mathbb{Z}_m^{n'}$.*

We now establish a crucial property of the α_i's, i.e., their composition is a *full* length cycle.

Claim 41. *$\alpha = \alpha_{n'} \circ \cdots \circ \alpha_1$ is a cycle of length $m^{n'}$.*

Proof. Consider the sequence of permutations $\tau_1, \ldots, \tau_{n'}$ such that $\tau_i = \alpha_i \circ \alpha_{i-1} \circ \cdots \circ \alpha_1$ for $i \in [n']$. Clearly, $\tau_{n'} = \alpha$. We now prove the claim. In fact, we establish the following stronger claim: for $i \in [n']$, τ_i is a permutation composed of $m^{n'-i}$ disjoint cycles, each of the cycles being of length m^i. Furthermore, for every $\langle a_{i+1}, \ldots, a_{n'} \rangle \in \mathbb{Z}_m^{n'-i}$ there is a cycle that involves all tuples of the form $\langle x_1, \ldots, x_i, a_{i+1}, \ldots, a_{n'} \rangle$. The claim, α is a cycle of length $m^{n'}$, follows as a consequence. We prove the stronger claim by induction on i.

Base case: $\tau_1 = \alpha_1$. From the definition of α_1, it follows that there is a cycle of length m of the form $(\langle 0, a_2, \ldots, a_{n'} \rangle, \langle 1, a_2, \ldots, a_{n'} \rangle, \ldots, \langle m-1, a_2, \ldots, a_{n'} \rangle)$ for each $\langle a_2, \ldots, a_{n'} \rangle \in \mathbb{Z}_m^{n'-1}$. Hence our induction hypothesis clearly holds for the base case.

Induction step: Suppose our induction hypothesis holds until some $i \in [n']$ and we would like to establish it for $i+1$. Let us consider the permutation α_{i+1}. We know that it is composed of $m^{n'-(i+1)}$ disjoint cycles of length m. Indeed, for each $\langle a_{i+2}, \ldots, a_{n'} \rangle \in \mathbb{Z}_m^{n'-(i+1)}$, α_{i+1} contains a cycle that involves all m tuples where the first i coordinates are all 0 and the last $n' - (i+1)$ coordinates are set to $\langle a_{i+2}, \ldots, a_{n'} \rangle$. From the cycle decomposition of α_{i+1} and τ_i into disjoint cycles, it is clear that for any cycle say C in α_{i+1} there are m disjoint cycles $C_1, \ldots, C_m$ in τ_i, each of them intersecting C in exactly one element. Consider $C' = C \circ C_1 \circ \cdots \circ C_m$. By repeated application of Proposition 56, we conclude that C' is a cycle of length $\sum_{i=1}^{m} |C_i| = m^{i+1}$. (Here by $|C_i|$ we mean the length of the cycle C_i.) Also C' involves tuples where the last $n' - (i+1)$ coordinates are set to some fixed $\langle a_{i+2}, \ldots, a_{n'} \rangle \in \mathbb{Z}_m^{n'-(i+1)}$. Thus, C' is a cycle of length m^{i+1} containing all tuples of the form $\langle x_1, \ldots, x_{i+1}, a_{i+2}, \ldots, a_{n'} \rangle$. Since $\tau_{i+1} = \alpha_{i+1} \circ \tau_i$ and α_{i+1} contains $m^{n'-(i+1)}$ disjoint cycles, we conclude that τ_{i+1} consists of exactly $m^{n'-(i+1)}$ disjoint cycles, each of length m^{i+1} and containing tuples of the required form. This finishes the proof. □

It can be noted that the above step does not use the fact that m is odd and, thus, it is true for any $m \in \mathbb{N}$. If we were to directly implement α_i by a decision assignment tree, its read complexity would be i. Hence, we would not get any savings in Lemma 38. So we need to further decompose α_i into permutations of small read complexity.

Step 2: Further decomposition of α_i's.

Our goal is to describe α_i is a composition of 2-functions. Recall, Claim 40, each α_i is an $(i-1)$-function on $\mathbb{Z}_m^{n'}$. Suppose, for $i \in [n']$, there exists a set of 2-functions $\alpha_{i,1}, \ldots, \alpha_{i,k_i}$ such that $\alpha_i = \alpha_{i,k_i} \circ \cdots \circ \alpha_{i,1}$. Then using Lemma 38, where $\alpha_{i,k}$'s play the role of σ_j's, we obtain a decision assignment tree implementing a 2-Gray code with potentially low read complexity. Indeed, each $\alpha_{i,k}$ has low read complexity by Claim 39, hence the read complexity essentially depends on how large is $\sum_i k_i$. In the following we will argue that α_i's can be decomposed into a *small* set of 2-functions, thus keeping the maximum k_i small. As a result, the read complexity bound in Theorem 58 will follow.

Note α_1, α_2 and α_3 are already 2-functions. In the case of α_i, $4 \leq i \leq n'-2$, we can directly adopt the technique from [BC92, BCK$^+$14] to generate the desired set of 2-functions. However, as discussed in Chapter 7, that technique falls short when $i > n' - 2$. (It needs two free registers to operate.) For $i = n' - 1$, it is possible to generalize the proof technique of [CG75] to decompose $\alpha_{n'-1}$. Unfortunately all the previously known techniques fail to decompose $\alpha_{n'}$ and we have to develop a new technique.

First we provide the adaptation of [BC92, BCK$^+$14], and then develop a new technique that allows us to express both $\alpha_{n'-1}$ and $\alpha_{n'}$ as a composition of small number of 2-functions, thus overcoming the challenge described above.

Lemma 42. *For any $4 \leq i \leq n' - 2$, let α_i be the permutation given by Definition 60. Then there exists a set of 2-functions $\alpha_{i,1}, \ldots, \alpha_{i,k_i}$ such that $\alpha_i = \alpha_{i,k_i} \circ \cdots \circ \alpha_{i,1}$, and $k_i \leq 4(i-1)^2 - 3$.*

It is worth noting that, although in this section we consider m to be odd, the above lemma holds for any $m \in \mathbb{N}$. In [BC92], computation uses only addition and multiplication from the ring, whereas we can use any function $g : \mathbb{Z}_m \to \mathbb{Z}_m$. This subtle difference makes the lemma to be true for any $m \in \mathbb{N}$ instead of being true only for prime powers.

Proof. Pick $i \leq n' - 2$. Let us represent α_i as an $(i-1)$-function. From the definition we have,

$$\alpha_i\left(\langle a_1, \ldots, a_i, \ldots, a_{n'}\rangle\right) = \langle a_1, \ldots, a_{i-1}, a_i + f(a_1, \ldots, a_{i-1}), a_{i+1}, \ldots, a_{n'}\rangle,$$

where the map $f : \mathbb{Z}_m^{i-1} \to \mathbb{Z}_m$ is defined as follows:

$$f(a_1, \ldots, a_{i-1}) = \begin{cases} 1 & \text{if } (a_1, \ldots, a_{i-1}) = (0, \ldots, 0), \\ 0 & \text{otherwise.} \end{cases}$$

Observe that f is an indicator function of a tuple; in particular, of the all-zeroes tuple. To verify the lemma, we would prove a stronger claim than the statement of the lemma. Consider the set $\mathcal{S}$ of r-functions, $1 \leq r \leq n' - 3$, such that the function f, used to define them is the indicator function of the all-zeroes tuple. That is, $\tau \in \mathcal{S}$ if and only if there exists a set $\{i_1, i_2, \ldots, i_r\} \subseteq [n']$ of size r and a $j \in [n'] \setminus \{i_1, i_2, \ldots, i_r\}$ such that

$$\tau(\langle a_1, \ldots, a_{n'}\rangle) = \langle a_1, \ldots, a_{j-1}, a_j + f(a_{i_1}, \ldots, a_{i_r}), a_{j+1}, \ldots, a_{n'}\rangle,$$

where $f(x_1, \ldots, x_r) = 1$ if $(x_1, \ldots, x_r) = (0, \ldots, 0)$, and 0 otherwise. Observe that $\alpha_i \in \mathcal{S}$ for $4 \leq i \leq n' - 2$. We establish the following stronger claim.

Claim 43. *For an r-function $\tau \in \mathcal{S}$, there exist 2-functions $\tau_1, \ldots, \tau_{k_r}$ such that $\tau = \tau_{k_r} \circ \cdots \circ \tau_1$ and $k_r \leq 4r^2 - 3$. (We stress that $\tau_1, \ldots, \tau_{k_r}$ need not belong to $\mathcal{S}$.)*

Clearly, the claim implies the lemma. We now prove the claim by induction on r. The base case is $r \leq 2$, in which case the claim trivially holds. Suppose the claim holds for all $(r-1)$-functions in $\mathcal{S}$. Let $\tau \in \mathcal{S}$ be an r-function identified by the set $S := \{i_1, i_2, \ldots, i_r\} \subseteq [n']$ and $j \in [n'] \setminus S$. Since f is an indicator function, it can be expressed as a product of indicator functions. That is, $f(a_{i_1}, \ldots, a_{i_r}) = \prod_{s \in S} g(a_s)$ where $g : \mathbb{Z}_m \to \mathbb{Z}_m$ is the following $\{0,1\}$-map:

$$g(y) = \begin{cases} 1 & \text{if } y = 0, \\ 0 & \text{otherwise.} \end{cases}$$

Consider a partition of S into two sets A and B of sizes $\lfloor r/2 \rfloor$ and $\lceil r/2 \rceil$, respectively. Let j_1 and j_2 be two distinct integers in $[n'] \setminus S \cup \{j\}$. The existence of such integers is guaranteed by the bound on r. We now express τ as a composition of $(r/2)$-functions and 2-functions, and then use induction hypothesis to complete the proof. The decomposition of τ is motivated by the following identity:

$$\begin{aligned} a_j + \prod_{s \in S} g(a_s) = \; & a_j + \left(a_{j_1} + \prod_{s \in A} g(a_s) - a_{j_1}\right)\left(a_{j_2} + \prod_{s \in B} g(a_s) - a_{j_2}\right) \\ = \; & a_j + \left(a_{j_1} + \prod_{s \in A} g(a_s)\right)\left(a_{j_2} + \prod_{s \in B} g(a_s)\right) \\ & - \left(a_{j_1} + \prod_{s \in A} g(a_s)\right) a_{j_2} - a_{j_1}\left(a_{j_2} + \prod_{s \in B} g(a_s)\right) + a_{j_1} a_{j_2}. \end{aligned}$$

Therefore, we consider three permutations γ, τ_A and τ_B such that for any $\langle a_1, \ldots, a_{n'} \rangle \in \mathbb{Z}_m^{n'}$ their maps are given as follows:

$$\begin{aligned} \gamma\left(\langle a_1, \ldots, a_n \rangle\right) &= \langle a_1, \ldots, a_{j-1}, a_j + a_{j_1} a_{j_2}, a_{j+1}, \ldots, a_{n'} \rangle, \\ \tau_A\left(\langle a_1, \ldots, a_{n'} \rangle\right) &= \langle a_1, \ldots, a_{j_1 - 1}, a_{j_1} + \textstyle\prod_{s \in A} g(a_s), a_{j_1+1}, \ldots, a_{n'} \rangle, \text{ and} \\ \tau_B\left(\langle a_1, \ldots, a_{n'} \rangle\right) &= \langle a_1, \ldots, a_{j_2 - 1}, a_{j_2} + \textstyle\prod_{s \in B} g(a_s), a_{j_2+1}, \ldots, a_{n'} \rangle, \end{aligned}$$

where both the multiplications and additions are in $\mathbb{Z}_m$. Using the identity it is easy to verify the following decomposition of τ:

$$\tau = \tau_B^{-1} \circ \gamma^{-1} \circ \tau_A^{-1} \circ \gamma \circ \tau_B \circ \gamma^{-1} \circ \tau_A \circ \gamma.$$

Clearly, γ is a 2-function, while τ_A and τ_B are $\lfloor r/2 \rfloor$-function and $\lceil r/2 \rceil$-function, respectively, and belong to $\mathcal{S}$. By induction hypothesis τ_A and τ_B can be expressed as a composition of 2-functions. Thus their inverses too. Hence we obtain a decomposition of τ in terms of 2-functions. The bound on k_r, the length of the decomposition, follows from the following recurrence:

$$T(r) \leq 2T(\lfloor r/2 \rfloor) + 2T(\lceil r/2 \rceil) + 4.$$

□

We would like to mention that another decomposition of τ in terms of 2-functions can be obtained by following the proof of [CG75], albeit with a much worse bound on the value of k_r. Further, by strengthening the induction hypothesis, it is easily seen that the above proof can be generalized to hold for certain special type of r-functions. Let β be an r-function, $r \leq n' - 3$, such that for any $\langle a_1, \ldots, a_{n'} \rangle \in \mathbb{Z}_m^{n'}$,

$$\beta\left(\langle a_1, \ldots, a_{n'} \rangle\right) = \langle a_1, \ldots, a_{i-1}, a_i + f_e(a_{i_1}, \ldots, a_{i_r}), a_{i+1}, \ldots, a_{n'} \rangle,$$

where the function $f_e : \mathbb{Z}_m^r \to \mathbb{Z}_m$ is defined by:

$$f_e(x) = \begin{cases} b & \text{if } x = e, \\ 0 & \text{otherwise,} \end{cases}$$

for some $b \in \mathbb{Z}_m$ and $e \in \mathbb{Z}_m^r$, i.e., f_e is some constant multiple of the characteristic function of the tuple e. A crucial step in the proof is to express f_e as a product of indicator functions. In this case we consider the following functions $g_{i_j} : \mathbb{Z}_m \to \mathbb{Z}_m$ for $1 \leq j \leq r$. Define $g_{i_1} : \mathbb{Z}_m \to \mathbb{Z}_m$ such that for any $y \in \mathbb{Z}_m$, $g_{i_1}(y) = b$ if $y = e_1$, and 0 otherwise. For any $2 \leq j \leq r$, define $g_{i_j} : \mathbb{Z}_m \to \mathbb{Z}_m$ as $g_{i_j}(y) = 1$ if $y = e_i$, and 0 otherwise. By definition, we have $f_e(x_1, \ldots, x_r) = g_{i_1}(x_1) g_{i_2}(x_2) \cdots g_{i_r}(x_r)$. Thus we get the following generalization of Lemma 42.

Lemma 44. *For any $m \in \mathbb{N}$ and $1 \leq r \leq n' - 3$, let τ be an r-function such that for any $\langle a_1, \ldots, a_{n'} \rangle \in \mathbb{Z}_m^{n'}$,*

$$\tau\left(\langle a_1, \ldots, a_{n'} \rangle\right) = \langle a_1, \ldots, a_{j-1}, a_j + f_e(a_{i_1}, \ldots, a_{i_r}), a_{j+1}, \ldots, a_{n'} \rangle$$

where the function $f_e : \mathbb{Z}_m^r \to \mathbb{Z}_m$ is defined by: $f_e(x) = b$ if $x = e$; and 0 otherwise. Then there exists a set of 2-functions $\tau_1, \ldots, \tau_{k_r}$ such that $\tau = \tau_{k_r} \circ \cdots \circ \tau_1$, and $k_r \leq 4r^2 - 3$.

Comment on decomposition of general r-functions for $3 \leq r \leq n' - 3$: Any function $f : \mathbb{Z}_m^r \to \mathbb{Z}_m$ can be expressed as $f = \sum_e c_e \cdot \chi_e$, where χ_e is the characteristic function of the tuple $e \in \mathbb{Z}_m^r$, and $c_e \in \mathbb{Z}_m$. Thus Lemma 44 suffices to argue that any r-function can be decomposed into a set of 2-functions. However, the implied bound on k_r may not be small. In particular, the number of tuples where f takes non-zero value might be large.

It remains to decompose $\alpha_{n'-1}$ and $\alpha_{n'}$. The following lemma about cycles that intersect at exactly one point serves as a key tool in our decomposition.

Lemma 45. *Suppose there are two cycles, $\sigma = (t, a_1, \cdots, a_{\ell-1})$ and $\tau = (t, b_1, \cdots, b_{\ell-1})$
, of length $\ell \geq 2$ such that $a_i \neq b_j$ for every $i, j \in [\ell - 1]$. Then, $(\sigma \circ \tau)^\ell \circ (\tau \circ \sigma)^\ell = \sigma^2$.*

Proof. By Proposition 56, we have

$$\beta := \tau \circ \sigma = (t, a_1, \cdots, a_{\ell-1}, b_1, \cdots, b_{\ell-1}), \text{ and}$$
$$\gamma := \sigma \circ \tau = (t, b_1, \cdots, b_{\ell-1}, a_1, \cdots, a_{\ell-1}).$$

Both β and γ are cycles of length $2\ell - 1$. Also note that $2\ell - 1$ is co-prime with ℓ. Thus both β^ℓ and γ^ℓ are also cycles of length $2\ell - 1$ given as follow:

$$\begin{aligned}\beta^\ell &= (t, b_1, a_1, b_2, a_2, \cdots, b_{\ell-1}, a_{\ell-1}), \text{ and}\\ \gamma^\ell &= (t, a_1, b_1, a_2, b_2, \cdots, a_{\ell-1}, b_{\ell-1}).\end{aligned}$$

Now by Proposition 57,

$$\begin{aligned}\sigma \circ \beta^\ell \circ \sigma^{-1} &= (\sigma(t), \sigma(b_1), \sigma(a_1), \cdots, \sigma(a_{\ell-2}), \sigma(b_{\ell-1}), \sigma(a_{\ell-1}))\\ &= (a_1, b_1, a_2, b_2, \cdots, a_{\ell-1}, b_{\ell-1}, t) = \gamma^\ell.\end{aligned}$$

Therefore,

$$\begin{aligned}\beta^\ell \circ \sigma^{-1} &= (t, a_{\ell-1}, a_{\ell-2}, \cdots, a_1) \circ (t, a_1, b_1, a_2, b_2, \cdots, a_{\ell-1}, b_{\ell-1})\\ &= (a_1, b_1)(a_2, b_2)\cdots(a_{\ell-1}, b_{\ell-1}).\end{aligned}$$

It is thus evident that $\left(\beta^\ell \circ \sigma^{-1}\right)^2$ is the identity permutation. Hence,

$$\begin{aligned}\gamma^\ell \circ \beta^\ell &= \sigma \circ \beta^\ell \circ \sigma^{-1} \circ \beta^\ell\\ &= \sigma \circ \beta^\ell \circ \sigma^{-1} \circ \beta^\ell \circ \sigma^{-1} \circ \sigma\\ &= \sigma^2.\end{aligned}$$

□

Before going into the detailed description of the decomposition procedure, let us briefly discuss the main idea. Here we first consider $\alpha_{n'}$. The case of $\alpha_{n'-1}$ will be analogous. Recall that $\alpha_{n'} = (\langle 00\cdots00\rangle, \langle 00\cdots01\rangle, \langle 00\cdots02\rangle, \cdots, \langle 00\cdots0(m-1)\rangle)$ is a cycle of length m. For $a = (m+1)/2$, we define $\sigma = (\langle 00\cdots0(0\cdot a)\rangle, \langle 00\cdots0(1\cdot a)\rangle, \langle 00\cdots0(2\cdot a)\rangle, \cdots, \langle 00\cdots0((m-1)\cdot a)\rangle)$, and $\tau = (\langle (0\cdot a)00\cdots0\rangle, \langle (1\cdot a)00\cdots0\rangle, \langle (2\cdot a)00\cdots0\rangle, \cdots, \langle ((m-1)\cdot a)00\cdots0\rangle)$, where the multiplication is in $\mathbb{Z}_m$. Since m is co-prime with $(m+1)/2$, σ and τ are cycles of length m. (Here we use the fact that m is odd.) Observe that $\sigma^2 = \alpha_{n'}$, so by applying Lemma 45 to σ and τ we get $\alpha_{n'}$. It might seem we didn't make much progress towards decomposition, as now instead of one $(n'-1)$-function $\alpha_{n'}$ we have to decompose two $(n'-1)$-functions σ and τ. However, we will not decompose σ and τ directly, but rather we obtain a decomposition for $(\sigma\circ\tau)^m$ and $(\tau\circ\sigma)^m$. Surprisingly this can be done using Lemma 44 although indirectly.

We consider an $(n'-3)$-function σ' whose cycle decomposition contains σ as one of its cycles. Similarly we consider a 3-function τ' whose cycle decomposition contains τ as one of its cycles. We carefully choose these σ' and τ' such that $(\sigma'\circ\tau')^m = (\sigma\circ\tau)^m$ and $(\tau'\circ\sigma')^m = (\tau\circ\sigma)^m$. We will use Lemma 44 to directly decompose σ' and τ' to get the desired decomposition.

Lemma 46. *For any $n'-1 \le i \le n'$, let α_i be the permutation over $\mathbb{Z}_m^{n'}$ given by Definition 60 where m is odd. Then, there exists a set of 2-functions $\alpha_{i,1}, \ldots, \alpha_{i,k_i}$ such that $\alpha_i = \alpha_{i,k_i} \circ \cdots \circ \alpha_{i,1}$, and $k_i = O(m\cdot(i-1)^2)$.*

Proof. For the sake of brevity, we will only describe the procedure to decompose $\alpha_{n'}$ into a set of 2-functions. The decomposition of $\alpha_{n'-1}$ is completely analogous. (We comment on this more at the end of the proof.)

Let σ be the following permutation: for any $\langle a_1, \ldots, a_{n'} \rangle \in \mathbb{Z}_m^{n'}$,

$$\sigma\left(\langle a_1, \ldots, a_{n'} \rangle\right) = \langle a_1, \ldots, a_{n'-1}, a_{n'} + f(a_1, \ldots, a_{n'-1}) \rangle$$

where the function $f : \mathbb{Z}_m^{n'-1} \to \mathbb{Z}_m$ is defined as follows:

$$f(x) = \begin{cases} (m+1)/2 & \text{if } x = \langle 0, \ldots, 0 \rangle, \\ 0 & \text{otherwise.} \end{cases}$$

Note that $(m+1)/2$ is well defined because m is odd. Further, since m and $(m+1)/2$ are co-prime, σ is a m length cycle. Moreover, $\sigma^2 = \alpha_{n'}$. The description of σ uses crucially that m is odd. If m were not odd, then the description fails. Indeed finding a substitute for σ is the main hurdle that needs to be addressed to handle the case when m is even.

We also consider another permutation τ such that for any $\langle a_1, \ldots, a_{n'} \rangle \in \mathbb{Z}_m^{n'}$,

$$\tau\left(\langle a_1, \ldots, a_{n'} \rangle\right) = \langle a_1 + f(a_2, \ldots, a_{n'}), a_2, \ldots, a_{n'} \rangle$$

where the function f is the same as in the definition of σ. So τ is also a cycle of length m. Let $t = (m+1)/2$. Then the cycle decomposition of σ and τ are,

$$\begin{aligned} \sigma &= \left(\langle 0, \ldots, 0, 0 \cdot t \rangle, \langle 0, \ldots, 0, 1 \cdot t \rangle, \langle 0, \ldots, 0, 2 \cdot t \rangle, \cdots, \langle 0, \ldots, 0, (m-1) \cdot t \rangle\right), \\ \tau &= \left(\langle 0 \cdot t, 0, \ldots, 0 \rangle, \langle 1 \cdot t, 0, \ldots, 0 \rangle, \langle 2 \cdot t, 0, \ldots, 0 \rangle, \cdots, \langle (m-1) \cdot t, 0, \ldots, 0 \rangle\right) \end{aligned}$$

where the multiplication is in $\mathbb{Z}_m$. Observe that $\langle 0, \ldots, 0 \rangle$ is the only common element involved in both cycles σ and τ. Therefore, by Lemma 45,

$$(\sigma \circ \tau)^m \circ (\tau \circ \sigma)^m = \sigma^2 = \alpha_{n'}.$$

We note that both σ and τ are $(n'-1)$-functions. Thus so far it is not clear how the above identity helps us to decompose $\alpha_{n'}$. We now define two more permutations σ' and τ' such that they are themselves decomposable into 2-functions and, moreover, $(\sigma' \circ \tau')^m = (\sigma \circ \tau)^m$ and $(\tau' \circ \sigma')^m = (\tau \circ \sigma)^m$.

The permutations σ' and τ' are defined as follows: for any $\langle a_1, \ldots, a_{n'} \rangle \in \mathbb{Z}_m^{n'}$,

$$\begin{aligned} \sigma'\left(\langle a_1, \ldots, a_{n'} \rangle\right) &= \langle a_1, \ldots, a_{n'-1}, a_{n'} + g(a_1, \ldots, a_{n'-3}) \rangle, \text{ and} \\ \tau'\left(\langle a_1, \ldots, a_{n'} \rangle\right) &= \langle a_1 + h(a_{n'-2}, a_{n'-1}, a_{n'}), a_2, \ldots, a_{n'} \rangle \end{aligned}$$

where $g : \mathbb{Z}_m^{n'-3} \to \mathbb{Z}_m$ is the following map:

$$g(x) = \begin{cases} (m+1)/2 & \text{if } x = \langle 0, \ldots, 0 \rangle, \\ 0 & \text{otherwise,} \end{cases}$$

and $h : \mathbb{Z}_m^3 \to \mathbb{Z}_m$ is similarly defined but on lower dimension:

$$h(x) = \begin{cases} (m+1)/2 & \text{if } x = \langle 0, 0, 0 \rangle, \\ 0 & \text{otherwise.} \end{cases}$$

Since m and $(m+1)/2$ are co-prime, σ' is composed of m^2 disjoint cycles, each of length m. Similarly, τ' is composed of $m^{(n'-4)}$ disjoint cycles, each of length

m. Moreover, σ is one of the cycles among m^2 disjoint cycles of σ' and τ is one of the cycles among $m^{(n'-4)}$ disjoint cycles of τ'. So, we can write

$$\sigma' = \sigma \circ C_1 \circ \cdots \circ C_{m^2-1}, \text{ and}$$
$$\tau' = \tau \circ C'_1 \circ \cdots \circ C'_{m^{(n'-4)}-1}$$

where each of C_i's and C'_j's is a m length cycle. An important fact regarding σ' and τ' is that the *only* element moved by both is the all-zeros tuple $\langle 0, \ldots, 0\rangle$. This is easily seen from their definitions. Recall we had observed that the all-zeroes is, in fact, moved by σ and τ. In other words, cycles in the set $\{C_1, \ldots, C_{m^2-1}, C'_1, \ldots, C'_{m^{(n'-4)}-1}\}$ are mutually disjoint, as well as they are disjoint from σ and τ. Thus, using the fact that C_i's and C'_j's are m length cycles, we have

$$(\tau' \circ \sigma')^m = (\tau \circ \sigma)^m, \text{ and}$$
$$(\sigma' \circ \tau')^m = (\sigma \circ \tau)^m.$$

Therefore, we can express $\alpha_{n'}$ in terms of σ' and τ',

$$\alpha_{n'} = \sigma^2 = (\sigma \circ \tau)^m \circ (\tau \circ \sigma)^m = (\sigma' \circ \tau')^m \circ (\tau' \circ \sigma')^m.$$

But, by definition, σ' and τ' are an $(n'-3)$-function and a 3-function, respectively. Furthermore, they satisfy the requirement of Lemma 44. Hence both σ' and τ' can be decomposed into a set of 2-functions. As a result we obtain a decomposition of $\alpha_{n'}$ into a set of $k_{n'}$ many 2-functions, where $k_{n'} \leq 2m\left(4(n'-3)^2 - 3 + 4 \cdot 3^2 - 3\right) = m\left(8(n'-3)^2 + 60\right) \leq 60 \cdot m\left(n'-3\right)^2$.

The permutation $\alpha_{n'-1}$ can be decomposed in a similar fashion. Note that $\alpha_{n'-1}$ is composed of m disjoint m length cycles. Each of the m disjoint cycles can be decomposed using the procedure described above. If we do so, we get the length $k_{n'-1} = O(m^2 \cdot (n'-3)^2)$, which would suffice for our purpose. However, we can improve this bound to $O(m \cdot (n'-2)^2)$ by a slight modification of σ, τ, σ' and τ'. Below we define these permutations and omit the details of the proof as it is analogous to the proof above. The argument should be carried with σ, τ, σ' and τ' defined as follows: for any $\langle a_1, \ldots, a_{n'}\rangle \in \mathbb{Z}_m^{n'}$,

$$\sigma\left(\langle a_1, \ldots, a_{n'}\rangle\right) = \langle a_1, \ldots, a_{n'-2}, a_{n'-1} + f(a_1, \ldots, a_{n'-2}), a_{n'}\rangle,$$
$$\tau\left(\langle a_1, \ldots, a_{n'}\rangle\right) = \langle a_1 + f(a_2, \ldots, a_{n'-1}), \ldots, a_{n'-1}, a_{n'}\rangle,$$
$$\sigma'\left(\langle a_1, \ldots, a_{n'}\rangle\right) = \langle a_1, \ldots, a_{n'-2}, a_{n'-1} + g(a_1, \ldots, a_{n'-3}), a_{n'}\rangle, \text{ and}$$
$$\tau'\left(\langle a_1, \ldots, a_{n'}\rangle\right) = \langle a_1 + h(a_{n'-2}, a_{n'-1}), a_2, \ldots, a_{n'-1}, a_{n'}\rangle,$$

where the function $f : \mathbb{Z}_m^{n'-2} \to \mathbb{Z}_m$ is defined as: $f(x) = (m+1)/2$ if $x = \langle 0, \ldots, 0\rangle$; otherwise $f(x) = 0$, the function $g : \mathbb{Z}_m^{n'-3} \to \mathbb{Z}_m$ is defined as: $g(x) = (m+1)/2$ if $x = \langle 0, \ldots, 0\rangle$; otherwise $g(x) = 0$ and the function $h : \mathbb{Z}_m^2 \to \mathbb{Z}_m$ is defined as: $h(x) = (m+1)/2$ if $x = \langle 0, 0\rangle$; otherwise $h(x) = 0$. The only difference is now that both σ and τ are composed of m disjoint cycles of length m, instead of just one cycle as in case of $\alpha_{n'}$. However every cycle of σ has non-empty intersection with exactly one cycle of τ, and furthermore, the intersection is singleton. Hence we can still apply Lemma 45 with m pairs of m length cycles, where each pair consists of one cycle from σ and another from τ such that they have non-empty intersection. □

Step 3: Invocation of Lemma 38

To finish the construction we just replace α_i's in $\alpha = \alpha_{n'} \circ \cdots \circ \alpha_1$ by $\alpha_{i,k_i} \circ \cdots \circ \alpha_{i,1}$. Take the resulting sequence as $\sigma = \sigma_k \circ \cdots \circ \sigma_1$, where $k = \sum_{i=1}^{n'} k_i \leq c_3 \cdot m \cdot n'^3$ for some constant $c_3 > 0$. Now we apply Lemma 38 to get a space-optimal counter over $\mathbb{Z}_m^n$. In Lemma 38, we set $k' = m^{r'}$ such that r' is the smallest integer for which $m^{r'} \geq k$ and set $\mathcal{D}' = \mathbb{Z}_m^{r'}$. Hence $r' \leq \log_m k + 1 \leq 3 \log_m n + c$, for some constant $c > 1$. Since each of σ_i's is a 2-function, by Claim 39 it can be implemented using a decision assignment tree T_i such that $\mathrm{READ}(T_i) = 3$ and $\mathrm{WRITE}(T_i) = 1$. In Lemma 38, we use the standard space-optimal Gray code over $\mathcal{D}' = \mathbb{Z}_m^{r'}$ as C'. The code C' can be implemented using a decision assignment tree T' with $\mathrm{READ}(T') = r'$ and $\mathrm{WRITE}(T') = 1$ (Theorem 54). Hence the final counter implied from Lemma 38 can be computed using a decision assignment tree T with $\mathrm{READ}(T) \leq 4 \log_m n$ and $\mathrm{WRITE}(T) = 2$. This completes our required construction.

9. Quasi-Gray Codes Over Even Sized Alphabets

In this chapter we describe how to construct quasi-Gray code over $\mathbb{Z}_m^n$ for any even m. We start with binary alphabet.

9.1 Quasi-Gray Codes over Binary Alphabet

9.1.1 Counters via Linear Transformation

The construction for binary alphabet is based on linear transformations. Consider the vector space $\mathbb{F}_q^n$, and let $L : \mathbb{F}_q^n \to \mathbb{F}_q^n$ be a linear transformation. A basic fact in linear algebra says that if L has *full* rank, then the mapping given by L is a bijection. Thus, when L is full rank, the mapping can also be thought of as a permutation over $\mathbb{F}_q^n$. Throughout this chapter we use many basic terms related to linear transformation without defining them, for the details of which we refer the reader to any standard text book on linear algebra (e.g. [Lan87]).

A natural way to build counter out of a full rank linear transformation is to fix a starting element, and repeatedly apply the linear transformation to obtain the next element. Clearly this only list out elements in the cycle containing the starting element. Therefore, we would like to choose the starting element such that we enumerate the largest cycle. Ideally, we would like the largest cycle to contain all the elements of $\mathbb{F}_q^n$. However this is not possible because any linear transformation fixes the all-zero vector. But there do exist full rank linear transformations such that the permutation given by them is a single cycle of length $q^n - 1$. Such a linear transformation would give us a counter over a domain of size q^n that enumerates all but one element. Clearly, a trivial implementation of the aforementioned argument would lead to a counter that reads and writes all n coordinates in the worst-case. In the rest of this chapter, we will develop an implementation and argue about the choice of linear transformation such that the read and write complexity decreases exponentially.

We start with recalling some basic facts from linear algebra.

Definition 61 (Linear Transformation)**.** *A map $L : \mathbb{F}_q^n \to \mathbb{F}_q^n$ is called a linear transformation if $L(c \cdot x + y) = cL(x) + L(y)$, for all $x, y \in \mathbb{F}_q^n$ and $c \in \mathbb{F}_q$.*

It is well known that every linear transformation L is associated with some matrix $A \in \mathbb{F}_q^{n \times n}$ such that applying the linear transformation is equivalent to the left multiplication by A. That is, $L(x) = Ax$ where we interpret x as a column vector. Furthermore, L has full rank iff A is invertible over $\mathbb{F}_q$.

Definition 62 (Elementary matrices)**.** *An $n \times n$ matrix over a field $\mathbb{F}$ is said to be an* elementary matrix *if it has one of the following forms:*

(a) The off-diagonal entries are all 0. For some $i \in [n]$, (i, i)-th entry is a non-zero $c \in \mathbb{F}$. Rest of the diagonal entries are 1. For a fixed i, we denote all matrices of this type by $E_{i,i}$. (See Fig. 9.1.)

(b) The diagonal entries are all 1. For some i and j, $1 \leq i \neq j \leq n$, (i,j)-th entry is a non-zero $c \in \mathbb{F}$. Rest of the off-diagonal entries are 0. For each i and j, $i \neq j$, we denote all matrices of this type by $E_{i,j}$. (See Fig. 9.1.)

$$(a) \quad \begin{array}{c} \\ i \\ \\ \\ \\ \\ \\ \end{array} \begin{array}{c} \begin{array}{ccccccc} & & & i & & & \end{array} \\ \begin{pmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & 1 & & & & \\ & & & c & & & \\ & & & & 1 & & \\ & & & & & \ddots & \\ & & & & & & 1 \end{pmatrix} \end{array} \quad \text{or,} \quad (b) \quad \begin{array}{c} \\ \\ \\ \\ i \\ \\ \\ \end{array} \begin{array}{c} \begin{array}{ccccccc} & & & j & & & \end{array} \\ \begin{pmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & 1 & & & & \\ & & & \ddots & & & \\ & & c & & 1 & & \\ & & & & & \ddots & \\ & & & & & & 1 \end{pmatrix} \end{array}$$

Figure 9.1: Elementary matrices

From the definition it is easy to see that left multiplication by an elementary matrix of type $E_{i,i}$ is equivalent to multiplying the i-th row with c, and by an elementary matrix of type $E_{i,j}$ is equivalent to adding c times j-th row to the i-th row.

Proposition 63. *Let $A \in \mathbb{F}^{n \times n}$ be invertible. Then A can be written as a product of k elementary matrices such that $k \leq n^2 + 4(n-1)$.*

Proof. Consider the inverse matrix A^{-1} which is also full rank. It is easily seen from Gauss elimination that by left multiplying A^{-1} with at most n^2 many elementary matrices, we can reduce A^{-1} to a permutation matrix. A permutation matrix is a $\{0,1\}$-matrix that has exactly one 1 in each row and column. Now we need at most $(n-1)$ row swaps to further reduce the obtained permutation matrix to the identity matrix. We claim that a row swap can be implemented by left multiplication with at most 4 elementary matrices. Indeed, to swap row i and row j, the following sequence of operation suffices: (i) add j-th row to i-th row, (ii) subtract i-th row from j-th row, (iii) add j-th row to i-th row, and (iv) multiply j-th row with -1. (The last operation is not required if the characteristic of the underlying field is 2.)

Hence, the inverse of A^{-1} which is our original matrix A is the product of k elementary matrices. □

9.1.2 Construction of the counter

Let A be a full rank linear transformation from $\mathbb{F}_q^n \to \mathbb{F}_q^n$ such that when viewed as permutation it is a single cycle of length $q^n - 1$. More specifically, A is an invertible matrix in $\mathbb{F}_q^{n \times n}$ such that for any $x \in \mathbb{F}_q^n$ where $x \neq (0, \ldots, 0)$, $Ax, A^2x, \ldots, A^{(q^n-1)}x$ are distinct. Such a matrix exists, for example, take A to be the matrix of a linear transformation that corresponds to multiplication from left by a fixed generator of the multiplicative group of $\mathbb{F}_{q^n}$ under the standard vector representation of elements of $\mathbb{F}_{q^n}$. Let $A = E_k E_{k-1} \cdots E_1$ where E_i's are elementary matrices.

Theorem 64. *Let q, A, and k be as defined above. Let $r \geq \log_q k$. There exists a quasi-Gray code on the domain $(\mathbb{F}_q)^{n+r}$ of length $q^{n+r} - q^r$ that can be implemented using a decision assignment tree T such that* $\mathrm{READ}(T) \leq r+2$ *and* $\mathrm{WRITE}(T) \leq 2$.

Proof. The proof follows readily from Lemma 38, where E_i's play the role of σ_i's, and noting that the permutation given by any elementary matrix can be implemented using a decision assignment tree that reads at most two coordinates and writes at most one. For the counter C' on $(\mathbb{F}_q)^r$ we chose a Gray code of trivial read complexity r and write complexity 1. □

Thus, we obtain a counter on a domain of size roughly kq^n that misses at most qk elements. Clearly, we would like to minimize k. A trivial bound on k is $O(n^2)$ that follows from Proposition 63. We now discuss the choice of A so that k becomes $O(n)$. We start with recalling a notion of primitive polynomials over finite fields.

Definition 65 (Primitive polynomials)**.** *A* primitive *polynomial $p(z) \in \mathbb{F}_q[z]$ of degree n is a monic irreducible polynomial over $\mathbb{F}_q$ such that any root of it in $\mathbb{F}_{q^n}$ generates the multiplicative group of $\mathbb{F}_{q^n}$.*

Theorem 66 ([LN96])**.** *The number of primitive polynomials of degree n over $\mathbb{F}_q$ equals $\phi(q^n - 1)/n$, where $\phi(\cdot)$ is the Euler ϕ-function.*

Let $p(z)$ be a primitive polynomial of degree n over $\mathbb{F}_q$. The elements of $\mathbb{F}_{q^n}$ can be uniquely expressed as a polynomial in z over $\mathbb{F}_q$ of degree at most $n-1$. In particular, we can identify an element of $\mathbb{F}_{q^n}$ with a vector in $\mathbb{F}_q^n$ that is given by the coefficient vector of the unique polynomial expression of degree at most $n-1$. But, since $p(z)$ is primitive, we also know that $\mathbb{F}_{q^n} = \{0, 1, z, z^2, \ldots, z^{q^n-2}\}$. This suggests a particularly nice linear transformation to consider: *left multiplication by z*. This is so because the matrix A of the linear transformation computing the multiplication by z is very *sparse*. In particular, if $p(z) = z^n + c_{n-1}z^{n-1} + c_{n-2}z^{n-2} + \cdots + c_1 z + c_0$, then A looks as follows:

$$\begin{pmatrix} -c_{n-1} & 1 & 0 & \cdots & 0 & 0 & 0 \\ -c_{n-2} & 0 & 1 & \cdots & 0 & 0 & 0 \\ -c_{n-3} & 0 & 0 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ -c_2 & 0 & 0 & \cdots & 0 & 1 & 0 \\ -c_1 & 0 & 0 & \cdots & 0 & 0 & 1 \\ -c_0 & 0 & 0 & \cdots & 0 & 0 & 0 \end{pmatrix}.$$

Thus, from the proof of Proposition 63, it follows that A can be written as a product of at most $n + 4(n-1)$ elementary matrices. (When q is a power of 2, then the number of elementary matrices in the product is at most $n + 3(n-1)$.) Hence, from the discussion above and using Theorem 64, we obtain the following corollaries. Setting $r = \lceil \log(4n-3) \rceil$ in Theorem 64 gives:

Corollary 67. *For any $n' \geq 2$, and $n = n' + \lceil \log(4n'-3) \rceil$, there exists a counter on $(\mathbb{Z}_2)^n$ that misses at most $8n$ strings and can be implemented by a decision assignment tree that reads at most $4 + \log n$ bits and writes at most 2 bits.*

By doubling the number of missed strings and increasing the number of read bits by one we can construct given counters for any $\mathbb{Z}_2^n$, where $n \geq 15$. In the above corollary the number of missed strings grows linearly with n. One might wonder if it is possible to keep the number of missing strings $o(n)$, while keeping the read complexity essentially the same. The next corollary shows that this is indeed possible, but at the cost of increasing the write complexity.

Corollary 68. *For $n \geq 2$, there exists a counter on $(\mathbb{Z}_2)^{n+O(\log n)}$ that misses out at most $O(n/\lceil \log n \rceil)$ strings. Furthermore, it can be implemented by a decision assignment tree that reads and writes at most $O(\log n)$ bits.*

Proof. The idea is simply to increase the underlying alphabet size. In particular, let $q = 2^{\lceil \log n \rceil}$ in Theorem 64. □

We also remark that by taking q to be $2^{\frac{n}{C}}$, where $C > 1$ is a universal constant, one would get a counter on $(\mathbb{Z}_2)^{n+O(1)}$ that misses only $O(C)$ strings (i.e. independent of n). However, the read and write complexity gets worse. They are at most $2\left(\frac{n}{C}\right) + O(1)$.

For the general case, when q is a prime power, we obtain the following corollary by setting r to $\lceil \log_q(5n-4) \rceil$ or $1 + \lceil \log_q(5n-4) \rceil$ in Theorem 64.

Corollary 69 (Generalization of Theorem 51)**.** *Let q be any prime power. For $n \geq 15$, there exists a counter on $\mathbb{Z}_q^n$ that misses at most $5q^2n$ strings and that is computed by a decision assignment tree with read complexity at most $6 + \log_q n$ and write complexity 2.*

9.2 Getting counters for Even m

We can combine the results from Theorem 50 and Theorem 51 to get a counter over $\mathbb{Z}_m^n$ for any even m. We have already mentioned in Chapter 7 that if we have space-optimal quasi-Gray codes over the alphabet $\mathbb{Z}_2$ and $\mathbb{Z}_o$ with o being odd then we can get a space-optimal quasi-Gray code over the alphabet $\mathbb{Z}_m$ for any even m. Unfortunately in Section 9.1.1, instead of space-optimal counters we were only able to generate a counter over binary alphabet that misses $O(n)$ many strings. As a consequence we cannot directly apply Theorem 55. The problem is following. Suppose $m = 2^\ell o$ for some $\ell > 0$ and odd o. By the argument used in the proof of Lemma 37 we know that there is a counter over $(\mathbb{Z}_{2^\ell})^{n-1}$ of the same length as that over $(\mathbb{Z}_2)^{\ell(n-1)}$. Furthermore the length of the counter is of the form $2^{O(\log n)}(2^{n'}-1)$, for some n' that depends on the value of ℓn (see the construction in Section 9.1.1). Now to apply Theorem 55 as in the proof of Lemma 36, $2^{n'}-1$ must be co-prime with o. However that may not always be the case. Nevertheless, we can choose the parameters in the construction given in Section 9.1.1 to make n' such that $2^{n'}-1$ is co-prime with o. This is always possible because of the following simple algebraic fact. In the following proposition we use the notation $\mathbb{Z}_o^*$ to denote the multiplicative group modulo o and $\mathtt{ord}_o(e)$ to denote the order of any element $e \in \mathbb{Z}_o^*$.

Proposition 70. *For any $n \in \mathbb{N}$ and odd $o \in \mathbb{N}$, consider the set $\mathcal{S} = \{n, n+1, \cdots, n + \mathtt{ord}_o(2) - 1\}$. Then there always exists an element $n' \in \mathcal{S}$ such that $2^{n'}-1$ is co-prime to o.*

Proof. Inside $\mathbb{Z}_o^*$, consider the cyclic subgroup G generated by 2, i.e., $G = \{1, 2, \cdots, 2^{\mathtt{ord}_o(2)}\}$. Clearly, $\{2^s \pmod o \mid s \in \mathcal{S}\} = G$. Hence there exists an element $n' \in \mathcal{S}$ such that $2^{n'} - 1 \pmod o = 1$. It is clear that if $\gcd(2^{n'} - 1, o) = 1$ then we are done. Now the proposition follows from the following easily verifiable fact: for any $a, b, c \in \mathbb{N}$, if $a \equiv b \pmod c$ then $\gcd(a, c) = \gcd(b, c)$. □

So for any $n \in \mathbb{N}$, in the proof of Lemma 36 we take the first coordinate to be $\mathbb{Z}_m^i$ for some suitably chosen $i \geq 1$ instead of just $\mathbb{Z}_m$. The choice of i will be such that the length of the counter over $(\mathbb{Z}_{2^\ell})^{n-i}$ will become co-prime with o. The above proposition guarantees the existence of such an $i \in [\mathtt{ord}_o(2)]$. Hence we can conclude the following.

Theorem 71. *For any even $m \in \mathbb{N}$ so that $m = 2^\ell o$ where o is odd, there is a quasi-Gray code C over $\mathbb{Z}_m^n$ of length at least $m^n - O(no^n)$, that can be implemented using a decision assignment tree which reads at most $O(\log_m n + \mathtt{ord}_o(2))$ coordinates and writes at most 3 coordinates.*

9.3 Bibliographical Notes

We comment that the algorithm given for constructing codes over binary alphabets can be made uniform. To achieve that we need to obtain a primitive polynomial of degree n over $\mathbb{F}_q$ uniformly. To this end, we can use a number of algorithms (deterministic or probabilistic); for example, [Sho92, Shpb, Shp96]. For a thorough treatment, we refer to Chapter 2 in [Shpa].

10. Conclusion

In our construction of quasi-Gray codes we have only talked about implementing $next(\cdot,\cdot)$ function for any counter. However we can similarly define complexity of computing $prev(\cdot,\cdot)$ function in the decision assignment tree model. We would like to emphasize that all our techniques can also be carried over to do that. To extend the result of Section 9.1.1 of Chapter 9, we take the inverse of the linear transformation matrix A and follow exactly the same proof to get the implementation of $prev(\cdot,\cdot)$. As a consequence we achieve exactly the same bound on read and write complexity. Now for the quasi-Gray code over $\mathbb{Z}_m$ for any odd m, instead of α in the Step 1 (in of Chapter 8), we simply consider the α^{-1} which is equal to $\alpha_1^{-1} \circ \alpha_2^{-1} \circ \cdots \circ \alpha_{n'}^{-1}$. Then we follow an analogous technique to decompose α_i^{-1}'s and finally invoke Lemma 38. Thus we get the same bound on read and write complexity.

Open problem. Our work clearly provides a hardness separation of quasi-Gray codes over odd and even sized alphabet domain. The fact that odd permutations are hard to generate seems to be the real bottleneck and for binary Gray-code the best read-complexity upper bound known so far is $n-1$ [Fre16]. It is believed that, this is probably the best that can be achieved. Hence, improving Raskin's lower bound from $n/2$ to $n-1$ is a nice open question.

Bibliography

[AB16] Amir Abboud and Greg Bodwin. The 4/3 additive spanner exponent is tight. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016*, pages 351–361, 2016.

[AB17] Amir Abboud and Arturs Backurs. Towards hardness of approximation for polynomial time problems. In *8th Innovations in Theoretical Computer Science Conference, ITCS 2017*, pages 11:1–11:26, 2017.

[ABP17] Amir Abboud, Greg Bodwin, and Seth Pettie. A hierarchy of lower bounds for sublinear additive spanners. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017*, pages 568–576, 2017.

[Abr87] Karl Abrahamson. Generalized string matching. *SIAM J. Comput.*, 16(6):1039–1051, 1987.

[ABW15] Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight hardness results for LCS and other sequence similarity measures. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015*, pages 59–78, 2015.

[ACIM99] Donald Aingworth, Chandra Chekuri, Piotr Indyk, and Rajeev Motwani. Fast estimation of diameter and shortest paths (without matrix multiplication). *SIAM J. Comput.*, 28(4):1167–1181, 1999.

[ADD+93] Ingo Althöfer, Gautam Das, David P. Dobkin, Deborah Joseph, and José Soares. On sparse spanners of weighted graphs. *Discrete & Computational Geometry*, 9:81–100, 1993.

[AHWW16] Amir Abboud, Thomas Dueholm Hansen, Virginia Vassilevska Williams, and Ryan Williams. Simulating branching programs with edit distance and friends: or: a polylog shaved is a lower bound made. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016*, pages 375–388, 2016.

[AKO10] Alexandr Andoni, Robert Krauthgamer, and Krzysztof Onak. Polylogarithmic approximation for edit distance and the asymmetric query complexity. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010*, pages 377–386, 2010.

[AMS12] Noga Alon, Ankur Moitra, and Benny Sudakov. Nearly complete graphs decomposable into large induced matchings and their applications. In *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012*, pages 1079–1090, 2012.

[AN10] Alexandr Andoni and Huy L. Nguyen. Near-optimal sublinear time algorithms for Ulam distance. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010*, pages 76–86, 2010.

[Ang76] Dana Angluin. The four russians' algorithm for boolean matrix multiplication is optimal in its class. In *ACM SIGACT News*, pages 19–33, 1976.

[AO09] Alexandr Andoni and Krzysztof Onak. Approximating edit distance in near-linear time. In *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing*, STOC '09, pages 199–204, 2009.

[AR18] Amir Abboud and Aviad Rubinstein. Fast and deterministic constant factor approximation algorithms for LCS imply new circuit lower bounds. In *9th Innovations in Theoretical Computer Science Conference, ITCS 2018*, pages 35:1–35:14, 2018.

[ASD90] D. J. Amalraj, N. Sundararajan, and Goutam Dhar. Data structure based on Gray code encoding for graphics and image processing. In *Proceedings of the SPIE: International Society for Optical Engineering*, pages 65–76, 1990.

[BC92] Michael Ben-Or and Richard Cleve. Computing algebraic formulas using a constant number of registers. *SIAM J. Comput.*, 21(1):54–58, 1992.

[BCJ+10] Prosenjit Bose, Paz Carmi, Dana Jansens, Anil Maheshwari, Pat Morin, and Michiel H. M. Smid. Improved methods for generating quasi-Gray codes. In *Algorithm Theory - SWAT 2010, 12th Scandinavian Symposium and Workshops on Algorithm Theory*, pages 224–235, 2010.

[BCK+14] Harry Buhrman, Richard Cleve, Michal Koucký, Bruno Loff, and Florian Speelman. Computing with a full memory: catalytic space. In *Symposium on Theory of Computing, STOC 2014*, pages 857–866, 2014.

[BCPS15] Gilad Braunschvig, Shiri Chechik, David Peleg, and Adam Sealfon. Fault tolerant additive and (μ, α)-spanners. *Theor. Comput. Sci.*, 580:94–100, 2015.

[BCR16] Surender Baswana, Keerti Choudhary, and Liam Roditty. Fault tolerant subgraph for single source reachability: generic and optimal. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016*, pages 509–518, 2016.

[BCR17] Surender Baswana, Keerti Choudhary, and Liam Roditty. An efficient strongly connected components algorithm in the fault tolerant model. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017*, pages 72:1–72:15, 2017.

[BEG+18a] Mahdi Boroujeni, Soheil Ehsani, Mohammad Ghodsi, Mohammad Taghi Hajiaghayi, and Saeed Seddighin. Approximating edit distance in truly subquadratic time: Quantum and MapReduce. In

Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, pages 1170–1189, 2018.

[BEG+18b] Mahdi Boroujeni, Soheil Ehsani, Mohammad Ghodsi, Mohammad Taghi Hajiaghayi, and Saeed Seddighin. Approximating edit distance in truly subquadratic time: Quantum and MapReduce (extended version of [BEG+18a]). 2018.

[BEK+03] Tugkan Batu, Funda Ergün, Joe Kilian, Avner Magen, Sofya Raskhodnikova, Ronitt Rubinfeld, and Rahul Sami. A sublinear algorithm for weakly approximating edit distance. In *Proceedings of the Thirty-fifth Annual ACM Symposium on Theory of Computing*, STOC '03, pages 316–324, 2003.

[Bel58] Richard Bellman. On a routing problem. *Quart. Appl. Math.*, 16:87–90, 1958.

[BES06] Tuğkan Batu, Funda Ergun, and Cenk Sahinalp. Oblivious string embeddings and edit distance approximations. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm*, SODA '06, pages 792–801, 2006.

[BGG+15] Davide Bilò, Fabrizio Grandoni, Luciano Gualà, Stefano Leucci, and Guido Proietti. Improved purely additive fault-tolerant spanners. In *Algorithms - ESA 2015 - 23rd Annual European Symposium, Proceedings*, pages 167–178, 2015.

[BGPS14] Gerth Stølting Brodal, Mark Greve, Vineet Pandey, and Srinivasa Rao Satti. Integer representations towards efficient counting in the bit probe model. *J. Discrete Algorithms*, 26:34–44, 2014.

[BGPW17] Greg Bodwin, Fabrizio Grandoni, Merav Parter, and Virginia Vassilevska Williams. Preserving distances in very faulty graphs. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017*, pages 73:1–73:14, 2017.

[BI15] Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing*, STOC '15, pages 51–58, 2015.

[BK13] Surender Baswana and Neelesh Khanna. Approximate shortest paths avoiding a failed vertex: Near optimal data structures for undirected unweighted graphs. *Algorithmica*, 66(1):18–50, 2013.

[BK15] Karl Bringmann and Marvin Künnemann. Quadratic conditional lower bounds for string problems and dynamic time warping. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015*, pages 79–97, 2015.

[BKMP05] Surender Baswana, Telikepalli Kavitha, Kurt Mehlhorn, and Seth Pettie. New constructions of (alpha, beta)-spanners and purely additive spanners. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2005*, pages 672–681, 2005.

[BW12] Nikhil Bansal and Ryan Williams. Regularity lemmas and combinatorial algorithms. *Theory of Computing*, 8(1):69–94, 2012.

[BYTKK04] Ziv Bar-Yossef, Jayram S. Thathachar, Robert Krauthgamer, and Ravi Kumar. Approximating edit distance efficiently. In *Foundations of Computer Science, 2004. Proceedings. 45th Annual IEEE Symposium on*, pages 550–559, 2004.

[CCC92] C. C. Chang, H. Y. Chen, and C. Y. Chen. Symbolic Gray code as a data allocation scheme for two-disc systems. *The Computer Journal*, 35(3):299–305, 1992.

[CD18] Diptarka Chakraborty and Debarati Das. Sparse weight tolerant subgraph for single source shortest path. In *16th Scandinavian Symposium and Workshops on Algorithm Theory, SWAT 2018*, pages 15:1–15:15, 2018.

[CDG+18] Diptarka Chakraborty, Debarati Das, Elazar Goldenberg, Michal Koucký, and Michael E. Saks. Approximating edit distance within constant factor in truly sub-quadratic time. In *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018*, pages 979–990, 2018.

[CDK18] Diptarka Chakraborty, Debarati Das, and Michal Koucký. Approximate online pattern matching in sub-linear time. *CoRR*, abs/1810.03551, 2018.

[CDKS18] Diptarka Chakraborty, Debarati Das, Michal Koucký, and Nitin Saurabh. Space-optimal quasi-gray codes with logarithmic read complexity. In *26th Annual European Symposium on Algorithms, ESA 2018*, pages 12:1–12:15, 2018.

[CEPP08] Raphaël Clifford, Klim Efremenko, Benny Porat, and Ely Porat. A black box for online approximate pattern matching. In *Combinatorial Pattern Matching, 19th Annual Symposium, CPM 2008*, pages 143–151, 2008.

[CG75] Don Coppersmith and Edna Grossman. Generators for certain alternating groups with applications to cryptography. *SIAM J. Appl. Math.*, 29(4):624–627, 1975.

[CGK16] Diptarka Chakraborty, Elazar Goldenberg, and Michal Koucký. Streaming algorithms for embedding and computing edit distance in the low distance regime. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016*, pages 712–725, 2016.

[CGPR95] Maxime Crochemore, Leszek Gasieniec, Wojciech Plandowski, and Wojciech Rytter. Two-dimensional pattern matching in linear time and small space. In *STACS*, pages 181–192, 1995.

[CH98] Richard Cole and Ramesh Hariharan. Approximate string matching: a simpler faster algorithm. In *In Proc. ACM-SIAM SODA'98*, pages 463–472, 1998.

[Cha15] Timothy M. Chan. Speeding up the four russians algorithm by about one more logarithmic factor. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015*, pages 212–217, 2015.

[Cho16] Keerti Choudhary. An optimal dual fault tolerant reachability oracle. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016*, pages 130:1–130:13, 2016.

[CJS15] Raphaël Clifford, Markus Jalsenius, and Benjamin Sach. Cell-probe bounds for online edit distance and other pattern matching problems. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015*, pages 552–561, 2015.

[Cle89] Richard Cleve. *Methodologies for Designing Block Ciphers and Cryptographic Protocols*. PhD thesis, University of Toronto, April 1989.

[CLPR09] Shiri Chechik, Michael Langberg, David Peleg, and Liam Roditty. Fault-tolerant spanners for general graphs. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009*, pages 435–444, 2009.

[Coh63] Martin Cohn. Affine m-ary Gray codes. *Information and Control*, 6(1):70–78, 1963.

[Cro92] Maxime Crochemore. String-matching on ordered alphabets. *Theor. Comput. Sci.*, 92(1):33–47, 1992.

[CS90] Ming Syan Chen and Kang G. Shin. Subcube allocation and task migration in hypercube multiprocessors. *IEEE Transactions on Computers*, 39(9):1146–1155, 1990.

[CS09] Raphaël Clifford and Benjamin Sach. Online approximate matching with non-local distances. In *Combinatorial Pattern Matching, 20th Annual Symposium, CPM 2009*, pages 142–153, 2009.

[CS10] Raphaël Clifford and Benjamin Sach. Pseudo-realtime pattern matching: Closing the gap. In *Combinatorial Pattern Matching, 21st Annual Symposium, CPM 2010*, pages 101–111, 2010.

[CW90] Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. *J. Symb. Comput.*, 9(3):251–280, 1990.

[CZ04] Artur Czumaj and Hairong Zhao. Fault-tolerant geometric spanners. *Discrete & Computational Geometry*, 32(2):207–230, 2004.

[DF04] David S. Dummit and Richard M. Foote. *Abstract Algebra*. John Wiley & Sons, 2004.

[DK11] Michael Dinitz and Robert Krauthgamer. Fault-tolerant spanners: better and simpler. In *Proceedings of the 30th Annual ACM Symposium on Principles of Distributed Computing, PODC 2011*, pages 169–178, 2011.

[DKS18] Debarati Das, Michal Koucký, and Michael Saks. Lower bounds for combinatorial algorithms for boolean matrix multiplication. In *35th Symposium on Theoretical Aspects of Computer Science (STACS 2018)*, pages 23:1–23:14, 2018.

[DPS96] C. Ding, D. Pei, and A. Salomaa. *Chinese Remainder Theorem: Applications in Computing, Coding, Cryptography*. World Scientific Publishing Co., Inc., 1996.

[DTCR08] Camil Demetrescu, Mikkel Thorup, Rezaul Alam Chowdhury, and Vijaya Ramachandran. Oracles for distances avoiding a failed node or link. *SIAM J. Comput.*, 37(5):1299–1318, 2008.

[Erd] Paul Erdös. Extremal problems in graph theory. In *IN THEORY OF GRAPHS AND ITS APPLICATIONS, PROC. SYMPOS. SMOLENICE*, pages 29–36.

[FF] Lester R. Ford and Delbert Ray Fulkerson. *Flows in Networks*. Princeton University Press.

[Flo56] Ivan Flores. Reflected number systems. *IRE Transactions on Electronic Computers*, EC-5(2):79–82, 1956.

[FM71] Michael J Fischer and Albert R Meyer. Boolean matrix multiplication and transitive closure. pages 129–131, 1971.

[FMS97] Gudmund Skovbjerg Frandsen, Peter Bro Miltersen, and Sven Skyum. Dynamic word problems. *J. ACM*, 44(2):257–271, 1997.

[For56] Lester R. Ford Jr. Network Flow Theory. *Santa Monica, California: RAND Corporation*, pages P–923, 1956.

[Fre78] Michael L. Fredman. Observations on the complexity of generating quasi-Gray codes. *SIAM J. Comput.*, 7(2):134–146, 1978.

[Fre16] Zachary Frenette. Towards the efficient generation of Gray codes in the bitprobe model. Master's thesis, University of Waterloo, Waterloo, Ontario, Canada, 2016.

[Fur70] M. E. Furman. Application of a method of fast multiplication of matrices in the problem of finding the transitive closure of a graph. page 11(5):1252, 1970.

[Gal14] François Le Gall. Powers of tensors and fast matrix multiplication. In *International Symposium on Symbolic and Algebraic Computation, ISSAC 2014*, pages 296–303, 2014.

[GG88] Zvi Galil and Raffaeles Giancarlo. Data structures and algorithms for approximate string matching. *Journal of Complexity*, pages 4:33–72, 1988.

[GLSB11] Eyal En Gad, Michael Langberg, Moshe Schwartz, and Jehoshua Bruck. Constant-weight Gray codes for local rank modulation. *IEEE Transactions on Information Theory*, 57(11):7431–7442, 2011.

[GP90] Zvi Galil and Kunsoo Park. An improved algorithm for approximate string matching. *SIAM Journal on Computing*, 19(6):989–999, 1990.

[GPR95] Leszek Gasieniec, Wojciech Plandowski, and Wojciech Rytter. The zooming method: A recursive approach to time-space efficient string-matching. *Theor. Comput. Sci.*, 147(1&2):19–30, 1995.

[Gra53] Frank Gray. Pulse code communication, 1953.

[Gra16] Szymon Grabowski. New tabulation and sparse dynamic programming based techniques for sequence similarity problems. *Discrete Applied Mathematics*, 212:96–103, 2016.

[GS81] Zvi Galil and Joel Seiferas. Time-space-optimal string matching (preliminary report). In *Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computing*, STOC '81, pages 106–113, 1981.

[GST17] Arnab Ganguly, Rahul Shah, and Sharma V. Thankachan. pBWT: Achieving succinct data structures for parameterized pattern matching and related problems. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017*, pages 397–407, 2017.

[GW12] Fabrizio Grandoni and Virginia Vassilevska Williams. Improved distance sensitivity oracles via fast single-source replacement paths. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012*, pages 748–757, 2012.

[HR16] Felix Herter and Günter Rote. Loopless Gray code enumeration and the tower of bucharest. In *8th International Conference on Fun with Algorithms, FUN 2016, June 8-10, 2016, La Maddalena, Italy*, pages 19:1–19:19, 2016.

[Ind98] Piotr Indyk. Faster algorithms for string matching problems: Matching the convolution bound. In *39th Annual Symposium on Foundations of Computer Science, FOCS '98,*, pages 166–173, 1998.

[Ita77] Alon Itai. Finding a minimum circuit in a graph. In *Proceedings of the 9th Annual ACM Symposium on Theory of Computing*, pages 1–10, 1977.

[JMSB09] Anxiao Jiang, Robert Mateescu, Moshe Schwartz, and Jehoshua Bruck. Rank modulation for flash memories. *IEEE Transactions on Information Theory*, 55(6):2659–2673, 2009.

[JWW80] James T. Joichi, Dennis E. White, and S. G. Williamson. Combinatorial Gray codes. *SIAM J. Comput.*, 9(1):130–141, 1980.

[KJP77] Donald E. Knuth, James H. Morris Jr., and Vaughan R. Pratt. Fast pattern matching in strings. *SIAM J. Comput.*, 6(2):323–350, 1977.

[Knu11] Donald E. Knuth. *The Art of Computer Programming. Volume 4A: Combinatorial Algorithms, Part 1.* Addison-Wesley Professional, 2011.

[KP18] Tsvi Kopelowitz and Ely Porat. A simple algorithm for approximating the text-to-pattern hamming distance. In *1st Symposium on Simplicity in Algorithms, SOSA 2018*, pages 10:1–10:5, 2018.

[LAAA75] Arlazarov Vladimir Lvovich, Dinitz Yefim A, Kronrod M A, and Faradzhev I. Aleksandrovich. On economic construction of the transitive closure of a directed graph. pages 11:1209–1210, 1975.

[Lan87] Serge Lang. *Linear Algebra*. Undergraduate Texts in Mathematics. Springer New York, 1987.

[Lev66] VI Levenshtein. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*, 10:707, 1966.

[LMS98] Gad M. Landau, Eugene W. Myers, and Jeanette P. Schmidt. Incremental string comparison. *SIAM J. Comput.*, 27(2):557–582, 1998.

[LN96] Rudolf Lidl and Harald Niederreiter. *Finite Fields*. Encyclopedia of Mathematics and its Applications. 1996.

[LNS02] Christos Levcopoulos, Giri Narasimhan, and Michiel H. M. Smid. Improved algorithms for constructing fault-tolerant spanners. *Algorithmica*, 32(1):144–156, 2002.

[LT79] Thomas Lengauer and Robert Endre Tarjan. A fast algorithm for finding dominators in a flowgraph. *ACM Trans. Program. Lang. Syst.*, 1(1):121–141, 1979.

[Luc59] Harold M. Lucal. Arithmetic operations for digital computers using a modified reflected binary code. *IRE Transactions on Electronic Computers*, EC-8(4):449–458, 1959.

[Lud81] J. Ludman. Gray code generation for mpsk signals. *IEEE Transactions on Communications*, 29(10):1519–1522, 1981.

[Luk99] Tamás Lukovszki. New results of fault tolerant geometric spanners. In *Algorithms and Data Structures, 6th International Workshop, WADS '99, Proceedings*, pages 193–204, 1999.

[LV89] Gad M. Landau and Uzi Vishkin. Fast parallel and serial approximate string matching. *Journal of Algorithms*, 10(2):157–169, 1989.

[MP80] William J. Masek and Michael S. Paterson. A faster algorithm computing string edit distances. *Journal of Computer and System Sciences*, 20(1):18 – 31, 1980.

[Mun71] Ian Munro. Efficient determination of the transitive closure of a directed graph. pages 1(2):56–58, 1971.

[Mye86] Eugene W Myers. Incremental alignment algorithms and their applications. *Technical Report*, 1986.

[Nav01] Gonzalo Navarro. A guided tour to approximate string matching. *ACM Comput. Surv.*, 33(1):31–88, 2001.

[NRR13] Patrick K. Nicholson, Venkatesh Raman, and S. Srinivasa Rao. A survey of data structures in the bitprobe model. In *Space-Efficient Data Structures, Streams, and Algorithms - Papers in Honor of J. Ian Munro on the Occasion of His 66th Birthday*, pages 303–318, 2013.

[NSS17] Timothy Naumovitz, Michael E. Saks, and C. Seshadhri. Accurate and nearly optimal sublinear approximations to Ulam distance. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017*, pages 2012–2031, 2017.

[NW78] Albert Nijenhuis and Herbert S. Wilf. *Combinatorial Algorithms*. Academic Press, 1978.

[Par14] Merav Parter. Vertex fault tolerant additive spanners. In *Distributed Computing - 28th International Symposium, DISC 2014, Proceedings*, pages 167–181, 2014.

[Par15] Merav Parter. Dual failure resilient BFS structure. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing, PODC 2015*, pages 481–490, 2015.

[Pat08] Mihai Patrascu. Succincter. In *49th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2008*, pages 305–313, 2008.

[PP13] Merav Parter and David Peleg. Sparse fault-tolerant BFS trees. In *Algorithms - ESA 2013 - 21st Annual European Symposium, Proceedings*, pages 779–790, 2013.

[PP14] Merav Parter and David Peleg. Fault tolerant approximate BFS structures. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014*, pages 1073–1092, 2014.

[Ras17] Mikhail Raskin. A linear lower bound for incrementing a space-optimal integer representation in the bit-probe model. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017*, pages 88:1–88:12, 2017.

[RC81] John P. Robinson and Martin Cohn. Counting sequences. *IEEE Transactions on Computers*, C-30(1):17–23, 1981.

[Ric86] Dana S. Richards. Data compression and Gray-code sorting. *Inf. Process. Lett.*, 22(4):201–205, 1986.

[RM10] Mohammed Ziaur Rahman and J. Ian Munro. Integer representation and counting in the bit probe model. *Algorithmica*, 56(1):105–127, 2010.

[RS78] Imre Z. Ruzsá and Endre Szemerédi. Triple systems with no six points carrying three triangles. In *Colloquia Mathematica Societatis János Bolyai*, pages 939–945, 1978.

[RZ12] Liam Roditty and Uri Zwick. Replacement paths and k simple shortest paths in unweighted directed graphs. *ACM Trans. Algorithms*, 8(4):33:1–33:11, 2012.

[Sah14] Barna Saha. The dyck language edit distance problem in near-linear time. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS , 2014*, pages 611–620, 2014.

[Sav97] Carla Savage. A survey of combinatorial Gray codes. *SIAM review*, 39(4):605–629, 1997.

[Sel80] Peter H. Sellers. The theory and computation of evolutionary distances: pattern recognition. *Journal of Algorithms*, pages 1:359–373, 1980.

[Sho92] Victor Shoup. Searching for primitive roots in finite fields. *Mathematics of Computation*, 58:369 – 380, 1992.

[Shpa] Igor Shparlinski. *Finite Fields: Theory and Computation*, volume 477 of *Mathematics and its Applications*. Springer Netherlands.

[Shpb] Igor E. Shparlinski. Finding irreducible and primitive polynomials. *Applicable Algebra in Engineering, Communication and Computing*, 4(4):263–268.

[Shp96] Igor Shparlinski. On finding primitive roots in finite fields. *Theoretical Computer Science*, 157(2):273 – 275, 1996.

[Sta17] Tatiana A. Starikovskaya. Communication and streaming complexity of approximate pattern matching. In *28th Annual Symposium on Combinatorial Pattern Matching, CPM 2017*, pages 13:1–13:11, 2017.

[Str69] Volker Strassen. Gaussian elimination is not optimal. In *Numer. Math*, pages 13:354–356, 1969.

[Ukk85] Esko Ukkonen. Algorithms for approximate string matching. *Inf. Control*, 64:100–118, 1985.

[UW93] Esko Ukkonen and Derick Wood. Approximate string matching with suffix automata. *Algorithmica*, 10(5):353–364, 1993.

[Val75] Leslie G. Valiant. General context-free recognition in less than cubic time. pages 10(2):308–315, 1975.

[VZA70] M. A. Kronrod V. Z. Arlazarov, E. A. Dinic. On economical construction of the transitive closure of a directed graph. pages 11(5):1209–1210, 1970.

[WF74] Robert A. Wagner and Michael J. Fischer. The string-to-string correction problem. *J. ACM*, 21(1):168–173, 1974.

[Wil11] Virginia Vassilevska Williams. Faster replacement paths. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011*, pages 1337–1346, 2011.

[Wil12] Virginia Vassilevska Williams. Multiplying matrices faster than coppersmith-winograd. In *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012*, pages 887–898, 2012.

[Woo10] David P. Woodruff. Additive spanners in nearly quadratic time. In *Automata, Languages and Programming, 37th International Colloquium, ICALP 2010*, pages 463–474, 2010.

[WY13] Oren Weimann and Raphael Yuster. Replacement paths and distance sensitivity oracles via fast matrix multiplication. *ACM Trans. Algorithms*, 9(2):14:1–14:13, 2013.

[Yao81] Andrew Chi-Chih Yao. Should tables be sorted? *J. ACM*, 28(3):615–628, 1981.

[YS12] Yonatan Yehezkeally and Moshe Schwartz. Snake-in-the-box codes for rank modulation. *IEEE Transactions on Information Theory*, 58(8):5471–5483, 2012.

[Yu15] Huacheng Yu. An improved combinatorial algorithm for boolean matrix multiplication. In *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015*, pages 1094–1105, 2015.

List of Figures

2.1 $\mathcal{V}(x_{I_1}, \epsilon)$ and $\mathcal{V}(x_{I_3}, \epsilon)$ are disjoint. 20

2.2 Illustration of the Covering Algorithm: Blue boxes are low cost boxes in dense w_1-strips, while the yellow ones are in sparse w_1-strips. The red line corresponds to the path τ that we are trying to cover. In each w_2-strip, τ is covered by either a collection of many w_1-boxes or it is covered by a diagonal extension of a low cost w_1-box. The various boxes might overlap vertically which is not shown in the picture. 23

2.3 Illustration of a path that crosses a grid: Solid blue edges are the edges of a given path τ. Dotted blue edges are the edges of path $\tau_{I'}$ that crosses the dashed strip G'. 28

2.4 An illustration of δ-aligned interval. Here interval J'' is δ-aligned but J' is not. 29

2.5 Illustration of diagonal extension: Given a w-box $I' \times J'$ its true diagonal extension is the grey box $I \times \hat{J}$. 30

2.6 (a) The shortcut edge e_i is added for box $I_i \times J_i$. (b) An example of a path τ (in solid) passing through a box $I_h \times J_h$. The dashed path ${\tau'}_h$ is an approximation of τ between p_h and p_{h+1}. Here $s_h = (\min(I_h), \min(J'_h))$ and $t_h = (\max(I_h), \max(J'_h))$. 37

5.1 Suppose the yellow colored region represents G_{short}. The edges of C_1 and C_2 are colored with blue and red respectively. Brown colored edges are the edges added in the auxiliary graph G' and the edges colored with green constitute the set $\mathcal{E}(t)$. Paths represented by the thick edges are the 3 edge disjoint paths in G' when $r = 2$. 78

5.2 Region shaded with green color represents G_σ for $\sigma = (1, -1, \cdots, -1)$ whereas yellow colored region is the shortest path subgraph of G. The edges of $C_{(-1,\cdots,-1),1}$ and $C_{(-1,\cdots,-1),2}$ are colored with blue and red respectively. G_σ is obtained by removing red colored edges. . . 81

9.1 Elementary matrices 112

List of Tables

6.1 Taxonomy of construction of Gray/quasi-Gray codes over $\mathbb{Z}_m^n$ . . . 92

List of publications

[CDGKS 18] **Approximating Edit Distance within Constant Factor in Truly Sub-Quadratic Time**

with Diptarka Chakraborty, Elazar Goldenberg, Michal Koucký, Michael Saks

59th IEEE Annual Symposium on Foundations of Computer Science (FOCS), 2018

[DKS 18] **Lower Bounds for Combinatorial Algorithms for Boolean Matrix Multiplication**

with Michal Koucký, Michael Saks

35th Symposium on Theoretical Aspects of Computer Science (STACS), 2018

[CD 18] **Sparse Weight Tolerant Subgraph for Single Source Shortest Path**

with Diptarka Chakraborty

16th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT), 2018

[CDKS 18] **Space-Optimal Quasi-Gray Codes with Logarithmic Read Complexity**

with Diptarka Chakraborty, Michal Koucký, Nitin Saurabh

26th Annual European Symposium on Algorithms (ESA), 2018

Unpublished Manuscripts

[CDK 18] **Approximate Online Pattern Matching in Sub-linear Time**

with Diptarka Chakraborty, Michal Koucký

CoRR abs/1810.03551, 2018