On Graph Algorithms for Large-Scale Graphs

Dissertation

zur Erlangung des Grades eines
Doktors der Naturwissenschaften
der Technischen Universität Dortmund
an der Fakultät für Informatik

von

Marc Bury geb. Gillé

Dortmund

2015

Tag der mündlichen Prüfung:	5. Februar 2016
Dekan:	Prof. Dr. Gernot Fink
Gutachter:	Prof. Dr. Beate Bollig
	Dr. (habil.) Martin Sauerhoff

Summary

The algorithmic challenges have changed in the last decade due to the rapid growth of the data set sizes that need to be processed. New types of algorithms on large graphs like social graphs, computer networks, or state transition graphs have emerged to overcome the problem of ever-increasing data sets. In this thesis, we investigate two approaches to this problem.

Implicit algorithms utilize lossless compression of data to reduce the size and to directly work on this compressed representation to solve optimization problems. In the case of graphs we are dealing with the characteristic function of the edge set which can be represented by Ordered Binary Decision Diagrams (OBDDs), a well-known data structure for Boolean functions. We develop a new technique to prove upper and lower bounds on the size of OBDDs representing graphs and apply this technique to several graph classes to obtain (almost) optimal bounds. A small input OBDD size is absolutely essential for dealing with large graphs but we also need algorithms that avoid large intermediate results during the computation. For this purpose, we design algorithms for a specific graph class that exploit the encoding of the nodes that we use for the results on the OBDD sizes. In addition, we lay the foundation on the theory of randomization in OBDD-based algorithms by investigating what kind of randomness is feasible and how to design algorithms with it. As a result, we present two randomized algorithms that outperform known deterministic algorithms on many input instances.

Streaming algorithms are another approach for dealing with large graphs. In this model, the graph is presented one-by-one in a stream of edge insertions or deletions and the algorithms are permitted to use only a limited amount of memory. Often, the solution to an optimization problem on graphs can require up to a linear amount of space with respect to the number of nodes, resulting in a trivial lower bound for the space requirement of any streaming algorithm for those problems. Computing a matching, i. e., a subset of edges where no two edges are incident to a common node, is an example which has recently attracted a lot of attention in the streaming setting. If we are interested in the size (or weight in case of weighted graphs) of a matching, it is possible to break this linear bound. We focus on so-called dynamic graph streams where edges can be inserted and deleted and reduce the problem of estimating the weight of a matching to the problem of estimating the size of a maximum matching with a small loss in the approximation factor. In addition, we present the first dynamic graph stream algorithm for estimating the size of a matching in graphs which are locally sparse. On the negative side, we prove a space lower bound of streaming algorithms that estimate the size of a maximum matching with a small approximation factor.

Acknowledgments

This work would not have been possible without the help of so many people. I would like to express my gratitude to my advisor Beate Bollig for many inspiring discussions on research and everyday life, for her friendly support during tougher times and for calling on me to keep going whenever I needed it. I am also very grateful to Martin Sauerhoff for being part of my thesis committee. I am very happy for this opportunity, as my two examiners were instrumental in motivating and inspiring me during my undergraduate studies and drew me towards theoretical computer science in the first place. I thank Chris Schwiegelshohn and Melanie Schmidt for the terrific time we spent on research and for always having an open ear. I thank Beate Bollig, Hendrik Fichtenberger, Melanie Schmidt, and Chris Schwiegelshohn for proof-reading this thesis. Also, I wish to thank Alexander Munteanu for making our shared office a comfortable working place and for so many inspiring coffee breaks! I wish to thank my colleagues, past and present, for the nice working climate and their willingness to help.

Most importantly, I extend my heartfelt gratitude to my wife Anna and my daughter Emma. I would never have managed without them.

Danksagungen

Ohne die Hilfe von so vielen Personen wäre diese Arbeit niemals möglich gewesen. Zuallererst bedanke ich mich bei meiner Betreuerin Beate Bollig für die vielen anrengenden Diskussionen sowohl über Forschungsthemen als auch über alltägliche Fragen und Probleme, die freundliche Unterstützung auch in nicht so guten Zeiten und auch für die manchmal notwendigen auffordernden Worte. Ich danke Martin Sauerhoff, dass er sich bereiterklärt hat, diese Arbeit zu begutachten. Es freut mich umso mehr, da meine beiden Gutachter auch diejenigen waren, die mich zu meiner Studienzeit motiviert und begeistert haben und mein Interesse an theoretischer Informatik erst geweckt haben. Ich danke Chris Schwiegelshohn und Melanie Schmidt für die tolle (Forschungs-)Zeit am Lehrstuhl und die vielen gemeinsamen Diskussionen. Fürs Korrekturlesen dieser Arbeit bedanke ich mich bei Beate Bollig, Hendrik Fichtenberger, Melanie Schmidt und Chris Schwiegelshohn. Außerdem danke ich Alexander Munteanu für die schöne Zeit im gemeinsamen Büro und die vielen inspirierenden Kaffeepausen! Meinen jetzigen und ehemaligen Kollegen danke ich für die angenehme Arbeitsatmosphäre und Hilfsbereitschaft.

Mein größter Dank geht schließlich an meine Frau Anna und meine Tochter Emma, ohne die ich es nicht soweit geschafft hätte.

Contents

1. Introduction and Overview

Less is sometimes more. This saying has a lot of interpretations and it had nothing to do with computer science in the first place. But if we try to apply it to the current development of theoretical computer science we see that there are at least two new interpretations that we present here to motivate the topic of this thesis.

Data is everywhere The continuous progress of networking has lead to a new kind of data: The emergence of massive graphs, for example the internet graph or social networks, changed the requirement on (graph) algorithms. Classically efficient algorithms with polynomial or even linear running time cannot be used on such graphs without incurring infeasible running times or memory requirements. These graphs are not only becoming even larger by embedding sensors, software, and electronics in everything, which is also known as Internet of Things, but also the velocity of the data is increasing more and more. Small devices such as embedded systems or network routers need to analyze and process incoming data but do not have enough memory to store information on every single data package. The computations also have to be done immediately when the data arrives. The kind of algorithms which are necessary in this scenario are usually called *data stream algorithms*. Streaming algorithms realize the "Less is sometimes more" paradigm in the context of Big Data analysis by using a small amount of space to maintain a summary or a sketch of the data.

Keep it all The massive size of the data in the data stream setting is often caused by the immense number of generators or users. Often, a lone data package yields no significant information by itself. We may be only interested in general properties of the entire data set, which may allow us to produce and analyze a rough aggregation or summary of the data instead. In terms of the type of solutions in streaming algorithms, we consider approximations and estimations which are neither optimal or exact nor succeeding all the time. While this phenomenon is inevitable for most streaming algorithms, and indeed sometimes even desirable, there nevertheless exist applications where we need an exact or optimal algorithm even when facing very large inputs. Examples include verification where it is crucial to have exact or optimal algorithms. A different line of research focuses on storing datasets implicitly, for instance via a function, in the hope of getting a more compact representation. *Implicit algorithms* operating in this model have no direct access to the data and are limited to using the implicit representation. In some cases, an implicit representation of the data set, as well

as the implicit representation of every intermediate result of the algorithm, is smaller than the explicit one. Perhaps unsurprisingly, not all datasets can be compressed by a significant amount. Thus, implicit algorithms are rather a heuristic approach to deal with massive inputs. Here, "Less is sometimes more" means that we use compression to deal with very large inputs. It also means that we use a small amount of space but in the sense of a compact representation of the entire data.

The Main Problem Due to the limitations of both mentioned models, the complexity of problems is increasing compared to classical computation models. In particular, easy problems which can be solved in polynomial time and space may not be feasible for implicit or streaming algorithms. For instance, a lot of graph problems which are known to be in the complexity class P become PSPACE-complete if the inputs are given in an implicit representation (e. g., see [46]) which is a significant increase in terms of complexity. Therefore, we focus on a classical and easy problem that can still be computed in the limited models: We investigate the problem of computing or approximating matchings in graphs. A matching in a graph $G = (V, E)$ where V is the set of nodes and $E \subseteq V \times V$ is the set of edges is a subset of E where no two edges have a node in common. A disjoint pairing of nodes has a lot of applications, for instance for assigning advertisers to display slots, finding partners in a social network, or for scheduling jobs on a set of machines. Matchings have already been investigated for both implicit algorithms and streaming algorithms. In particular, there is a lot of current work on matchings in the streaming framework.

In order to present the results of this thesis, we first describe the two aforementioned frameworks in more detail. Then we give a brief overview on the contribution of this thesis in the areas of implicit algorithms and streaming algorithms. Last but not least, we explain the structure of the thesis and mention the publications that this thesis is based on and the contribution of the author to these publications.

1.1. Implicit Algorithms

A way to implicitly store a binary encoded object $O \in \{0,1\}^n$ is to use the characteristic function χ_O of O which is defined by $\chi_O(i) = 1$ iff $O_i = 1$. This function can be represented by known data structures for Boolean functions. We use so-called *Ordered Binary Decision Diagrams* (OBDDs) introduced by Bryant [25] to represent χ_O. For an overview of the representation of Boolean function by OBDDs and other data structures we refer to the book by Wegener [128]. The reason for choosing OBDDs is that on the one hand OBDDs allow a significant reduction in the representation size for Boolean functions while supporting several so-called functional operations efficiently. Among these are equality testing, conjunction or disjunction of two Boolean functions, or counting the number of inputs evaluating to 1. All algorithms presented in this thesis are graph algorithms. A graph $G = (V, E)$ consists of a set

of nodes V and a set of edges $E \subseteq V \times V$ representing connections between the nodes. Then the implicit representation of the graph is the characteristic function of the edge set. Several graph problems, for instance maximum flow [57, 115], topological sorting [131], connectivity [51], minimum spanning tree [15], and maximum matching [18] can be solved by OBDD-based algorithms using mainly functional operations. The complexity of OBDD-based algorithms is measured in the number of functional operations and the number of input bits of the used functions. However, the actual running time of the algorithms (and functional operations) is determined by the OBDD sizes of the used functions. Thus, keeping both number of functional operations and number of input bits which is a rough measure for the worst-case OBDD size as small as possible is the overall goal in designing implicit algorithms.

1.2. Graph Stream Algorithms

In a graph data stream, the input graph is defined by a data stream which in this thesis consists of insertions and deletions of edges. A recent survey by McGregor [95] gives a comprehensive overview on the state of the art in graph streaming. Henzinger, Raghavan, and Rajagopalan [59] were the first to consider graph problems in the streaming model. Many streaming algorithms on graphs are in the semi-streaming model where the space usage is up to logarithmic factors proportional to the number of nodes in the graph. Graph problems are often intractable when only sublinear space with respect to the number of nodes is permitted (for instance, see [45]). Computing or approximating a matching in a data stream has recently gained a lot of attention [1, 9, 29, 28, 36, 42, 44, 53, 63, 73, 74, 80, 81, 133]. Some of these works break the linear space requirement which is inherently necessary whenever the output is a matching by estimating the size of the maximum matching instead. Depending on whether the stream also consists of edge deletions, the complexities (in terms of required space) of the matching problems can vary. Therefore, the posed challenges are also different depending on the model and the problem. In insertion-only streams, it is an open question what the best approximation factor smaller or equal than 2 is with $\mathcal{O}(|V| \operatorname{polylog} |V|)$ space if we want to compute a matching. With edge deletions even $|V|^{\varepsilon}$ approximation factors need $\Omega(|V|^{2-3\varepsilon})$ space [9]. The problem of estimating the maximum matching size is less well understood. Even in the insertion-only streams, it is unknown whether it is possible to estimate the matching size with sublinear space in general graphs. We conclude this section by a quote of an entry in the list of open problems for sublinear algorithms [93] which was suggested by Andrew McGregor and which was the initial motivation for investigating matchings in data streams:

> *Consider an unweighted graph on N nodes defined by a stream of edge insertions and deletions. Is it possible to approximate the size of the maximum cardinality matching up to constant factor given a single pass and $o(N^2)$ space? Recall that a factor 2 approximation is easy in $\mathcal{O}(N \log N)$ space if there are no edge deletions.*

1.3. Organization and Results

As a general rule, every chapter begins with a brief summary of the content. If the chapter contains new results, then there is also a section on related work and on the contributions of the results in the context of previous works.

Chapter 2 In order to lay the foundations for the results and related work, we introduce the necessary notions and results in the area of graph classes, OBDDs, probability theory, graph streaming, and communication complexity.

Chapter 3 We show how to use the adjacency matrix of a graph to analyze the size of the OBDD representing the graph. Then we apply this technique to several graph classes such as (unit) interval graphs, (bi)convex graphs, trees, and more. For these graphs, the crucial point is to find a labeling of the nodes such that the neighborhood of a node has some structural properties. We show upper bounds on the OBDD size as well as lower bounds where a lower bound is the worst-case OBDD size for some graph from the class. For all graph classes we show optimal or up to logarithmic factors optimal upper bounds on the OBDD size. We also present a maximum matching algorithm for unit interval graphs where we exploit the labeling of the nodes to get a simple algorithm using only $\mathcal{O}(\log N)$ functional operations. Then we present a coloring algorithm for interval graphs and unit interval graphs using $\mathcal{O}(\log^2 N)$ functional operations that colors the nodes of the graph with a minimum amount of colors while two adjacent nodes cannot have the same color.

Chapter 4 In order to design simple implicit algorithms with a small number of functional operations, we investigate the uses of randomization in OBDD-based algorithms by constructing OBDDs for random functions. In general, a function where the function values are chosen randomly is not representable by an OBDD of small size. Thus, we use random functions where the function values are not completely independent but k-wise independent. Informally, this means that the distribution of at most k function values behaves as in the case of complete independence. We show that there is a sharp separation between 4-wise random functions and 3-wise random functions. While the former has an exponentially large lower bound of $2^{\Omega(n)}$, the latter can be represented by OBDDs of size $\mathcal{O}(n)$ where n is the number of input bits. We also give an OBDD construction of so-called almost k-wise independent random functions that are "close" to k-wise independent random functions measured by some distance parameter ε. The resulting OBDD size is $\mathcal{O}((nk)^2/\varepsilon)$. Altogether, this gives a clear picture of the landscape of random functions which can be presented by OBDDs of small size. In the algorithmic part, we present a randomized minimum spanning tree (MST) algorithm and a randomized maximal matching algorithm. The MST algorithm has no proven guarantee on the number of functional operations and it is more an example how to use randomization

to design an OBDD-based algorithm. On the other hand, the matching algorithm is not only simple but we can also prove an upper bound of $\mathcal{O}(\log^3 N)$ on the expected number of functional operations. The matching algorithm is currently the best implicit algorithm with respect to both the number of used input variables ($3 \log N$) and the number of functional operations.

Chapter 5 For an experimental evaluation we have implemented all OBDD-based algorithms presented in this thesis and the implicit maximum matching algorithms by Bollig, Gillé, and Pröger [18]. The maximum matching algorithms can outperform the well-known explicit matching algorithm by Hopcroft and Karp on some instances. But in general the experiments show that decreasing the number of functional operations does not necessarily improve the running time and can even be contra-productive. Especially the maximal matching algorithm by Bollig and Pröger [20] with $\mathcal{O}(\log^4 N)$ functional operations is much worse than an algorithm by Hachtel and Somenzi [57] with $\mathcal{O}(N \log N)$ functional operations. Comparing the algorithm by Hachtel and Somenzi with our randomized maximal matching algorithm shows that the randomized algorithm is better on sparse graphs which makes it also possible to run the randomized algorithm on some sparse real-world instances where the other algorithm exceeds the memory limitation. The coloring algorithm for interval graphs is not feasible even on small instances whereas the maximum matching algorithm on unit interval graphs outperforms an explicit algorithm.

Chapter 6 As mentioned before, it is an open problem how to estimate the size of a maximum matching in a graph stream. However, for some special cases, for instance for locally sparse graphs with edge insertions only [44] or randomly ordered streams [74], it is possible to get an approximation on the matching size. Given an algorithm $\mathcal{A}$ that can estimate the size of a maximum matching with approximation factor λ, we present an algorithm that can estimate the weight of an optimal weighted matching up to an $\mathcal{O}(\lambda^4)$ factor using roughly the same space as $\mathcal{A}$. This reduction works in every streaming model and is only limited by the requirements of $\mathcal{A}$. We also extend a recent estimation algorithm by Esfandiari et al. [44] for locally sparse graphs in insertion-only streams to the case of edge deletions. We also prove a space lower bound of $\Omega(N^{1-\varepsilon})$ for every streaming algorithm that estimates the matching size with approximation factor $1 + \mathcal{O}(\varepsilon)$.

1.4. Publications and Contribution of the Author

The content of this thesis is based on the following publications:

- Chapter 3 and parts of Chapter 5 are based on [52]
  Gillé, M. OBDD-based representation of interval graphs. In *Proceedings of the 39th*

Workshop on Graph-Theoretic Concepts in Computer Science (WG) (2013), pp. 286–297. *Best Student Paper Award.*

- Chapter 4 and parts of Chapter 5 are based on:
  Bury, M. Randomized OBDD-based graph algorithms. In *Proceedings of the 22nd International Colloquium on Structural Information and Communication Complexity (SIROCCO)* (2015), To appear. *Best Student Paper Award.*

- Parts of Chapter 5 are based on [18]:
  Bollig, B., Gillé, M., and Pröger, T. Implicit computation of maximum bipartite matchings by sublinear functional operations. *Theoretical Computer Science 560* (2014), 131–146.

  The experimental evaluation in this paper was mainly done by the author.

- Chapter 6 is based on:
  Bury, M., and Schwiegelshohn, C. Sublinear Estimation of Weighted Matchings in Dynamic Data Streams. In *Proceedings of the 23rd Annual European Symposium (ESA)* (2015), To appear.

  Both authors contributed equally to this publication.

2. Preliminaries

This chapter contains the necessary notions, tools, and known results that we need for the next chapters. First, we give a brief overview of some notations we will use in this thesis. Then we start by presenting graph theoretic concepts including graph classes whose implicit representation size we are going to investigate. In Section 2.3, we define OBDDs formally, give some lower bound techniques for the size of an OBDD representing a Boolean function, and show how OBDDs and so-called functional operations can be used for optimization algorithms. Section 2.4 gives an introduction in probability spaces and random variables as a foundation for our randomized algorithms. Known results and techniques for streaming algorithms are given in Section 2.5 while Section 2.6 presents space lower bound techniques for streaming algorithms in terms of communication complexity.

2.1. Notation

The sets of all natural and real numbers are denoted by $\mathbb{N}$ and $\mathbb{R}$, respectively. For $a, b \in \mathbb{R}$ we denote by $[a, b]$ and (a, b) the closed interval of real numbers $x \in \mathbb{R}$ such that $a \leq x \leq b$ and the open interval of real numbers such that $a < x < b$, respectively. Accordingly, $(a, b]$ and $[a, b)$ denote the half-open intervals. In all cases, the numbers a and b are called endpoints of the interval where a is the left endpoint and b is the right endpoint. We denote the set of Boolean functions $f : \{0,1\}^n \to \{0,1\}$ by B_n. Let $(x_0, \ldots, x_{n-1}) = x \in \{0,1\}^n$ be a binary number of length n and $|x|_2 := \sum_{i=0}^{n-1} x_i \cdot 2^i$ the value of x. For any natural number $l \in \mathbb{N}$, let $[l]_2$ denote the corresponding binary number of l, i. e., $|[l]_2|_2 = l$. For two binary numbers $x, y \in \{0,1\}^n$ we denote by $x \oplus y$ the binary number resulting from computing the bitwise exclusive or of x and y. Further, we denote by $[x, y] \in \{0,1\}^{2n}$ the concatenation of the two vectors. The vector of length n where all entries are zero (one) is denoted by 0^n (1^n). We denote the set $\{1, \ldots, n\}$ by $[n]$. In asymptotic running times and space requirements we use $\tilde{O}(f(n))$ to hide factors polylogarithmic in $f(n)$. We say that an event occurs with high probability if the probability of the event is at least $1 - o(1)$.

2.2. Graphs

For the sake of clarity, we start with some formal definitions of graphs.

Definition 2.2.1 (General Graphs)**.**

- A *directed* (*undirected*) *graph* $G = (V, E)$ consists of a finite set of nodes V and a set E of ordered (unordered) pairs of nodes called edges.
- A *weighted graph* $G = (V, E, w)$ is a directed or undirected graph with an additional weight function $w : E \to \mathbb{R}$ that assigns a weight to each edge.
- A graph $G' = (V', E')$ is a *subgraph* of G if $V' \subseteq V$ and $E' \subseteq E \cap (V' \times V')$.
- For a subset $U \subseteq V$ of nodes we denote the set of edges containing only nodes from U by $E(U) \subseteq E$ and say that U *induces* the subgraph $G(U) = (U, E(U))$ on G.

If nothing else is stated a graph $G = (V, E)$ is undirected and we denote the number of nodes by N, i. e., $|V| = N$.

Definition 2.2.2. Let $G = (V, E)$ be a graph.

- Two nodes $u, v \in V$ are called *adjacent* iff $e = \{u, v\} \in E$. In this case, the edge e is called *incident* to u and v.
- The *degree* $\deg_E(v)$ of a node $v \in V$ is the number of edges incident to v.
- The *neighborhood* $N(v)$ of a node $v \in V$ is the set of adjacent nodes, i. e., $N(v) = \{u \in V \mid \{u, v\} \in E\}$.
- A sequence $P = (v_1, \ldots, v_l)$ of pairwise distinct nodes is a *path* if $\{v_1, v_2\}, \ldots, \{v_{l-1}, v_l\} \in E$. The *length* of P is the number $l - 1$ of the edges of P. If also $\{v_l, v_1\} \in E$ then $C = (v_1, \ldots, v_l, v_1)$ is called a *cycle* and the length of C is l.
- A *chord* of a path P (cycle C) is an edge between two nodes of P (C) that is not an edge of the path (cycle). A path (cycle) with no chord is called *chordless*.
- G is *connected* if for all $u, v \in V$ with $u \neq v$, there is a path $(u = v_1, \ldots, v_l = v)$ in G.
- G is called *acyclic* if there exists no cycle in the graph.

Definition 2.2.3. Let $G = (V, E)$ be a directed graph.

- G is called *symmetric* if $(u, v) \in E \Leftrightarrow (v, u) \in E$ holds for all all $u, v \in V$.
- The *indegree* $d^-(v)$ and *outdegree* $d^+(v)$ of a node $v \in V$ is the number of edges in E of the form (u, v) and (v, u), respectively.
- G is *rooted* at a node $v \in V$ if $d^-(v) = 0$ and there is a path to every node in V starting in v.
- A node $v \in V$ is called a *sink* if $d^+(v) = 0$.

The *adjacency matrix* or *adjacency lists* are two possible ways to represent a graph $G = (V, E)$ explicitly. Let $V = \{v_0, \ldots, v_{N-1}\}$. In an adjacency matrix $A \in \{0,1\}^{N \times N}$ the entry a_{ij} is equal to 1 if $\{v_i, v_j\} \in E$ and 0 otherwise. In the adjacency list representation we have a list L_i containing all the neighbors of the node v_i for every $0 \leq i \leq N-1$. The space usage of an adjacency matrix is $\Theta(N^2)$ whereas the adjacency list representation needs $\Theta(|E| \log N)$ space.

Regarding counting of graphs we have to be careful whenever two graphs are essentially the same. In this case we say that the graphs are isomorphic which is formally defined as follows.

Definition 2.2.4 (Isomorphic Graphs)**.** Two graphs $G = (V_1, E_1)$ and $H = (V_2, E_2)$ are *isomorphic* ($G \sim H$) if there is a bijection $l : V_1 \to V_2$ such that for all $u, v \in V_1$ it is $\{u, v\} \in E_1$ if and only if $\{l(u), l(v)\} \in E_2$.

2.2.1. Graph Classes

Based on the definitions of the last section we define some graph classes which we will later investigate in terms of their OBDD size. Most of the definitions and results are taken from [23]. We start with two important classical graph classes: trees and bipartite graphs.

Definition 2.2.5 (Tree)**.** A graph $G = (V, E)$ is called a *tree* if G contains no cycle and is connected. Nodes with degree 1 in a tree are called *leaves*.

Trees have been studied extensively and there are several characterizations and properties of them. Here we mention only those that are important for the rest of the thesis.

Proposition 2.2.6 ([23])**.**

- *A connected graph G is a tree if and only if G has $N - 1$ edges.*
- *Every tree with more than one node has at least two leaves.*

Definition 2.2.7 (Bipartite Graph)**.** A graph $G = (V, E)$ is called *bipartite* if the node set can be partitioned into two sets $V = A \cup B$ such that for every edge $\{u, w\} \in E$ it holds either $u \in A$ and $v \in B$ or $u \in A$ and $v \in B$, i. e., two nodes from the same set A or B must not be adjacent. For a bipartite graph partitioned into A and B we also write $G = (A, B, E)$.

Bipartite graphs can also be characterized as the graphs without cycles of odd length which implies that trees are always bipartite. The next few graph classes are instances of so-called intersection graphs where every node represents a set and the adjacency relation between nodes is equivalent to the intersection relation between the corresponding sets.

Definition 2.2.8 (Intersection Graph)**.** Let $\mathcal{S} = \{S_1, \ldots, S_N\}$ be a family of sets. The intersection graph $G_\mathcal{S} = (V, E)$ of $\mathcal{S}$ is a graph with one node $v_i \in V$ for each set S_i where two nodes are adjacent whenever the two corresponding sets have a nonempty intersection, i. e., the edge set is defined by

$$E = \{\{v_i, v_j\} \mid i \neq j \text{ and } S_i \cap S_j \neq \emptyset\}.$$

We call $\mathcal{S}$ an *intersection model* of $G_\mathcal{S}$.

We start with a simple family of sets, namely intervals.

Definition 2.2.9 ((Unit) Interval (Bi)Graph)**.** A *(unit) interval graph* is an intersection graph that has an intersection model consisting of (unit length) intervals on a straight line. A bipartite graph $G = (A, B, E)$ is an *(unit) interval bigraph* if every vertex can be assigned to a (unit length) interval on the real line such that for all $u \in A$ and $v \in B : \{u, v\} \in E$ iff the corresponding intervals intersect.

Note that a (unit) interval bigraph is not a bipartite interval graph in general since the intervals in A or B can intersect.

Straight lines in Euclidean space can be another family of sets of an intersection graph, i. e., two nodes are adjacent if and only if their corresponding lines are crossing. Restricted straight lines can be used to characterize so-called permutation graphs which we define next.

Definition 2.2.10 ((Bipartite) Permutation Graph)**.** A graph is a *permutation graph* if it has an intersection model consisting of line segments whose endpoints lie on two parallel lines. A graph is a *bipartite permutation graph* if it is both bipartite and a permutation graph.

An interesting property of bipartite permutation graphs is that they admit an order of A and an order of B such that the neighborhood of every node from A (B) is consecutive in the order of B (A). This kind of order is interesting in itself.

Definition 2.2.11 (Properties of Node Orders)**.** Let $G = (A, B, E)$ be a bipartite graph.

An order $\prec_A$ of A in G has the *adjacency property* if for each node $u \in B$ the neighborhood $N(u) \subseteq A$ of u consists of nodes that are consecutive in the order $\prec_A$.

An order $\prec_A$ of A in G has the *enclosure property* if for each pair $u, v \in B$ such that $N(u) \subset N(v)$ the set $N(v) \setminus N(u)$ consists of nodes that are consecutive in the order $\prec_A$.

A *strong order* is a labeling of A and a labeling of B such that for all $\{a_i, b_j\} \in E$ and $\{a_{i'}, b_{j'}\} \in E$, where $a_i, a_{i'} \in A$ and $b_j, b_{j'} \in B$, if $i < i'$ and $j' < j$, then $\{a_i, b_{j'}\} \in E$ and $\{a_{i'}, b_j\} \in E$.

Theorem 2.2.12 ([121]). *Let $G = (A, B, E)$ be a bipartite graph. Then the following statements are equivalent:*

- *G is a bipartite permutation graph.*
- *G admits an order of A that has the adjacency and the enclosure property.*
- *G admits a strong order.*

We will see that ordering nodes and characterizing the neighborhood of a node with respect to the order is important for some upper bounds for the representation size. Therefore, we investigate additional graph classes whose definition is based on the above order properties.

Definition 2.2.13 ((Bi)Convex Graph). A bipartite graph $G = (A, B, E)$ is *convex* if there is an order of either A or B that fulfills the adjacency property.

A bipartite graph $G = (A, B, E)$ is *biconvex* if there is an order of A and B that fulfills the adjacency property.

We finish this subsection with cographs and two graph classes which are defined by node weights and form interesting subclasses of permutation graphs and bipartite permutation graphs. Figure 2.1 gives an overview of the intersection relationships between the classes defined here (see [23]).

Definition 2.2.14 (Cograph). A graph $G = (V, E)$ is a *cograph* if G contains no induced P_4, i. e., a path of length 3 on 4 nodes.

Definition 2.2.15 (Threshold Graph). A graph $G = (V, E)$ is a *threshold graph* if there is a real number T (the threshold) and for every node $v \in V$ there is a real weight w_v such that: $\{u, v\} \in E$ if and only if $w_u + w_v \geq T$.

Definition 2.2.16 (Chain/Difference Graph). The graph $G = (V, E)$ is a chain graph (or a difference graph) if there is a real number T and for every node $v \in V$ there is a real weight w_v with $|w_v| \leq T$ such that $\{u, v\} \in E$ if and only if $|w_v - w_u| \geq T$.

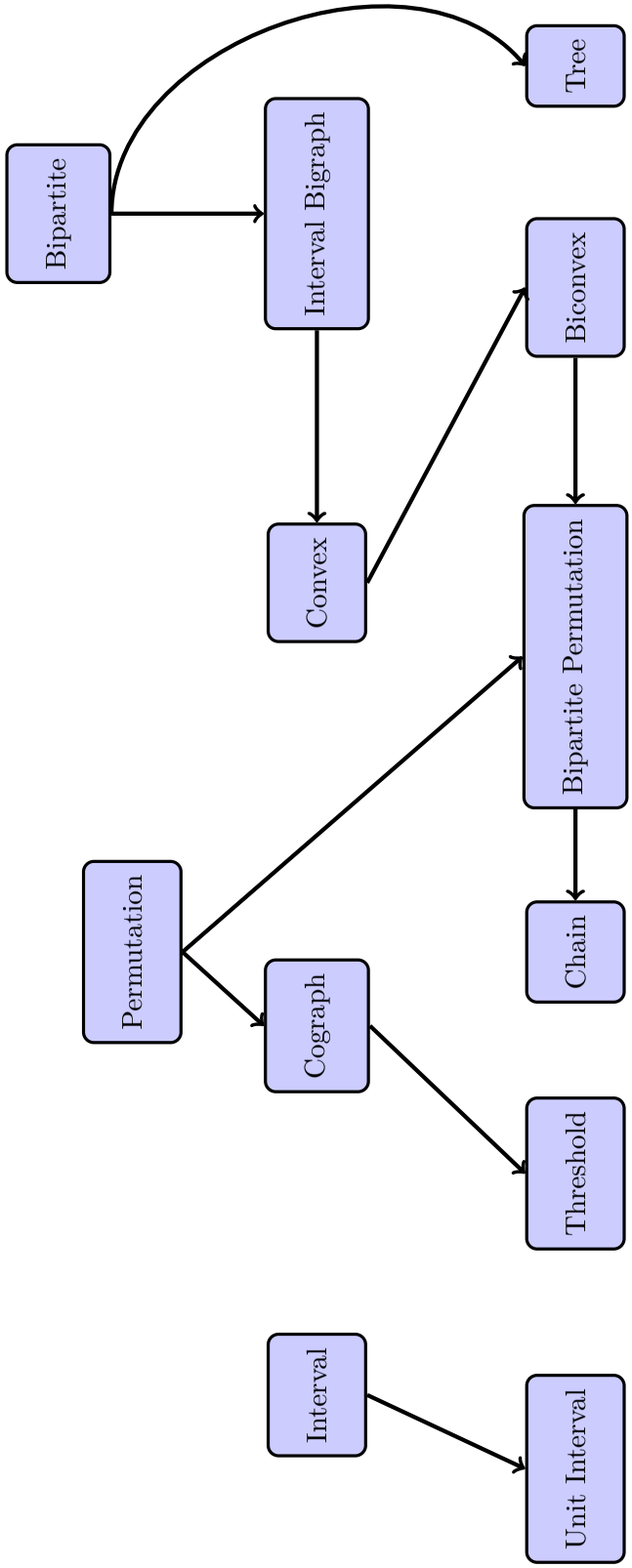


Figure 2.1.: Inclusion map of the defined graph classes. The arrangement of the classes reflects their representation size from Chapter 3 (see also Fig. 3.1).

2.2.2. Counting Graphs

Since we are interested in the representation size of graphs, we also want to prove some space lower bounds in terms of the worst case representation size of a graph coming from a graph class $\mathcal{C}$. Regarding bit complexity, counting is a simple but powerful technique to prove lower bounds: The number of necessary bits to represent K distinct objects is lower bounded by $\log K$ since b bits can represent at most 2^b different objects. In Section 2.3, we will see how the counting argument also works for lower bounds of the size of OBDDs representing graphs.

At a first glance, in order to count graphs we could fix a node set $V = \{v_0, \dots, v_{N-1}\}$ of size N, enumerate all graphs on V and count the number of graphs with are from a graph class $\mathcal{C}$. The main issue is that we count isomorphic graphs separately but for a lower bound we want to treat isomorphic graphs as the same graph. Therefore, we have to count the number of different equivalence classes $[G] := \{H \mid G \sim H \text{ with } H \in \mathcal{C}\}$ of graphs $G = (V, E)$ with size N. These two methods are known as counting *labeled* graphs and counting *unlabeled* graphs, respectively. We restrict this exposition to known results (or slight adaptations) about the graph classes we are interested in and give no detailed introduction in the topic of graph counting. We refer to [58] for an in-depth survey.

Lower bounds on the number of unlabeled graphs from a graph class would suffice to get lower bounds for the representation size. However, in classical graph counting there are often asymptotic exact expressions for the number of graphs, i. e., the number of graphs divided by the expression converges to 1. One of the first asymptotic formulas was given by Otter in 1948 for counting trees.

Theorem 2.2.17 ([109]). *The number of unlabeled trees $T(N)$ of size N satisfies*

$$\lim_{N\to\infty} \frac{T(N)}{C\alpha^N \cdot N^{-5/2}} = 1$$

for constants C and α.

For interval graphs Gavoi and Paul [50] showed a lower bound of $2^{\Omega(N \log N)}$ which we use to show a lower bound for convex graphs.

Theorem 2.2.18 ([50]). *The number $I(N)$ of labeled interval graphs is $2^{2N \log N - N \log\log N - \mathcal{O}(1)}$. The number of unlabeled interval graphs of size N is at least $I(N)/N!$ which is at least $2^{N \log N - o(N \log N)}$.*

We show that $2^{N \log N - o(N \log N)}$ is also a lower bound for the number of unlabeled convex graphs. The proof idea is from [123]. However, for the author of this thesis it was unclear whether they considered labeled or unlabeled graphs which is why we reformulate the proof here.

Theorem 2.2.19 (Similar to [123]). *The number $CONV(2N, N)$ of unlabeled convex graphs with partition sizes $|A| = 2N$ and $|B| = N$ is $2^{\Omega(N \log N)}$.*

Proof. We show that there is a surjective mapping ϕ from the set of labeled convex graphs to the set of labeled interval graphs. In addition, we show that for two non-isomorphic interval graphs there are two non-isomorphic convex graphs which are mapped to the two interval graphs under ϕ. This implies that $CONV(2N, N)$ is at least $2^{N \log N - o(N \log N)}$ by Theorem 2.2.18.

We start with the mapping from the labeled convex graphs into the set of all labeled interval graphs. Let $G = (V, E)$ be a convex graph. V is partitioned into A of size $2N$ and B of size N and w.l.o.g. A has the adjacency property, i.e., there exists an order $\preceq_A$ for the nodes in A such that for all $u \in B$ the neighborhood $N(u)$ consists of nodes that are consecutive in $\preceq_A$. W.l.o.g. assume that there are no isolated nodes and $A = \{a_1, \ldots, a_{2N}\}$ where $a_i \preceq a_j$ iff $i < j$. Let $\phi(G) = (V', E')$ be a graph with

$$V' = B \text{ and } E' = \{\{v, u\} \mid N(v) \cap N(u) \neq \emptyset\}.$$

We represent each node $u \in B$ by an interval $[l_u, r_u]$ where l_u and r_u is the smallest index of a neighbor of u and the largest index of a neighbor with respect to the order $\preceq_A$ of A, respectively. Clearly, this is an interval representation of $\phi(G)$ concluding that $\phi(G)$ is an interval graph.

Now let $H = (V, E)$ be an interval graph with an interval representation $I = \{[l_v, r_v] \mid v \in V\}$. W.l.o.g. we assume that all endpoints of the intervals in I are distinct, i. e., I has $2N$ different endpoints (see, e. g., [98] for a transformation if there are common endpoints). For every endpoint e we create a node a_e and define A to be the union of these nodes. Let $B = V$ and $E = \{\{v, a_e\} \mid v \in B, a_e \in A \text{ and } e \in [l_v, r_v]\}$. Then $G = (A, B, E)$ is a convex graph with the sorted sequence of endpoints as the order for A which has the adjacency property. Furthermore, it holds that $\phi(G) = H$ which means that ϕ is surjective.

Now, we show that for two non-isomorphic interval graphs G_I and $G_{I'}$ with interval representation $I = \{[l_i, r_i] \mid i \in \{1, \ldots, N\}\}$ and $I' = \{[l'_i, r'_i] \mid i \in \{1, \ldots, N\}\}$ there are two non-isomorphic convex graphs G and G' such that $\phi(G) = G_I$ and $\phi(G') = G_{I'}$. Let $G = (A, B, E)$ and $G' = (A', B', E')$ be the convex graphs from the construction above, i.e.

$$\begin{aligned} A &= \{u_p \mid p \text{ is endpoint in } I\}, \\ B &= \{1, \ldots, N\}, \\ E &= \{\{i, u_p\} \mid i \in B, u_p \in A \text{ and } p \in [l_i, r_i]\} \end{aligned}$$

$$\begin{aligned} A' &= \{u'_p \mid p \text{ is endpoint in } I'\}, \\ B' &= \{1, \ldots, N\}, \\ E' &= \{\{i, u'_p\} \mid i \in B', u'_p \in A' \text{ and } p \in [l'_i, r'_i]\} \end{aligned}$$

Assume that G and G' are isomorphic, i.e. there exists a bijective mapping $\pi : A \cup B \to A' \cup B'$

such that the edge relation is preserved, i.e. $\{i, u_p\} \in E$ iff $\{\pi(i), \pi(u_p)\} \in E'$. W.l.o.g. $\pi(i) \in A'$ and $\pi(u_p) \in B'$ for all $i \in B$ and $u_p \in A$. Otherwise, we can add a node u_{all} to A and a node w_{all} to B that are adjacent to all nodes from B and A, respectively (including the new nodes). Repeat this for A' and B'. If G and G' are isomorphic then the new graphs are also isomorphic (just add $u_{all} \to u'_{all}$ and $w_{all} \to w'_{all}$ to the mapping). If the new graphs are isomorphic then the node u_{all} and the node w_{all} have to be mapped to u'_{all} and w'_{all}, respectively, because no other nodes have the same degrees of $|B|+1$ and $|A|+1$. Therefore, the mapping induced on the nodes $A \cup B$ is a bijection to $A' \cup B'$ and preserves the edge relation. Thus, G and G' are also isomorphic. Therefore, we can split π into two bijections $\pi_1 : B \to B'$ and $\pi_2 : A \to A'$. But then for every $i \neq j$, $i, j \in \{1, \ldots, N\}$ we have

$$\begin{aligned}
[l_i, r_i] \cap [l_j, r_j] \neq \emptyset \quad &\Leftrightarrow \quad \exists u_p \in A : p \in [l_i, r_i] \text{ and } p \in [l_j, r_j] \\
&\Leftrightarrow \quad \{i, u_p\} \in E \text{ and } \{j, u_p\} \in E \\
&\Leftrightarrow \quad \{\pi_1(i), \pi_2(u_p)\} \in E' \text{ and } \{\pi_1(j), \pi_2(u_p)\} \in E' \\
&\Leftrightarrow \quad [l'_{\pi_1(i)}, r'_{\pi_1(i)}] \cap [l'_{\pi_1(j)}, r'_{\pi_1(j)}] \neq \emptyset
\end{aligned}$$

which can not be true since G_I and $G_{I'}$ are not isomorphic. Therefore, G and G' are also not isomorphic which completes the proof. □

Since in interval bigraphs both sides can use the same set of intervals, the lower bounds for interval graphs can be easily applied to interval bigraphs.

Theorem 2.2.20. *The number $BI(N, N)$ of unlabeled interval bigraphs with a size of N for both partitions is $2^{\Omega(N \log N)}$.*

The next theorems summarize known results for counting the remaining graph classes: (bipartite) permutation graphs, threshold graphs, and chain graphs.

Theorem 2.2.21 ([13])**.** *The number $P(N)$ of unlabeled permutation graphs of size N is $2^{\Omega(N \log N)}$.*

Theorem 2.2.22 ([112])**.** *For $N \geq 2$, the number $BP(N)$ of unlabeled connected bipartite permutation graphs of size N is given by*

$$BP(N) = \begin{cases} \frac{1}{4}\left(C(N-1) + C(N/2-1) + \binom{N}{N/2}\right) & \text{if } N \text{ is even,} \\ \frac{1}{4}\left(C(N-1) + \binom{N-1}{(N-1)/2}\right) & \text{otherwise} \end{cases}$$

where $C(N) := \frac{1}{N+1}\binom{2N}{N}$ is the N-th Catalan number.

Theorem 2.2.23 (Chapter 17.2 in [91])**.** *The number of unlabeled threshold graphs with maximum clique size k is $\binom{N-1}{k-1}$. This implies that the number $TH(N)$ of unlabeled threshold graphs is 2^{N-1}.*

Theorem 2.2.24 ([111]). *The number $CH(N)$ of unlabeled chain graphs of size N is $2^{N-2} + 2^{\lfloor N/2 \rfloor - 1}$.*

2.2.3. Matchings

The computation of a matching in a graph is a fundamental and intensively studied problem in graph theory. Lovász and Plummer [88] dedicate an entire book to the theory of matchings which illustrates the importance and complexity of matchings. We start with the formal definitions of different matching variants which we are investigating in this thesis.

Definition 2.2.25 (Matchings). Let $G = (V, E)$ be a graph. A set of edges $M \subseteq E$ is called a *matching* if no two edges of M have a node in common, i. e., if it holds for all $e, e' \in M$ with $e \neq e'$ that $e \cap e' = \emptyset$. A node is called *matched* (with respect to M) if it has an incident edge in M and *free* (with respect to M) otherwise.

M is called *maximal matching* if there is no edge $e \in E$ such that $M \cup e$ is a matching. M is called *maximum matching* if there is no matching with larger cardinality. In a weighted graph $G = (V, E, w)$ the weight of a matching M is the sum of the edge weights, i. e., $w(M) := \sum_{e \in M} w(e)$. M is called *maximum weighted matching* if for all matchings M' the weight $w(M')$ is at most $w(M)$.

One combinatorial approach to compute a maximum matching is to successively find a maximal set of so-called *augmenting paths* with respect to a current matching M: An M-augmenting path for a matching M is a path where the edges belong alternatingly to M and not to M and the path starts from and ends in a free node. By replacing the matching edges with the non-matching edges on the augmenting path, we can increase the matching size by one. Eventually, a matching is a maximum matching if and only if there exists no augmenting path in the graph [61]. In 1973, Hopcroft and Karp [61] gave the best known algorithm for computing a maximum matching in bipartite graphs with a running time of $\mathcal{O}(|E|\sqrt{N})$ which is based on finding augmenting paths. Edmonds [41] extended this result to general graphs with running time $\mathcal{O}(N^4)$ which was improved by Micali and Vazirani [100] to $\mathcal{O}(|E|\sqrt{N})$.

Using an algebraic point of view, Mucha and Sankowski [102] showed that a maximum matching can be computed in time $\mathcal{O}(N^\omega)$ where $\omega < 2.373$[1] is the exponent of the best known matrix multiplication algorithm. A key notion for algebraic matching algorithms is the *Tutte matrix* of the graph.

[1] The famous algorithm by Coppersmith and Winograd [34] with running time $\mathcal{O}(n^{2.376})$, which was the best bound for over twenty years, was recently improved several times: First in 2012 by Stothers [122] to $\mathcal{O}(n^{2.3737})$, then by Williams [130] to $\mathcal{O}(n^{2.3729})$, and, finally, by Le Gall [48] to $\mathcal{O}(n^{2.3728})$.

Definition 2.2.26 (Tutte Matrix)**.** Let $G = (V, E)$ be a graph with $V = \{v_0, \ldots, v_{N-1}\}$. The *Tutte matrix* T_G of G is a $N \times N$ matrix with entries

$$t_{ij} = \begin{cases} x_{ij} & \text{if } \{v_i, v_j\} \in E \text{ and } i < j, \\ -x_{ij} & \text{if } \{v_i, v_j\} \in E \text{ and } j < i, \\ 0 & \text{otherwise} \end{cases}$$

where x_{ij} are indeterminates.

Tutte [124] showed that a graph G has a perfect matching, i. e., a matching of size $N/2$, if and only if the determinant of T_G, which is a polynomial in the indeterminates, is not equal to 0. Lovász [87] generalized this result by relating the rank of the Tutte matrix to the size of a maximum matching.

Theorem 2.2.27 (Lovász [87])**.** *Let $G = (V, E)$ be a graph with a maximum matching M and Tutte matrix T_G. For an assignment $w \in \mathbb{R}^{|E|}$ to the indeterminates of T_G we denote the matrix by $T_G(w)$ where the indeterminates are replaced by the corresponding assignment in w. Then we have*

$$\max_w \{rank(T_G(w))\} = 2 \cdot |M|.$$

In order to calculate the maximum of the rank, Lovász [87] also showed that the rank of the matrix where the indeterminates are replaced by random numbers uniformly drawn from $\{1, \ldots, R\}$ is equal to $\max_w\{rank(T_G(w))\}$ with probability at least $1 - |E|/R$.

Theorem 2.2.28 (Lovász [87])**.** *Let $G = (V, E)$ be a graph and $r \in \mathbb{R}^{|E|}$ be a random vector where each coordinate is uniformly chosen from $\{1, \ldots, R\}$ with $R \geq |E|$. Then we have*

$$rank(T_G(r)) = \max_w \{rank(T_G(w))\}$$

with probability at least $1 - |E|/R$.

2.3. OBDD-Based Graph Representation and Algorithms

Let $G = (V, E)$ be a directed graph with node set $V = \{v_0, \ldots, v_{N-1}\}$ and edge set $E \subseteq V \times V$. Here, an undirected graph is interpreted as a directed symmetric graph. Implicit algorithms are working on the characteristic function $\chi_E \in B_{2n}$ of E where $n = \lceil \log N \rceil$ is the number of bits needed to encode a node of V and $\chi_E(x, y) = 1$ if and only if $(v_{|x|_2}, v_{|y|_2}) \in E$. The question is how χ_E can be represented in a compact way such that it is still possible to operate on this compact representation to solve classical graph problems.

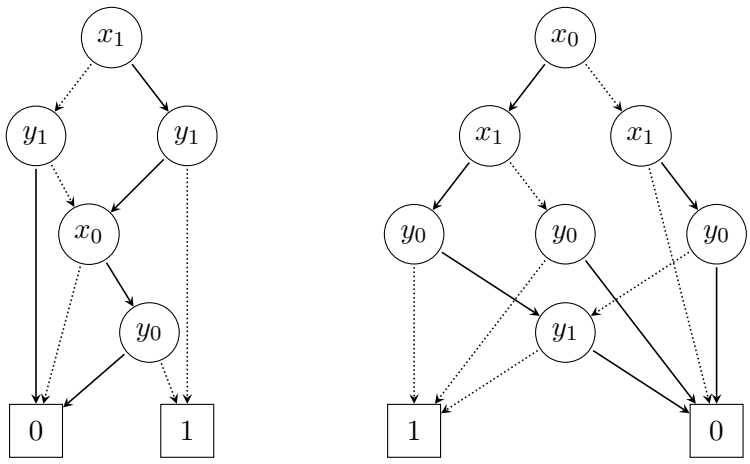


Figure 2.2.: Two π-OBDDs representing the function $GT(x, y)$ with $GT(x, y) = 1$ if and only of $|x|_2 > |y|_2$ with a good variable order $\pi = (x_1, y_1, x_0, y_0)$ and a bad variable order $\pi = (x_0, x_1, y_0, y_1)$.

The Data Structure: Ordered Binary Decision Diagrams

In order to deal with Boolean functions, *Ordered Binary Decision Diagrams* (OBDDs) were introduced by Bryant [25] to get a compact representation of not too few Boolean functions and also to support several functional operations efficiently.

Definition 2.3.1 (Ordered Binary Decision Diagram (OBDD))**.**

Order. A *variable order* π on the input variables $X = \{x_0, \ldots, x_{n-1}\}$ of a Boolean function $f \in B_n$ is a permutation of $\{0, \ldots, n-1\}$. We slightly abuse notation and often write π as a sequence of variables instead of indices (see, e. g., Fig. 2.2).

Representation. A *π-OBDD* is a directed, acyclic and rooted graph G with two sinks labeled by the constants 0 and 1. Each inner node is labeled by an input variable from X and has exactly two outgoing edges labeled by 0 and 1. Each edge (x_i, x_j) has to respect the variable order π, i. e., $\pi^{-1}(i) < \pi^{-1}(j)$. The π-OBDD is called *complete* if every variable is tested on each path, i. e., for every variable x_i there is a node labeled by x_i on each path.

Evaluation. An assignment $a \in \{0, 1\}^n$ of the variables defines a path from the root to a sink by leaving each x_i-node via the a_i-edge. A π-OBDD G represents f if for every $a \in \{0, 1\}^n$ the defined path ends in a sink with label $f(a)$.

Complexity. The *size* of a π-OBDD G, denoted by $size(G)$, is the number of nodes in G. The π-OBDD size of a function f is the minimum size of a π-OBDD representing f. The OBDD size of f is the minimum π-OBDD size over all variable orders π. The *width* of G is the maximum number of nodes labeled by the same input variable. The *OBDD width* (*complete OBDD width*) of f is the minimum width of an (complete) OBDD representing f.

In the following we describe a list of important operations on Boolean functions which we will

use in this thesis and give the time requirements in the case of OBDDs (see, e. g., Section 3.3 in [128] for a detailed list). Let π be a fixed variable order on the variable set $X = \{x_0, \ldots, x_{n-1}\}$. Let f and g be Boolean functions in B_n on the variables X and let G_f and G_g be π-OBDDs representing f and g, respectively.

1. **Negation:** Given G_f, compute a π-OBDD for the function $\overline{f} \in B_n$. Time: $\mathcal{O}(1)$
2. **Replacement by constant:** Given G_f, an index $i \in \{0, \ldots, n-1\}$, and a constant $c_i \in \{0, 1\}$, compute a π-OBDD for the subfunction $f_{|x_i=c_i}$ where the variable x_i is replaced by the constant c_i. Time: $\mathcal{O}(size(G_f))$
3. **Equality test:** Given G_f and G_g, decide whether f and g are equal. Time: $\mathcal{O}(1)$ in most implementations (when using so-called *Shared OBDDs*, see [128]), otherwise $\mathcal{O}(size(G_f) + size(G_g))$
4. **Satisfiability:** Given G_f, decide whether f is not the constant function 0. Time: $\mathcal{O}(1)$
5. **Satisfiability count:** Given G_f, compute $|f^{-1}(1)|$. Time: $\mathcal{O}(size(G_f))$
6. **Synthesis:** Given G_f and G_g and a binary Boolean operation $\otimes \in B_2$, compute a π-OBDD representing the function $h \in B_n$ defined as $h := f \otimes g$. Time: $\mathcal{O}(size(G_f) \cdot size(G_g))$
7. **Quantification:** Given G_f, an index $i \in \{0, \ldots, n-1\}$ and a quantifier $Q \in \{\exists, \forall\}$, compute a π-OBDD representing the function $h \in B_n$ defined as $h := Qx_i : f$ where $\exists x_i : f := f_{|x_i=0} \vee f_{|x_i=1}$ and $\forall x_i : f := f_{|x_i=0} \wedge f_{|x_i=1}$. Time: see replacement by constant and synthesis

In the rest of the thesis quantifications over k Boolean variables $Qx_1, \ldots, x_k : f$ are denoted by $Qx : f$, where $x = (x_1, \ldots, x_k)$. The following operation (see, e. g., [117]) is useful to reverse the edges of a given graph: For a directed graph $\chi_E(x, y)$ we want to compute $\chi_E(y, x)$ which represents the edge set $\{(v_{|y|_2}, v_{|x|_2}) \mid (v_{|x|_2}, v_{|y|_2}) \in E\}$ consisting of the reverse edges of E.

Definition 2.3.2. Let $k \in \mathbb{N}$, ρ be a permutation of $\{1, \ldots, k\}$ and $f \in B_{kn}$ be defined on Boolean variable vectors $x^{(1)}, \ldots, x^{(k)}$ of length n. The argument reordering $\mathcal{R}_\rho(f) \in B_{kn}$ with respect to ρ is defined by $\mathcal{R}_\rho(f)(x^{(1)}, \ldots, x^{(k)}) := f(x^{(\rho(1))}, \ldots, x^{(\rho(k))})$.

This operation can be computed by renaming the variables and repairing the variable order using $3(k-1)n$ functional operations [19].

Discussion: What are the Parameters of Interest for our Algorithms?

Since we are interested in designing algorithms using OBDDs and functional operations, we have to think about the running time of these algorithms. As we can see in the above listing

of operations, the running time of such algorithms depends on the actual size of the OBDDs which are used for a functional operation during the computation. Therefore, there are two main factors influencing the running time of such algorithms: the number of functional operations and the size of the OBDDs which are used for the operations. In general, it is difficult to prove a good upper bound on the running time because we have to know a good upper bound on the size of every OBDD used as an input for an operation which is very difficult except for very structured graphs [131, 20, 18]. Note that to enable the functional operations to be efficient we have to use the same variable order for the OBDDs. But computing the optimal variable order for a function is known to be NP-hard for both general OBDDs [21] and complete OBDDs [16]. In addition, if the size of the OBDD representing the input graph is large, any implicit algorithm using this OBDD is likely to have an inadequate running time. Beside the variable order, the labeling of the nodes (which is independent from the variable order) is another optimization parameter with huge influence on the input OBDD size. For OBDDs representing state transitions in finite state machines, Meinel and Theobald [97] showed that there can be an exponential increase of the OBDD size from a good labeling to a worst-case labeling. Nevertheless, a small input OBDD size, i. e., a good labeling of the nodes for some variable order, does not guarantee a good performance of the implicit algorithm since the sizes of the intermediate OBDDs do not have to be small. Indeed, an exponential blowup from input to output size is possible (see, e. g., [117, 15]).

It may seem that the OBDD size is more important than the number of functional operations. Unfortunately, that is not the case. There seems to be a trade-off: The number of operations is an important measure of difficulty [14] but decreasing the number of operations often results in an increase of the number of variables of the used functions. Since the worst case OBDD size is exponentially large in the input variables (see next subsection), the number of variables should be as small as possible. This trade-off was also empirically observed. For instance, an implicit algorithm computing the transitive closure that uses an iterative squaring approach and a polylogarithmic number of operations is often inferior to an implicit sequential algorithm, which needs in worst case a linear number of operations [14, 60].

Characterizing Minimal OBDDs and the OBDD Size of Basic Functions

A general upper bound of the π-OBDD size for every function $f \in B_n$ and every variable order is $(2 + o(1)) \cdot 2^n/n$ due to Breitbart et al. [24]. It is also easy to see that the π-OBDD size is bounded from above by $\mathcal{O}\left(n \cdot \min\{|f^{-1}(1)|, |f^{-1}(0)|\}\right)$. These bounds imply that the worst case OBDD size representing the characteristic function $\chi_E \in B_{2n}$ of the edge set of a graph is at most $\mathcal{O}(\min\{N^2/\log N, |E| \log N\})$.

Recall that $f_{|x_{\pi(0)}=a_{\pi(0)},\ldots,x_{\pi(i-1)}=a_{\pi(i-1)}}$ denotes the subfunction where $x_{\pi(j)}$ is replaced by the constant $a_{\pi(j)}$ for $0 \leq j \leq i-1$. The function f *depends essentially* on a variable x_i iff $f_{|x_i=0} \neq f_{|x_i=1}$. A characterization of minimal π-OBDDs due to Sieling and Wegener [120]

can often be used to bound the OBDD size.

Definition 2.3.3. Let $f, g \in B_n$. The functions f and g are *equal* if $f(x) = g(x)$ for all $x \in \{0,1\}^n$. Otherwise, they are called *different*. For a sequence of functions $(f_1, \ldots, f_l)$ for some $l \geq 1$, the *number of different functions* is the cardinality of the set $\{f_i \mid i \in \{1, \ldots, l\}\}$. Examples of such a sequence are the possible subfunctions of f where a fixed number of variables are replaced by constants or the functions represented by OBDDs of size at most s.

Theorem 2.3.4 ([120])**.** *Let $f \in B_n$ and for all $i = 0, \ldots, n-1$ let s_i be the number of different subfunctions that result from replacing all variables $x_{\pi(j)}$ with $0 \leq j \leq i-1$ by constants and which essentially depend on $x_{\pi(i)}$. Then the minimal π-OBDD representing f has s_i nodes labeled by $x_{\pi(i)}$, i. e., the minimal π-OBDD has size $\sum_{i=0}^{n-1} s_i$.*

If we can bound the width of complete π-OBDDs G_f and G_g, there is an easy bound on the size of the OBDD G_h resulting from the synthesis $h := f \otimes g$. This lemma is known by folklore (see, e. g., [131]).

Lemma 2.3.5. *Let $f, g \in B_n$ and let $\otimes$ be a Boolean operation. If there exist complete π-OBDDs for f and g which have on level i ($1 \leq i \leq n$) at most $s_{f,i}$ and $s_{g,i}$ nodes, respectively, then there exists a π-OBDD for $f \otimes g$ which has on level i at most $s_{f,i} \cdot s_{g,i}$ nodes.*

Some of our lower bounds also hold for so-called *Free Binary Decision Diagrams* (FBDDs) also known as *Read-Once Branching Programs* which were introduced by Masek [92] and are a more general computational model than OBDDs. In an FBDD every variable can also only be read once on a path from the root to a sink but the edges do not have to respect a global variable order. Lower bound techniques for FBDDs have to take into account that the order can change for different paths. The following property due to Jukna [70] can be used to show good lower bounds for the FBDD size.

Definition 2.3.6. A function $f \in B_n$ with input variables $X = \{x_0, \ldots, x_{n-1}\}$ is called *r-mixed* if for all $V \subseteq X$ with $|V| = r$ the 2^r assignments to the variables in V lead to different subfunctions.

Lemma 2.3.7 ([70])**.** *The FBDD size of a r-mixed function is bounded below by $2^r - 1$.*

An important variable order is the interleaved variable order which is defined on vectors of length n where the variables with the same significance are tested one after another.

Definition 2.3.8. Let $x^{(1)}, \ldots, x^{(k)} \in \{0,1\}^n$ be k input variable vectors of length n. Let π be a permutation of $\{0, \ldots, n-1\}$. Then

$$\pi_{k,n} = (x^{(1)}_{\pi(0)}, x^{(2)}_{\pi(0)}, \ldots, x^{(k)}_{\pi(0)}, \ldots, x^{(1)}_{\pi(n-1)}, \ldots, x^{(k)}_{\pi(n-1)})$$

is called *k-interleaved* variable order for $x^{(1)}, \ldots, x^{(k)}$. If $\pi = (n-1, \ldots, 0)$ then we say that the variables are tested with decreasing significance.

It is well known that the OBDD size of the equality function $EQ(x, y)$ and greater than function $GT(x, y)$ with $EQ(x, y) = 1 \Leftrightarrow |x|_2 = |y|_2$ and $GT(x, y) = 1 \Leftrightarrow |x|_2 > |y|_2$ is linear in the number of input bits for an interleaved variable order with decreasing significance (see, e. g., [128]) and it is also possible to construct the representing OBDDs for these functions in linear time. We illustrate this by means of another simple function, which we will use later on: The inner product $IP_n(x, y) = \bigoplus_{i=0}^{n-1} x_i \wedge y_i$ of two binary vectors $x, y \in \{0, 1\}^n$. Let π be a 2-interleaved variable order for x, y and w.l.o.g. let $\pi = (x_0, y_0, \ldots, x_{n-1}, y_{n-1})$. Define $b_j := \bigoplus_{i=0}^{j} x_i \wedge y_i$ for $0 \leq j \leq n-1$. If we know the value of b_j for some j, we can easily extend it to the value of b_{j+1} by reading the pair (x_{j+1}, y_{j+1}) since $b_{j+1} = b_j \oplus (x_{j+1} \wedge y_{j+1})$. Thus, the π-OBDD representing IP_n has size $\mathcal{O}(n)$ and width 2 (see Fig. 2.3). Notice that the π-OBDD size is also $\mathcal{O}(n)$ if we replace an input vector, e. g., y, by a constant vector $c \in \{0, 1\}^n$. Many such auxiliary functions which are often used in implicit algorithms, e. g., the equality or the greater than function, and multivariate threshold functions, which we will define in Section 3.3.1, have a compact representation for the interleaved variable order with decreasing significance. Thus, choosing this variable order in OBDD-based algorithms is common practice. Another advantage of an interleaved variable order is that the argument reordering operation from Definition 2.3.2 does not result in an increase of the representation size. Bollig [17] improved a result by Sawitzki [116] and showed the following.

Theorem 2.3.9 ([17])**.** *Let $n, k \in \mathbb{N}$, ρ be a permutation of $\{1, \ldots, k\}$ and let $f \in B_{kn}$ be defined on k Boolean variable vectors $x^{(1)}, \ldots, x^{(k)}$ of length n. Let $\pi_{k,n}$ be a k-interleaved variable order for $x^{(1)}, \ldots, x^{(k)}$ and G_f be a complete $\pi_{k,n}$-OBDD of width w representing f. Then a $\pi_{k,n}$-OBDD $G_{\mathcal{R}_\rho(f)}$ for the argument reordering $\mathcal{R}_\rho(f)$ can be constructed in time $\mathcal{O}(2^k wkn \log(2^k wkn))$ and space $\mathcal{O}(2^k wkn)$. Furthermore, the width of $G_{\mathcal{R}_\rho(f)}$ is bounded by $2^{k-1}w$.*

Lower Bounds for the Size of OBDDs Representing χ_E

As we have seen earlier in this chapter, the OBDD size is an important measure of complexity. Therefore, we also need some tool to prove lower bounds for the OBDD size of certain graph classes. In order to achieve this, Nunkesser and Woelfel [107] used a counting argument which works as follows: Wegener [128] showed that OBDDs of size s can represent at most $sn^s(s+1)^{2s}/s!$ different functions $f \in B_n$. It is also clear that each element in the set of unlabeled graphs of size N from a graph class $\mathcal{G}$ need to be represented by a different function. Thus, counting unlabeled graphs gives a lower bound on the size of OBDDs representing such graphs.

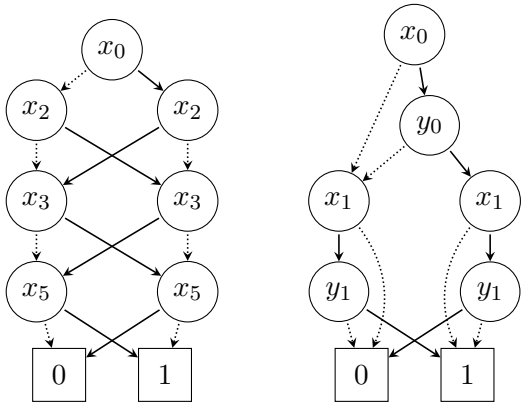


Figure 2.3.: Two π-OBDDs with $\pi = (x_0, y_0, \ldots, x_{n-1}, y_{n-1})$ for the functions $IP_6(x, y)$ where y is replaced by the constant vector $(1, 0, 1, 1, 0, 1)$ and $IP_2(x, y)$.

Lemma 2.3.10 ([107])**.** *Let $\mathcal{G}$ be a graph class and let $N_{\mathcal{G}}(N)$ be the number of unlabeled graphs of size N from $\mathcal{G}$. Let $s : \mathbb{N} \to \mathbb{R}$ be a function. If*

$$\lim_{N\to\infty} \frac{s2^s\lceil \log N\rceil^s(s+1)^{2s}}{s! \cdot N_{\mathcal{G}}(N)} < 1$$

then there are graphs of size $N_0 \in \mathbb{N}$ which cannot be represented by OBDDs of size $s(N_0)$ or less where the nodes are labeled by $\lceil \log N \rceil$ bits for every $N \in \mathbb{N}$.

Notice that a lower bound of the OBDD size of a graph class does not mean that the OBDD size of every graph from this class is bounded below by this value. In the following corollary we explicitly calculate the lower bounds for some typical values of $N_{\mathcal{G}}$.

Corollary 2.3.11. *The size of OBDDs representing graphs from a graph class $\mathcal{G}$ is bounded from below by*

$$\begin{aligned} \Omega(N/\log N) \quad & \text{if } N_{\mathcal{G}} = 2^{\Omega(N)} \text{ and} \\ \Omega(N) \quad & \text{if } N_{\mathcal{G}} = 2^{\Omega(N\log N)}. \end{aligned}$$

Proof. It is known that $\log s! \geq s\log(s/e)$. Thus, $s2^s\lceil \log N\rceil^s(s+1)^{2s}/s!$ can be bounded from above by $2^{s\log s + s\log\log N + \mathcal{O}(s)}$. If $N_{\mathcal{G}} = 2^{\Omega(N)}$ and $s(N) \leq c_1 N/\log N$ we have

$$2^{s\log s + s\log\log N + \mathcal{O}(s) - \Omega(N)} \leq 2^{c_1 N + o(N) - \Omega(N)}$$

which is smaller than 1 for large N and c_1 small enough. Therefore, the OBDD size is bounded from below by $\Omega(N/\log N)$. Similarly, for $N_{\mathcal{G}} = 2^{\Omega(N\log N)}$ and $s(N) \leq c_2 N$ we have

$$2^{s\log s + s\log\log N + \mathcal{O}(s) - \Omega(N\log N)} \leq 2^{c_2 N\log N + \mathcal{O}(N\log\log N) - \Omega(N\log N)}$$

Algorithm 1 *TransitiveClosure*$(R(x,y))$

Input: Boolean function $R(x,y) \in B_{2n}$
Output: Transitive closure $R^*(x,y)$ of $R(x,y)$

$R^*(x,y) = R(x,y)$
for $i = 0$ to n **do**
  $R^*(x,y) = \exists z : R^*(x,z) \wedge R^*(z,y)$
end for
return $R^*(x,y)$

which is smaller than 1 for large N and c_2 small enough which implies a lower bound of $\Omega(N)$. □

Warm Up: Basic Implicit Algorithms

Now we have the foundations to start with some simple OBDD-based algorithms. For the sake of code readability, we use $|x|_2 = |y|_2$ and $|x|_2 > |y|_2$ to denote $EQ(x,y)$ and $GT(x,y)$ in our algorithms. Furthermore, by $|x|_2 > c$ ($|x|_2 = c$) for some constant c we denote the function $GT(x,y)_{|y=[c]_2}$ ($EQ(x,y)_{|y=[c]_2}$) where the y-variables are replaced by constants corresponding to the binary number $[c]_2$ of c.

We slightly abuse the notation and denote by $f(x,y)$ the function over the input variables x, y and not the evaluation of the function on input x and y. For instance, $R^*(x,y) = \ldots$ in Algorithm 1 means that we assign a new function to $R^*(x,y)$ and not just updating one function value. Now, let $R(x,y) \in B_{2n}$ be a Boolean function. $R(x,y)$ can be seen as a binary relation R on the set $\{0,1\}^n$ with $x\,R\,y \Leftrightarrow R(x,y) = 1$. The *transitive closure* of this relation is the function $R^*(x,y)$ with $R^*(x,y) = 1$ if and only if there is a sequence $x = x_1, \ldots, x_l = y$ with $R(x_i, x_{i+1}) = 1$ for all $i = 1, \ldots, l-1$. For instance, let $R(x,y) = \chi_E(x,y) \vee (|x|_2 = |y|_2)$ be the function that returns 1 if and only if there is an edge between $v_{|x|_2}$ and $v_{|y|_2}$ or $|x|_2 = |y|_2$. Then it is $R^*(x,y) = 1$ iff the nodes $v_{|x|_2}$ and $v_{|y|_2}$ are in the same connected component. The transitive closure can be computed implicitly by $\mathcal{O}(n^2)$ functional operations using the so-called iterative squaring or path doubling technique (see Algorithm 1).
Let $O(x,y)$ represent a total order $\prec$ on the input bit strings, i. e., $O(x,y) = 1 \Leftrightarrow x \prec y$ (e. g., $O(x,y) = 1 \Leftrightarrow |x|_2 \leq |y|_2$). Since $\prec$ is a total order, the input bit strings can be sorted in ascending order according to $\prec$. Let $EO(x,l) = 1$ iff x is in the $|l|_2$-th position in this sorted sequence. Similar to the transitive closure, it is known (see, e. g., [118]) that $EO(x,l)$ can be computed using $\mathcal{O}(n^2)$ functional operations (see Algorithm 2).

Implicit Matching Algorithms

The implicit maximum flow algorithm by Hachtel and Somenzi in [57] uses the following maximal matching heuristic: The general idea is to iteratively add edges to the current

Algorithm 2 *EnumerateOrder*$(O(x,y))$

Input: Total order $O(x,y) \in B_{2n}$
Output: $EO(x,l)$ with $EO(x,l) = 1$ iff the rank of x is $|l|_2$ in the ascending order

$\triangleright$ Compute the pairs of direct successors
$DS(x,y) = O(x,y) \wedge \overline{\exists z : O(x,z) \wedge O(z,y)}$
$\triangleright$ $EO_i(x,y,l) = 1$ iff $|l|_2 \leq 2^i$ and the distance between the position of x
$\triangleright$ and y is equal to $|l|_2$
$EO_0(x,y,l) = ((|l|_2 = 0) \wedge (|x|_2 = |y|_2) \vee ((|l|_2 = 1) \wedge DS(x,y)))$
$\triangleright$ Divide and conquer approach: If $2^{i-1} < |l|_2 \leq 2^i$ then there has to be
$\triangleright$ an intermediate bit string z with distance 2^{i-1} to x and $|l|_2 - 2^{i-1}$ to y
for $i = 1$ to n **do**

$$EO_i(x,y,l) = ((|l|_2 \leq 2^{i-1}) \wedge EO_{i-1}(x,y,l)) \vee [(2^{i-1} < |l|_2 \leq 2^i) \wedge \exists l_1, z : EO_{i-1}(x,z,2^{i-1}) \wedge EO_{i-1}(z,y,l_1) \wedge (|l_1|_2 + 2^{i-1} = |l|_2)]$$

end for
$\triangleright$ Compute the rank according to the distance to the first element
$EO(x,l) = \exists z : EO_n(z,x,l) \wedge \overline{\exists z' : O(z',z)}$
return $EO(x,l)$

matching until this matching is maximal. A building block of this heuristic is a priority function which is used to break symmetries and selects edges in parallel which are added to the current matching (see Algorithm 3). The priority function $\Pi(x,y,z) : \{0,1\}^{3n} \to \{0,1\}$ is defined by $\Pi(x,y,z) = 1$ iff $y \prec_x z$ where $\prec_x$ is a total order of $\{0,1\}^n$ for every $x \in \{0,1\}^n$. It is possible that $\prec_x$ does not depend on x at all and is always the same total order.

Hachtel and Somenzi use two different priority functions: $\Pi_1(x,y,z) = 1$ iff $y < z$ and $\Pi_2(x,y,z) = 1$ iff $|y \oplus x|_2 < |z \oplus x|_2$. Since Π_2 makes use of the possibility to choose different orders based on x, it is more likely that the chosen edges have different endpoints and the number of iterations is smaller than with Π_1. But in general, if many nodes choose the same node in a lot of iterations, then the heuristic has a poor performance. For example, using Π_1 on a complete bipartite graph results in $\Omega(N)$ iterations because only $\mathcal{O}(N)$ edges are deleted at the end of the loop. Nevertheless, the number of variables on which a function in this algorithm depends is independent of the underlying priority function and is at most $3 \log N$.

In contrast to this heuristic, Bollig and Pröger [20] gave a OBDD-based maximal matching algorithm using $\mathcal{O}(\log^4 N)$ functional operations. However, the number of variables increases to $6 \log N$. This algorithm is based on the parallel maximal matching algorithm by Kelsen [78]. Note that the number of functional operations was decreased at the cost of an increase in the number of used variables. In order to improve the quality of the maximal matching, the authors also paired the heuristic by Hachtel and Somenzi with a strategy of Karp-Sipser [77] where edges incident to nodes with degree 1 are always added to the matching. Whenever there is no such node, a random edge is added to the matching. Instead of selecting a single random edge, Bollig and Pröger decided to do one round of the Hachtel and Somenzi heuristic.

In terms of maximum matching, Bollig, Gillé, and Pröger [18] gave two implicit algorithms:

Algorithm 3 Implicit maximal matching algorithm [57]

Input: χ_A, χ_B, χ_E of a bipartite Graph $G = (A, B, E)$ with $V = A \cup B$
Output: χ_M where M is a maximal matching

$\chi_M(x, y) = 0$
$\triangleright$ Direct the edges from A to B
$\chi_E(x, y) = \chi_E(x, y) \wedge \chi_A(x) \wedge \chi_B(y)$
while $\chi_E(x, y) \neq 0$ **do**
  $\triangleright$ For every $x \in A$ choose smallest outgoing edge according to $\Pi(x, y, z)$
  $P(x, y) = \chi_E(x, y) \wedge \overline{\exists z : \chi_E(x, z) \wedge \Pi(x, z, y)}$
  $\triangleright$ For every $y \in B$ choose smallest incoming edge according to $\Pi(x, y, z)$
  $Q(x, y) = P(x, y) \wedge \overline{\exists z : P(z, y) \wedge \Pi(y, z, x)}$
  $\triangleright$ Add edges to current matching
  $\chi_M(x, y) = \chi_M(x, y) \vee Q(x, y)$
  $\chi_M(x, y) = \chi_M(x, y) \vee \chi_M(y, x)$
  $Matched(x) = \exists y : \chi_M(x, y)$
  $\triangleright$ Delete edges incident to matched nodes
  $\chi_E(x, y) = \chi_E(x, y) \wedge \overline{Matched(x)} \wedge \overline{Matched(y)}$
end while
return $\chi_M(x, y)$

One is a classic augmenting path based algorithm like the algorithm by Hopcroft and Karp and the other is based on a so-called push-relabel technique. The authors also wanted to minimize the number of functional operations and showed that the algorithms need $\mathcal{O}(N^{3/4} \log^{5/2} N)$[1] and $\mathcal{O}(N^{2/3} \log^{3.375} N)$[1] functional operations, respectively, and the number of variables is $\max\{3 \log N, MV(N)\}$ and $\max\{4 \log N, MV(N)\}$, respectively, where $MV(N)$ denotes the number of variables used by the maximal matching subroutine.

Table 2.1 summarizes the implicit matching algorithms regarding functional operations and numbers of variables. The dependency on the maximal matching subroutine for the algorithms from [18] is also highlighted. All the aforementioned algorithms need a bipartite graph as input. We note that all maximal matching algorithms working on bipartite graphs can be generalized to arbitrary graphs by decomposing the graph into a logarithmic number of bipartite graphs [78] resulting in a logarithmic blowup in the number of functional operations.

Implicit Algorithms and Parallel Algorithms

Known ideas or techniques from parallel algorithms are often used in implicit algorithms with a small number of functional operations. For an overview on the theory of parallel algorithms, we refer to the book by Greenlaw, Hoover, and Ruzzo [54]. Sawitzki [116, 118] showed that all problems which are decidable by polynomially many processors using polylogarithmic time (i. e., which are in the complexity class NC) are computable by an implicit algorithm using a polylogarithmic number of functional operations on a logarithmic number of Boolean

[1] They used the algorithm from [20] as the maximal matching subroutine.

	Functional Operations	Variables
Maximal Matching		
Hachtel, Somenzi [57]	$\mathcal{O}(N \log N)$	$3 \log N$
Bollig, Pröger [20]	$\mathcal{O}(\log^4 N)$	$6 \log N$
Karp-Sipser Variant [20]	$\mathcal{O}(N \log N)$	$3 \log N$
Maximum Matching		
Bollig, Gillé, Pröger [18]	$\tilde{\mathcal{O}}(N^{1/2+c} \cdot MO(N) + N^{1-c})$	$\max\{3 \log N, MV(N)\}$
Bollig, Gillé, Pröger [18]	$\tilde{\mathcal{O}}(N^{2/3} \cdot MO(N))$	$\max\{4 \log N, MV(N)\}$

Table 2.1.: Comparison of the implicit matching algorithms with respect to the number of functional operations and the number of variables. Here, $MV(N)$ is the number of variables and $MO(N)$ is the number of operations used by the implicit maximal matching subprocedure.

variables. This is a structural result and does not lead to either an efficient transformation of parallel algorithms into implicit algorithms or a guarantee that implicit algorithms using a polylogarithmic number of functional operations perform well in practice (as seen in the discussion about the transitive closure computation). We will further discuss this result in Chapter 4 where we introduce randomization in implicit algorithms.

2.4. Probability Spaces and Random Variables

Randomization is a powerful tool for designing efficient algorithms. Usually the number of used random bits is not restricted. However, in certain cases a small representation of the used random bits is necessary, e. g., for derandomization [3, 89], hashing [26, 40] or in space bounded computation models such as streaming [6]. Let $X_0, \ldots, X_{m-1}$ be binary random variables. Here, we focus on binary random variables but the definitions can be easily extended to general random variables with finite range. Formally, a binary random variable is a function $X : S \to \{0, 1\}$ where S is a set of possible outcomes of a random experiment. S is also called sample space. This means for a definition of a random variable, we have to assign a value $X(s)$ to every possible input $s \in S$. The randomness is now given by a probability distribution over the sample space that assigns a probability $\mathbf{Pr}[s]$ to every $s \in S$. We write $\mathbf{Pr}[X = i]$ for probability of the event that $X(s) = i$, i. e., formally it is $\mathbf{Pr}[\{s \mid X(s) = i\}]$.

If $X_0, \ldots, X_{m-1}$ are completely independent, then we have no chance to get a better representation than storing them all because there are 2^m possible outcomes. Therefore, we need some kind of limited independence to be able to get a succinct representation of our random variables.

Definition 2.4.1 ((Almost) k-wise Independence)**.** Let $X_0, \ldots, X_{m-1}$ be binary random variables. These variables are called *k-wise independent* with $k \leq m$ if and only if for all $0 \leq i_1 < \ldots i_k \leq m-1$ and for all $l_1, \ldots, l_k \in \{0,1\}$

$$\mathbf{Pr}\left[X_{i_1} = l_1 \wedge \ldots \wedge X_{i_k} = l_k\right] = 2^{-k}$$

and they are called *(ε, k)-wise independent* iff

$$\left|\mathbf{Pr}\left[X_{i_1} = l_1 \wedge \ldots \wedge X_{i_k} = l_k\right] - 2^{-k}\right| \leq \varepsilon.$$

If m is a power of 2, the random variables can be seen as function values of a Boolean function. Formally, we define the notion of a random function whose OBDD size is investigated in Chapter 4.

Definition 2.4.2 ((Almost) k-wise Independent Function)**.** For $r, n \in \mathbb{N}$ let $S = \{0,1\}^r$ and $m = 2^n$. A random function $f : S \to B_n$ maps an element s from the sample space S which is drawn uniformly at random to a Boolean function $f(s)$. We will use the notation f_s for the Boolean function $f(s)$.

A random function $f : S \to B_n$ is called *k-wise ((ε, k)-wise) independent* if the random variables $X_0(s) := f_s(0^n), \ldots, X_{m-1}(s) := f_s(1^n)$ are k-wise ((ε, k)-wise) independent.

In the next subsection, we give constructions of (almost) k-wise independent random variables. But first, we want to present results to bound sums of dependent random variables. Let $X = \sum_{i=0}^{m-1} X_i$ be a sum of k-wise independent variables with $k \geq 2$. We can easily verify that $\mathbf{Var}[X] = \sum_{i=0}^{m-1} \mathbf{Var}[X_i]$: The *covariance* $\mathbf{Cov}[Y, Z]$ between two random variables Y and Z is defined by $\mathbf{E}[Y \cdot Z] - \mathbf{E}[Y] \cdot \mathbf{E}[Z]$. If Y and Z are independent, then the covariance is 0. It is known that $\mathbf{Var}\left[\sum_{i=0}^{m-1} X_i\right] = \sum_{i=0}^{m-1} \mathbf{Var}[X_i] + 2 \cdot \sum_{i=0}^{m-1} \sum_{j>i} \mathbf{Cov}[X_i, X_j]$ (see, e. g., [101]) where the sum of covariances is 0 if the variables are pairwise independent. Therefore, if we can bound the variance of every X_i we can use Chebyshev's inequality to bound X.

Theorem 2.4.3 (Chebyshev)**.** *Let $X = \sum_{i=0}^{m-1} X_i$ be a sum of k-wise independent variables with $k \geq 2$. Then it is*

$$\mathbf{Pr}\left[|X - \mathbf{E}[X]| \geq \delta\right] \leq \frac{\mathbf{Var}[X]}{\delta^2} = \frac{\sum_{i=0}^{m-1} \mathbf{Var}[X_i]}{\delta^2}.$$

Another kind of limited independence is the so-called *negatively correlation* of random variables.

Definition 2.4.4 (Negatively Correlated Variables)**.** Boolean random variables $X_0, \ldots, X_{m-1}$ are called *negatively correlated* if for every subset $S \subseteq \{0, \ldots, m-1\}$ it holds

$$\mathbf{Pr}\left[\wedge_{i \in S} X_i = 1\right] \leq \prod_{i \in S} \mathbf{Pr}\left[X_i = 1\right].$$

Fortunately, negative correlation does not increase the distribution tails of the sum of the variables compared to the case of complete independence. Therefore, Panconesi and Srinivasan [110] were able to extent the famous Chernoff-Hoefding bounds to negatively correlated random variables. This gives us a strong tool to bound deviation from expectation.

Theorem 2.4.5 ([110]). *If the random variables $X_0, \ldots, X_{m-1} \in \{0,1\}$ are negatively correlated, then for $X = \sum_{i=0}^{m-1} X_i$, and $0 < \delta < 1$ we have*

$$\mathbf{Pr}\left[X < (1-\delta)\mathbf{E}\left[X\right]\right] \leq e^{\delta^2 \mathbf{E}[X]/2}.$$

Constructions of k-wise Independent Random Variables

The so-called BCH scheme introduced by Alon, Babai, and Itai [3] is a construction of k-wise independent random variables $X_0, \ldots, X_{2^n-1} \in \{0,1\}$ that only needs $\lfloor k/2 \rfloor n+1$ independent random bits and works as follows: Let $r_0 \in \{0,1\}$ be a random bit, $r^{(j)} \in \{0,1\}^n$ for $1 \leq j \leq l$ with $l \in \mathbb{N}$ be uniformly random row vectors, and the row vector $r = \left[r_0, r^{(1)}, \ldots, r^{(l)}\right] \in \{0,1\}^{ln+1}$ be the concatenation of the vectors. For $1 \leq i \leq 2^n - 1$ define

$$X_i = IP_{ln+1}\left(r, \left[1, [i]_2, [i]_2^3, \ldots, \left[i^{2l-1}\right]_2\right]\right)$$

where i^{2j-1} for $j = 1, \ldots, l$ is computed in the finite field $GF(2^n)$. This scheme generates $(2l+1)$-wise independent random bits [3]. This construction is optimal in the sense that it is known that the construction of k-wise independent random variables needs at least $\log\left(\sum_{i=1}^{\lfloor k/2 \rfloor} \binom{2^n}{i}\right) \approx nk/2$ random bits [30]. For constant k this means that the size of the sample space is polynomial in the number of random variables which is particularly useful in derandomization. In order to reduce the number of random bits even more and in particular get a polynomially large sample space for $k \in \mathcal{O}(n)$, Naor and Naor [104] introduced the notion of (ε, k)-wise independence and gave a construction together with Alon et al. [4] which uses only $\mathcal{O}(\log n + \log k + \log \frac{1}{\varepsilon})$ random bits. Savický [114] focused on a simple representation of the variables and gave a random Boolean formula of size $\mathcal{O}(n \log^2 k \log \frac{1}{\varepsilon})$ and depth 3 representing (ε, k)-wise independent random variables.

Instead of minimizing the number of random bits, another line of research focuses on the trade-off between the evaluation time of one random variable, i. e., the time needed to compute the value of one random variable, and the space requirement. Siegel [119] showed that if a k-wise independent function $h : \mathbb{F} \to \mathbb{F}$ with finite field $\mathbb{F}$ needs evaluation time $t < k$ then the space usage is at least $|\mathbb{F}|^{1/t}$. Thus, for constant time and non constant k every function based construction needs at least $|\mathbb{F}|^{\delta}$ space for some constant $\delta > 0$. However, Christiani and Pagh [31] circumvent this trade-off by looking for so-called k-wise independent generators which produce random variables one after another. This implies that they allow only sequential access to the sequence of random variables instead of random access as in Siegel's lower bound. Table 2.2 summarizes known constructions of (almost) k-wise independent random

Construction	Time	Space	Seed Length	Range	Comment
k-wise RVs					
BCH [3]	$\mathcal{O}(k)$	$\mathcal{O}(k)$	$\lfloor k/2 \rfloor + 1$	$\{0,1\}$	
Polynomials [67, 129]	$\mathcal{O}(k)$	$\mathcal{O}(k)$	k	$\mathbb{F}$	
Expander Hashing [119]	$\mathcal{O}(1)$	$\mathcal{O}(2^{\varepsilon n})$	$\mathcal{O}(k)$	$\mathbb{F}$	Probabilistic
Exp. + Multipoint [31]	$\mathcal{O}(1)$	-	$\tilde{\mathcal{O}}(k)$	$\mathbb{F}$	Sequential access
(ε, k)-wise RVs					
Several Methods [104, 4]	$\mathcal{O}(k)$	-	$\mathcal{O}(\log \frac{nk}{\varepsilon})$	GF(d)	d prime
Boolean Formula [114]	$\mathcal{O}(1)$	-	$\mathcal{O}(n \log^2 k \log \frac{1}{\varepsilon})$	$\{0,1\}$	Depth 3, parallel time

Table 2.2.: Besides the Boolean formula, all constructions operate in a finite field $\mathbb{F}$ (which is not necessarily identical to the range) with $|\mathbb{F}| \geq 2^n$ and it is assumed that arithmetic operations in finite fields can be done in constant time. In the case of k-wise random variables, seed length and space is measured in the number of elements of $\mathbb{F}$, i. e., a factor $\log |\mathbb{F}|$ away from the number of bits. A hyphen (-) in the space column means that the space usage is in the same order of magnitude than the seed length.

variables.

2.5. Dynamic Graph Streaming

Another approach to deal with massive graphs is to process the graph in the data stream model where for instance the edges arrive one by one and only a limited amount of memory can be used. Henzinger, Raghavan, and Rajagopalan [59] were the first considering graph problems in the streaming model. Typically the space usage should be sublinear with respect to the input size but most of the recent work designing streaming algorithms on graphs use the semi-streaming model [45, 103] where $\mathcal{O}(N \operatorname{polylog} N)$ space is permitted. This has become the model of choice because it turned out that most problems are intractable if only sublinear space is allowed [59] whereas many problems can be solved using almost linear space (with respect to the number of nodes). For an overview of graph streaming algorithms, e. g., for connectivity or minimum spanning tree, we refer to the recent survey by McGregor [95]. It is also possible to investigate a stream where edges can not only be inserted but also deleted. More formally, the so-called dynamic graph streams, which were introduced by Ahn, Guha, and McGregor [2], are defined as follows.

Definition 2.5.1 (Dynamic Graph Stream)**.** A dynamic graph stream of a weighted, undirected graph $G = (V, E)$ consists of a sequence $s_1, s_2, \dots s_t$ of edge updates to any initially empty graph, where $s_i \in (V \times V, \mathbb{R}^+, \{-1, 1\})$ and -1 and 1 signify deletion and insertion. A stream is called *consistent* if there exists a deletion operation $(e, w, -1)$ in between any two insertion operations $(e, w, 1)$ and no deletion operation $(e, w, -1)$ of an edge occurs if (e, w) is not contained in the current set of edges.

We always assume our input streams to be consistent. This model is similar to, but weaker than, the full turnstile update model where the input is a vector x and the stream consists of additive updates of the vector's entries. The difference is that while in the turnstile update model the weights of an edge can change arbitrarily with each update, in this model an edge (e, w) must be first deleted $(e, w, -1)$ before getting reinserted with the desired weight.

Of course, problems in the dynamic graph stream can be much harder than in the insertion-only model and, indeed, algorithms for dynamic graph streams need completely new techniques. Graph sketching [2] utilizes the well studied theory on random linear projections which arises in the context of dimensionality reduction, e. g., the famous Johnson-Lindenstrauss Lemma [68] and estimating statistics in data streams [103]. We say that a sketch is linear if the sketches of two vectors or matrices can be added to get the sketch of the sum of the vectors or matrices. Due to the linearity of the sketches mentioned above, it is mainly a matter of addition and subtraction of sketches to deal with insertions and deletions of edges. But nothing comes without a price. In the remaining section we cover the basic ideas and strengths of graph sketching but also show where the main obstacles lie.

The general approach of graph sketching is to summarize edge sets via a small linear sketch. These edge sets typically consist either of the entire set of edges, for instance by interpreting the adjacency matrix as an N^2-dimensional vector, or the neighborhood of a single node, that is a column or row of the adjacency matrix. The main goal of the graph sketches is to sample edges uniformly at random from the summarized set of edges. For instance, our algorithms in Chapter 6 require randomly chosen edges and randomly chosen neighbors. Such random edges can be maintained in a stream under edge deletions by employing so-called ℓ_0 sampling. The ℓ_p-norm of a d-dimensional vector $x = (x_1, \dots, x_d) \in \mathbb{R}^d$ is defined as $\ell_p(x) = \left(\sum_{i=1}^{d} |x_i|^p\right)^{1/p}$. For $p = 0$, we have $\ell_0(x) = |\{i : x_i \neq 0\}|$ (if we define $1/0$ to be 1) and define $\ell_0^0(x) := \ell_0(x)$.

Definition 2.5.2 (ℓ_p-Sampling)**.** An (ϵ, δ) ℓ_p-sampler for non-zero $x \in \mathbb{R}^d$ returns $\perp$ with probability at most δ and otherwise returns a random $i \in [d]$ with probability in the range

$$\left[\frac{(1-\epsilon)|x_i|^p}{\ell_p^p(x)}, \frac{(1+\epsilon)|x_i|^p}{\ell_p^p(x)}\right].$$

An algorithm for ℓ_0-sampling is due to Jowhari, Saglam, and Tardos [69].

Lemma 2.5.3. *Let $\delta > 0$. There exist linear sketch based algorithms for $(0, \delta)$ ℓ_0-sampling of a vector $x \in \mathbb{R}^d$ using $\mathcal{O}(\log^2 d \cdot \log(1/\delta))$ bits of space.*

Since we focus on matching algorithms, it might now be tempting to consider the following maximal matching algorithm for a dynamic graph stream: We summarize the neighborhood of each node via an ℓ_0-sampler and attempt to perform a local parallel matching algorithm due to Israeli and Itai [66]. The main idea is to compute matchings consecutively on the subgraph induced by the free nodes and subsequently delete all edges incident to matched nodes. There are two problems with this algorithm: First, multiple queries to the same sketch of a node's neighborhood do not yield independent adjacent nodes and while linear sketches may be combined through addition, it is not possible to use a small space-sketch to query an entry and subsequently delete it to query the remaining entries. Essentially this limits us to querying every ℓ_0-sampler at most once and the space complexity of our algorithms is roughly the number of randomly chosen edges. Second, how can we obliviously maintain the set of edges contained in an induced subgraph? The strength of the sketches as pointed out, e. g., in [2] is to sample edges from any cut of the graph. A cut defined by a partitioning of the nodes $(A, V \setminus A)$ for $A \subseteq V$ is the set of edges that are incident to a node in S and a node $V \setminus A$. Now, consider the following neighborhood vector $a_v \in \mathbb{R}^{N^2}$ for each node $v \in V$:

$$a_{v,(u,w)} = \begin{cases} 1 & \text{if } u = v \text{ and } \{v, w\} \in E \\ -1 & \text{if } w = v \text{ and } \{v, u\} \in E \\ 0 & \text{otherwise} \end{cases}$$

Now, let $(A, V \setminus A)$ define a cut in the graph. Then $\sum_{v \in A} a_v \in \mathbb{R}^{N^2}$ is a vector whose non-zero entries correspond to the edges in the cut and whose sketch can be easily computed if we have linear sketches of every a_v. The entries corresponding to edges in the subgraph induced by A cancel out such that the edges in the cut remain. However, this trick can not be used to get a sketch of the subgraph induced by a set of nodes.

Beside sampling of edges, sketches can also be used to recover a graph if the graph is sparse enough.

Definition 2.5.4 (k-Sparse Recovery Sketch)**.** A k-sparse recovery sketch of a vector $x \in \mathbb{R}^d$ can recover $\min\{k, \ell_0(x)\}$ non-zero entries of x such that each sampled entry is chosen uniformly at random.

Here, it is sufficient to think of a k-sparse recovery sketch as $\mathcal{O}(k \operatorname{polylog} k)$ ℓ_0-samplers requiring $\mathcal{O}(k \operatorname{polylog} k \operatorname{polylog} d)$ space. For an overview of recovery sketches, we refer to the work of Barkay, Porat, and Shalem [12].

As stated in Theorem 2.2.27 and 2.2.28, the rank of the Tutte matrix, where the indeterminates are replaced by random values from $\{1, \ldots, R\}$, is equal to twice the size of a maximum matching with probability at least $1 - |E|/R$. In general, calculating the rank of an $N \times N$ matrix in a dynamic data stream is hard and requires at least $\Omega(N^2)$ space [33]. However, the following weaker rank decision problem can be decided in a stream using $\mathcal{O}(k^2 \log^2 \frac{N}{\delta})$

space with probability at least $1 - \delta$ [33].

Definition 2.5.5 (Rank Decision)**.** Given a positive integer k and a stream over updates to a matrix A, an algorithm for the rank decision problem outputs 1 if $\text{rank}(A) \geq k$ and 0 otherwise.

Replacing the indeterminates independently by random values needs $\Omega(N^2)$ random bits which would be infeasible for a graph streaming algorithm. Nisan [106] showed that any randomized algorithm that uses space S (excluding the random bits) and R completely independent random bits can be modified to a $\mathcal{O}(S \log R)$ space algorithm using $\mathcal{O}(S \log R)$ completely independent random bits. The difference of the probability of correctness between the two algorithms is negligible (exponentially small in S).

Theorem 2.5.6 (Nisan PRG [106], informal)**.** *Any randomized algorithm running in space S and using R random bits may be converted into one that uses only $\mathcal{O}(S \log R)$ random bits and runs in space $\mathcal{O}(S \log R)$.*

We cannot directly apply this result to a streaming algorithm maintaining the Tutte matrix. Since each entry of the matrix can be arbitrarily updated over the stream, we have to use the same random value each time we update an entry. This would not only result in $\Omega(N^2)$ random bits but also $\Omega(N^2)$ space. However, if the random values are aggregated by an operation that is commutative, then we can apply Nisan's result on a sorted stream, where the entries belonging to a random value occurred en bloc, to get a streaming algorithm which works on a sorted stream with small amount of randomness. Since the operation is commutative the output of the algorithm is the same on an unsorted stream. This idea was formalized and proven by Indyk [65] and we rephrase it here for dynamic graph streams and matrices.

Theorem 2.5.7 ([65])**.** *Let S be a stream of edge insertions and deletions of the form $s_i \in D := (V \times V, \{-1, 1\})$ and $f : D \times \{0,1\}^L \to \mathbb{R}^{d \times d'}$ be a function with $L, d, d' \in \mathbb{N}$. Let $\mathcal{A}$ be an algorithm that does the following:*

1. *Initialize O by the $d \times d'$ matrix containing only 0 and initialize $w_{i,j} \in \{0,1\}^L$ with L independent random bits for all $1 \leq i, j \leq N$ and $i \neq j$.*
2. *For each update $((i,j), u)$ set $O = O + f(((i,j), u), w_{i,j})$.*
3. *Output $A(S) = O$.*

Note that $\mathcal{A}$ uses $\mathcal{O}(L \cdot N^2)$ random bits overall. Assume that the entries of O can be stored using B bits and that f can be computed by an algorithm using $\mathcal{O}(C + L)$ space. Then there is an algorithm $\mathcal{A}'$ outputting $\mathcal{A}'(S)$ using only $\mathcal{O}((C + L + d \cdot d' \cdot B) \cdot \log(LN))$ random bits and space and it holds $\mathcal{A}(S) = \mathcal{A}'(S)$ with high probability.

The rank decision algorithm by Clarkson and Woodruff [33] maintains a $k \times k$ matrix $M = H' \cdot A \cdot H''$ where A is the $N \times N$ input matrix and H' and H'' are some appropriately chosen

random matrices. Since this is a linear operation on A, updating an entry of A can be modeled by the function f as in Theorem 2.5.7. Clarkson and Woodruff showed that each entry of M can be stored using $\mathcal{O}(\log(N/\delta))$ bits if the entries in A need $\mathcal{O}(\log N)$ bits and each update can be computed in $\mathcal{O}(k^2 \log(N/\delta))$ space. The number of random bits is $\mathcal{O}(k \log(N/\delta))$. Combining everything, we get the following theorem.

Theorem 2.5.8. *Let S be a dynamic graph stream of a graph $G = (V, E)$. There is an $\mathcal{O}(k^2 \log^2 N)$ space streaming algorithm that can decide whether the maximal rank of the Tutte matrix T_G is at least k with high probability.*

Proof. In Theorem 2.2.28 we choose each entry of the random assignment $r \in \mathbb{R}^{|E|}$ uniformly at random from $\{1, \ldots, N^3\}$ such that $\operatorname{rank}(T_G(r)) = \max_w\{\operatorname{rank}(T_G(w))\}$ with probability at least $1 - |E|/N^3 \geq 1 - 1/N$. Using Theorem 2.5.7 with the rank decision algorithm by Clarkson and Woodruff [33], i. e., $L = \mathcal{O}(k \log(N/\delta))$, $C = \mathcal{O}(k^2 \log(N/\delta))$, $d = d' = k$, and $B = \mathcal{O}(\log(N/\delta))$, we get a streaming algorithm that can decide in $\mathcal{O}(k^2 \log^2(N/\delta))$ space whether $\operatorname{rank}(T_G(r)) \geq k$ with probability at least $1 - \delta$. Setting $\delta = 1/N$ gives the desired result. □

Another useful streaming result from numerical linear algebra is due to Kane, Nelson, and Woodruff [72] who showed that $\ell_0(x)$ of a N-dimensional vector x can be $(1 \pm \varepsilon)$ estimated in dynamic streams using $\mathcal{O}(\frac{1}{\varepsilon^2} \log N(\log \frac{1}{\varepsilon} + \log\log(M)))$ space, where M is the sum of absolute values of the vector in the stream. We assume $M \leq 2^{N^c}$ with a constant c for graph streams, resulting in $\mathcal{O}(\frac{1}{\varepsilon^2} \log^2 N)$ bits of space.

Theorem 2.5.9 ([72])**.** *Let S be a dynamic graph stream of a graph $G = (V, E)$. Let the length of S be bounded by 2^{N^c}. Let $x \in \mathbb{R}^N$ be a vector whose entries are arbitrarily updated on an edge insertion/deletion and every coordinate x_i is polynomially bounded in N. Then there is an algorithm estimating $\ell_0(x)$ within a $(1 \pm \varepsilon)$ factor using $\mathcal{O}(\frac{1}{\varepsilon^2} \log^2 N)$ space.*

2.6. Communication Complexity

In 1979, communication complexity was introduced by Yao [132] as a measure of the amount of communication that is needed to evaluate a function whose input is distributed among several parties. Communication complexity is a powerful tool to show space lower bounds for algorithms, e. g., in data streams as we see next, or to prove lower bounds on the size of data structures, e. g., OBDDs [128]. The book on communication complexity by Kushilevitz and Nisan [83] gives a good introduction into the topic and presents many more applications. Here, we focus on the relationship between communication complexity and space in a streaming algorithm. For this, assume that we have two players, usually called Alice and Bob, who want to evaluate a function $f : X \times Y \to \{0, 1\}$ where Alice gets the input $x \in X$, Bob the input $y \in Y$. Alice is allowed to send a single message to Bob who has to output the value

of $f(x, y)$. How many communication bits are necessary to achieve that Bob can output the correct answer? Of course, the answer depends on the function f but it also depends on the actual model of communication which we are using. Here, we are only interested in the randomized communication complexity which we now define formally using the notation of the book by Kushilevitz and Nisan [83]. We also refer to the book for an overview of the different kinds of communication models and for lower and upper bounds on the complexity of some functions for these models.

Definition 2.6.1 (Randomized One-Way Communication Complexity)**.** Let $f : X \times Y \to \{0, 1\}$ be a function. In a *randomized communication protocol* P Alice and Bob have access to the inputs $x \in X$ and $y \in Y$ and random bit strings r_A and r_B, respectively, where the random bit strings are chosen independently from some probability distributions. We say that P *computes the function* f *with error* δ if the output of the protocol $P(x, y)$ is not equal to $f(x, y)$ with probability at most $\delta < 1/2$ for every pair $(x, y) \in X \times Y$. The *communication cost of* P is the maximum number of bits communicated over every choice of x, y, r_A, and r_B. We denote the *randomized one-way communication complexity* of f by $R_\delta(f)$ which is defined as the minimum cost of a randomized communication protocol computing f with error δ.

As an easy example of space lower bounds for streaming algorithms, we show an $\Omega(N^2)$ lower bound for deciding whether a graph contains a perfect matching. This proof is similar to a proof by Feigenbaum et al. [45]. We need a communication problem which is used quite often for lower bounds: The index function $IND : \{0, 1\}^n \times [n] \to \{0, 1\}$ is defined by $IND(x, i) = x_i$, i. e., Alice gets a vector $x \in \{0, 1\}^n$ and Bob gets an index i and has to output the value of x_i. It is well known that $R_\delta(IND) = \Omega(n)$ for any constant δ [82]. Now, we want to reduce the index problem to the perfect matching problem. Let $x \in \{0, 1\}^{N^2}$ and $i \in [N^2]$ be an input for the index problem. The vector x can be interpreted as an adjacency matrix of a bipartite graph $G = (A, B, E)$ with $|A| = |B| = N$. Bob's index i describes exactly one pair $(a, b) \in A \times B$. Let $\mathcal{A}$ be a randomized streaming algorithm that decides whether there is a perfect matching in a graph with constant probability. Now, in the reduction Alice inserts the edges represented by the input x into the stream and runs $\mathcal{A}$ on this part of the stream. Then she sends the state of the algorithm (described by the memory content of $\mathcal{A}$) to Bob. For every $a' \in A \setminus \{a\}$ Bob adds an edge $\{a', v_{a'}\}$ and an edge $\{b', u_{b'}\}$ for every $b' \in B \setminus \{b\}$ (see Figure 2.4). Now, it is easy to see that there is a perfect matching in the graph if and only if the graph contains the edge $\{a, b\}$ which means that $x_i = 1$. Therefore, the message size and with it the space usage of $\mathcal{A}$ has to be at least $\Omega(N^2)$.

Recently, another well studied communication problem [11, 126, 49], the Boolean Hidden Matching Problem (BHM), was used in streaming lower bounds for algorithms approximating the size of the maximum matching and the max-cut, i. e., a cut of maximal size [44, 75]. BHM is defined as follows: Let $n \in \mathbb{N}$ be even. Alice gets a vector $x \in \{0, 1\}^n$, Bob a perfect matching M on the coordinates $[n]$ of x and a bit string $w \in \{0, 1\}^{n/2}$. This means

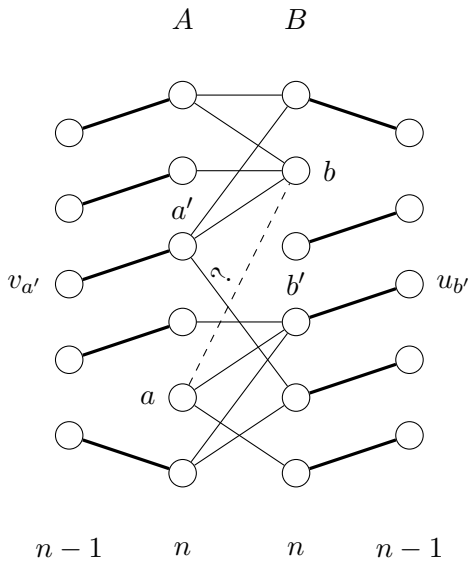


Figure 2.4.: Reduction from the index problem to perfect matching. Alice' input vector corresponds to the edges between A and B and Bob's index defines the pair of nodes (a, b). The only possible perfect matching can consist of the thick edges and the possible edge between a and b.

$M = \{(i_1, j_1), \ldots, (i_{n/2}, j_{n/2})\}$ with $\bigcup_{l=1}^{n/2}\{i_l, j_l\} = [n]$. The matching induces a bit string of length $n/2$ defined by $z = (x_{i_1} \oplus x_{j_1}), \ldots, (x_{i_{n/2}} \oplus x_{j_{n/2}})$. Alice and Bob get the promise that either $z = w$ or $\overline{z} = w$ where $\overline{z}$ is the coordinate-wise negation of the bits of z. Alice can send one message and Bob has to decide whether $z = w$ or $\overline{z} = w$. Intuitively, BHM can be solved by the following randomized protocol: Alice sends $\Theta(\sqrt{n})$ randomly chosen bits of x to Bob. Using the birthday paradox, it is possible to show that with constant probability Alice sends both bits of at least one pair, say $(x_{i_1} \oplus x_{j_1})$. Due to the promise, Bob knows that either $w_1 = x_{i_1} \oplus x_{j_1}$ or $\overline{w_1} = (x_{i_1} \oplus x_{j_1})$ and can correctly output the answer.

Here, we will use the more general Boolean Hidden Hypermatching Problem investigated by Verbin and Yu [126].

Definition 2.6.2 (Boolean Hidden Hypermatching Problem [126])**.** In the *Boolean Hidden Hypermatching* Problem $BHH_{t,n}$, Alice gets a boolean vector $x \in \{0,1\}^n$ with $n = 2kt$ for some $k \in \mathbb{N}$ and Bob gets a perfect t-hypermatching M on the n coordinates of x, i. e., each edge has exactly t coordinates, and a string $w \in \{0,1\}^{n/t}$. We denote the Boolean vector of length n/t given by $(\bigoplus_{1\leq i\leq t} x_{M_{1,i}}, \ldots, \bigoplus_{1\leq i\leq t} x_{M_{n/t,i}})$ by Mx where $(M_{1,1}, \ldots, M_{1,t}), \ldots, (M_{n/t,1}, \ldots, M_{n/t,t})$ are the edges of M. It is promised that either $Mx \oplus w = 1^{n/t}$ or $Mx \oplus w = 0^{n/t}$. The problem is to return 1 in the first case and 0 otherwise.

The idea of the upper bound also works for $BHH_{t,n}$ which gives a randomized protocol with $\mathcal{O}(n^{1-1/t})$ bits of communication. Verbin and Yu [126] showed that this protocol is

asymptotically optimal by presenting a lower bound of $\Omega(n^{1-1/t})$ for the randomized one-way communication complexity for $BHH_{t,n}$. Note that for $t = 2$ the problem $BHH_{t,n}$ is the Boolean Hidden Matching Problem.

Theorem 2.6.3 ([126])**.** *The randomized one-way communication complexity $R_{1/3}(BHH_{t,n})$ of $BHH_{t,n}$ when $n = 2kt$ for some integer $k \geq 1$ is $\Omega(n^{1-1/t})$.*

3. OBDD Sizes of Graph Classes and Applications

In this chapter, we investigate the OBDD size of graphs coming from one of the following graph classes: (unit) interval graphs, trees, bipartite permutation graphs, (bi)convex graphs, chain graphs, and threshold graphs. In Section 3.1, we give an overview of the related work regarding graph representations by OBDDs and OBDD-based algorithms. Section 3.2 starts with an overview of the results on the OBDD size of graphs presented in this section. In the first subsection 3.2.1 we present a new technique which enables us to show upper and lower bounds of the OBDD size for graphs with a fixed variable order. In subsection 3.2.2, 3.2.3, 3.2.5, and 3.2.6, we apply this technique to the aforementioned graph classes where we can show asymptotically optimal bounds or bounds that are optimal up to a $\log N$ factor. In the last Section 3.3, we design an OBDD-based algorithm for computing a maximum matching algorithm on unit interval graphs and a graph coloring algorithm for interval graphs.

3.1. Related Work and Contribution

The first question one should consider when investigating the OBDD size of graphs is the number of bits used for labeling the nodes of the graph. Obviously, $\lceil \log N \rceil$ bits are necessary to label N distinct nodes. However, increasing the number of bits can lead to a smaller representation size by OBDDs. Nunkesser and Woelfel [107] used $\lceil \log N \rceil$ bits for encoding the nodes of cographs and showed that the OBDD size is bounded by $\mathcal{O}(N \log N)$ whereas Meer and Rautenbach [96] use $c \cdot \log N$ bits for some constant $c > 1$ to improve the representation size to $\mathcal{O}(N)$. More precisely, Meer and Rautenbach [96] investigated graphs with bounded clique-width or tree-width. They showed that the OBDD size is $\mathcal{O}(\log N)$ for graphs with bounded tree-width and $\mathcal{O}(N)$ for cographs where each node is labeled by $c \cdot \log N$ bits for a constant $c > 1$. In addition, they gave an upper bound of $\mathcal{O}(N)$ for graphs with bounded clique-width where the nodes have encoding length of $\mathcal{O}(N)$.

Here, we use the minimal amount of bits to label the nodes of a graph because we are not only interested in a small representation size but also in algorithms working on OBDDs representing the graphs. As we know from the previous chapter, the worst-case OBDD size is exponentially large in the number of variables which is why we want to keep this number as small as possible. Another problem with a larger domain for the node labels is that it also

possibly increases the size of the data structure storing the valid labels which is often needed in implicit algorithms. This means that for the node set V of a graph $G = (V, E)$ we have to have a function $\chi_V : \{0,1\}^l \to \{0,1\}$ which maps a valid encoding $x \in \{0,1\}^l$ to 1 and an invalid encoding to 0 where l is the length of the encodings of the nodes. If the encodings get more complicated, the space needed for χ_V increases. Here, we use the simple function $\chi_V(x) = 1$ if and only if $|x|_2 < N$ which has a small OBDD size of $\mathcal{O}(n)$. Once we have fixed the number of bits for the encodings, we have to compute a good labeling for the nodes that minimizes the OBDD size. As discussed in the previous chapter, we use the interleaved variable order with decreasing significance in order to have small OBDDs for our auxiliary functions. But even for a fixed variable order, the computational complexity of computing the best node labeling is unknown. For a restricted case where it is only allowed to permute the order of the bits of a given encoding, Bollig [16] showed that it is NP-hard to compute the best labeling.

Beside the OBDD size of cographs, Nunkesser and Woelfel [107] also showed that interval graphs can be represented by OBDDs of size $\mathcal{O}(N^{3/2}/\log^{3/4} N)$ and unit interval graphs with OBDD size $\mathcal{O}(N/\sqrt{\log N})$. We know that the size of an OBDD representing an arbitrary graph is $\mathcal{O}(N^2/\log N)$. For bipartite graphs they were able to show a lower bound of $\Omega(N^2/\log N)$. In addition, they proved a lower bound of $\Omega(N)$ for general interval graphs, $\Omega(N/\log N)$ for unit interval graphs, and $\Omega(N/\log N)$ for cographs. We introduce a new technique to show upper and lower bounds of the OBDD size for a fixed variable order which enables us to improve the bounds of Nunkesser and Woelfel for interval graphs to $\mathcal{O}(N \log N)$ and for unit interval graphs to $\mathcal{O}(N/\log N)$. The key idea for this technique is to identify subfunctions of the characteristic function χ_E by submatrices of a special adjacency matrix where the rows and columns are sorted with respect to the variable order. The advantage of this perspective is that we can count the number of different submatrices instead of dealing with subfunctions. This is often much easier because we can use the structure of the adjacency matrix, e. g., for (unit) interval graphs [98], and the neighborhood of the nodes to recognize patterns in these submatrices. Applying this technique to trees, bipartite permutation graphs, biconvex graphs, chain graphs, and threshold graphs give us an asymptotically optimal upper bound on the OBDD size of $\mathcal{O}(N/\log N)$. For convex graphs we get an upper bound of $\mathcal{O}(N \log N)$. The lower bounds of $\Omega(N)$ for convex graphs and $\Omega(N/\log N)$ for the other graphs can be obtained by using the counting results from Section 2.2. Independently, Takaoka, Tayu, and Ueno [123] also investigated the size of bipartite permutation graphs and biconvex graphs. They also used our technique to achieve an upper bound of $\mathcal{O}(N/\log N)$ with a matching lower bound of $\Omega(N/\log N)$. Similarly to the proof of the upper bound for interval graphs by Nunkesser and Woelfel, they were able to show an $\mathcal{O}(N^{3/2}/\log^{3/4} N)$ bound on the OBDD size for permutation graphs and convex graphs. In addition to the small OBDD size, our labeling of the nodes is easy to compute and often coincides in some way with the orders known from

the graph classes in Section 2.2. On the negative side, we show that there is a distribution on interval graphs such that with constant probability the OBDD size is $\Omega(N \log N)$ if we use our labeling and variable order. This means that any further improvement towards the $\Omega(N)$ lower bounds needs either a different labeling or another variable order.

In terms of implicit OBDD-based graph algorithms, Hachtel and Somenzi [57] were one of the first using OBDDs to solve an optimization problem on graphs, namely the maximum flow problem in 0-1-networks. They were able to solve instances up to 10^{36} edges and 10^{27} nodes in reasonable time. Sawitzki [115] described another implicit algorithm for the same problem, which uses $\mathcal{O}(N \log^2 N)$ functional operations. The algorithm by Hachtel and Somenzi was generalized to maximum weighted matchings in bipartite graphs [55] and to a relaxation of matchings, called semi-matching, in bipartite graphs [56]. Since implicit algorithms approach a problem in a different way, designing implicit algorithms for optimization problems can give new insights into the problems. For instance, Gentilini, Piazza, and Policriti [51] developed the notion of spine-sets in the context of implicit algorithms for connectivity related problems. Aiming for a small number of functional operations, implicit algorithms using a polylogarithmic number of operations were designed for instance for topological sorting [131], maximal matching [20] and minimum spanning tree [15]. We present two implicit algorithms for (unit) interval graphs: A maximum matching algorithm for unit interval graphs using only $\mathcal{O}(\log N)$ functional operations and a coloring algorithm for interval graphs using $\mathcal{O}(\log^2 N)$ functional operations where we color the nodes such that all adjacent nodes have different colors and the number of used colors is minimal. The matching algorithm takes advantage of the information given by the labels of the nodes. Furthermore, we were able to compute the transitive closure of a unit interval graph using only $\mathcal{O}(\log N)$ operations instead of $\mathcal{O}(\log^2 N)$ operations, which are needed in general. In order to implement this algorithm efficiently, we have to extend a known result due to Woelfel [131] to the interleaved variable order with decreasing significance for constructing OBDDs representing multivariate threshold functions. These are functions of the form $\sum_{j=1}^{k} w_j \cdot |x^{(j)}|_2 \geq T$ for Boolean vectors $x^{(j)}$, weights w_j, and threshold T. For the coloring algorithm we show how to get a total order on the right endpoints (given that the labels of the nodes respect the order of the left endpoints) and how to compute a minimal coloring of the nodes by using these orders based on a optimal greedy algorithm [108]. To the best of the author's knowledge, this is the first time that the labeling of nodes is used to speed up an implicit algorithm for a large graph class and to improve the number of functional operations.

3.2. OBDD Size of Graphs

We start with a summary visualized in Fig. 3.1 where we reuse the inclusion map of the graph classes in Fig.2.1 and illustrate our results regarding the OBDD sizes of the graph classes.

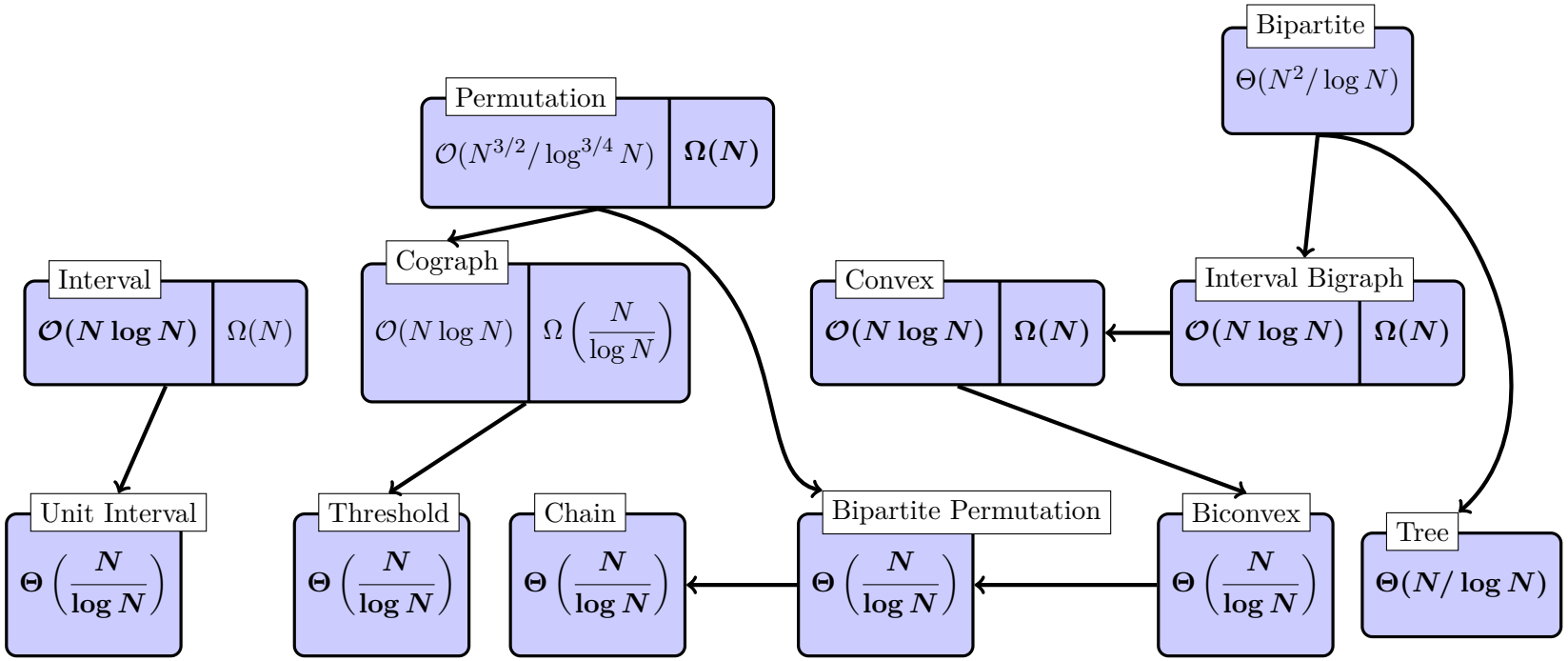


Figure 3.1.: Overview of the OBDD size of several graph classes where the bold values are shown in this thesis. The diagram is sorted in descending order by the upper bounds of the graph classes. Classes with the same upper bound appear on the same level, e. g., interval graphs, convex graphs, cographs, and interval bigraphs.

3.2.1. The π-ordered Adjacency Matrix

In order to bound the size of a Boolean function f by using Theorem 2.3.4, we have to count different subfunctions of f. We present a way to count the subfunctions of the characteristic function χ_E of the edge set of a graph using the adjacency matrix of the graph. This can give us a better understanding what a subfunction looks like in the graph scenario and get a more graph theoretic approach to subfunctions. The adjacency matrix of graphs from special graph classes (e. g., for interval graphs) yields structural properties which we use to bound the size of the π-OBDD. So if we use the knowledge about the structure of the adjacency matrix for a fixed labeling to bound the number of different subfunctions for a variable order π, then we can show an upper and/or lower bound on the π-OBDD size.

The rows (columns) of an adjacency matrix correspond to the x-variables (y-variables) of $\chi_E(x, y)$. We can sort the rows of the adjacency matrix according to a variable order π by connecting the i-th row to the input x with $\sum_{l=0}^{n-1} x_{\pi(l)} \cdot 2^{n-1-l} = i$, i. e., we let the l-th x-variable in π have significance 2^{n-1-l} to sort the rows. This can be done analogously to sort the columns. Thus, the variable order π defines a permutation of the rows and columns of the adjacency matrix resulting in a new matrix which we call π-ordered adjacency matrix.

Definition 3.2.1 (π-Ordered Adjacency Matrix)**.** Let $G = (V, E)$ be a graph and $\pi_{2,n}$ be a 2-interleaved variable order for the characteristic function χ_E. The *π-ordered* adjacency matrix A_π of G is defined as follows: $a_{ij} = 1$ iff $\chi_E(x, y) = 1$ with $\sum_{l=0}^{n-1} x_{\pi(l)} \cdot 2^{n-l-1} = i$ and $\sum_{l=0}^{n-1} y_{\pi(l)} \cdot 2^{n-l-1} = j$.

Notice that the π-ordered adjacency matrix is equal to the "normal" adjacency matrix where the rows and columns are sorted by the node labels iff the variables in π are sorted with decreasing significance. The π-ordered adjacency matrix gives us a visualization of the subfunctions in terms of *blocks* of the matrix. We focus here on interleaved variable orders which means that in every prefix of the order the number of tested x-variables and y-variables differs at most by one. As a consequence, the blocks from the next definition are almost quadratic but the definitions can be easily generalized to arbitrary variable orders.

Definition 3.2.2 (Blocks)**.** Let $n \in \mathbb{N}$ and A be a $2^n \times 2^n$ matrix. For $0 \leq k_1, k_2 \leq n$ with $k_2 \in \{k_1 - 1, k_1\}$, $0 \leq i \leq 2^{k_1} - 1$, and $0 \leq j \leq 2^{k_2} - 1$ the *block* $B_{i,j}^{k_1,k_2}$ of A is defined by the submatrix of size $2^{n-k_1} \times 2^{n-k_2}$ which is formed by the intersection of the rows $i \cdot 2^{n-k_1}, \ldots, (i+1) \cdot 2^{n-k_1} - 1$ and the columns $j \cdot 2^{n-k_2}, \ldots, (j+1) \cdot 2^{n-k_2} - 1$. If $k_1 = k_2$ we denote $B_{i,j}^{k_1,k_2}$ by $B_{i,j}^{k_1}$.

Recall Theorem 2.3.4 that we want to count the number of different subfunctions which result from replacing the first i variables according to the variable order by constants. We will see later that for an upper bound it is enough to consider only the case when i is even, i. e., the number of replaced x- and y-variables is exactly $i/2$. Now, we can observe that the block

$B^{i/2}_{|\alpha|_2,|\beta|_2}$ represents the function table of the subfunction which results from replacing the x-variables by $\alpha \in \{0,1\}^{i/2}$ and the y-variables by $\beta \in \{0,1\}^{i/2}$. Therefore, counting the number of different blocks $B^{i/2}_{|\alpha|_2,|\beta|_2}$ is equivalent to counting the number of different subfunctions. For an upper bound it is sufficient to bound the number of different subfunctions from above but for a lower bound we also have to ensure that the subfunctions are essentially dependent on the next variable. Since replacing the next variable by a constant is equivalent to halve the corresponding block, we can relate the dependency on a variable to a notion of a symmetric block which we define next.

Definition 3.2.3 (Symmetric Block)**.** Let $0 < k_1 \leq n$, $0 \leq k_2 < n$ with $k_2 \in \{k_1 - 1, k_1\}$ and $0 \leq i \leq 2^{k_1} - 1$, $0 \leq j \leq 2^{k_2} - 1$. A block $B^{k_1,k_2}_{i,j}$ of a π-ordered adjacency matrix is called *symmetric* if

- $B^{k_1+1,k_2}_{2i,j} = B^{k_1+1,k_2}_{2i+1,j}$ if $k_1 = k_2$ or
- $B^{k_1,k_2+1}_{i,2j} = B^{k_1,k_2+1}_{i,2j+1}$ if $k_2 = k_1 - 1$

and *asymmetric* otherwise.

Now, we can formally prove the relationship between subfunctions and blocks.

Lemma 3.2.4. *Let $G = (V, E)$ be a graph and $\pi_{2,n}$ be a 2-interleaved variable order for the characteristic function χ_E. Let s_{k_1,k_2} for $0 < k_1 \leq n$, $0 \leq k_2 < n$, and $k_2 \in \{k_1 - 1, k_1\}$ be the number of different subfunctions of χ_E which results from replacing all variables $x_{\pi(k)}$ with $0 \leq k \leq k_1$ and all variables $y_{\pi(k)}$ with $0 \leq k \leq k_2$ by constants and which essentially depend on $x_{\pi(k_1+1)}$ if $k_1 = k_2$ or $y_{\pi(k_1)}$ if $k_2 = k_1 - 1$. Then s_{k_1,k_2} is equal to the number of different and asymmetric blocks $B^{k_1,k_2}_{i,j}$.*

Proof. Let k_1 and k_2 be arbitrary but fixed with $0 < k_1 \leq n$, $0 \leq k_2 < n$, and $k_2 \in \{k_1 - 1, k_1\}$. By the definition of the π-ordered adjacency matrix A_π, we know that $\chi_E(x, y) = 1$ if and only if $a_{ij} = 1$ with $i = \sum_{l=0}^{n-1} x_{\pi(l)} \cdot 2^{n-l-1}$ and $j = \sum_{l=0}^{n-1} y_{\pi(l)} \cdot 2^{n-l-1}$. Now, for $\alpha \in \{0,1\}^{k_1}, \beta \in \{0,1\}^{k_2}$ let $f_{|\alpha,\beta} : \{0,1\}^{n-k_1} \times \{0,1\}^{n-k_2} \to \{0,1\}$ be the subfunction of $f := \chi_E$ where $x_{\pi(0)}, \ldots, x_{\pi(k_1-1)}$ are replaced by $\alpha_{k_1-1}, \ldots, \alpha_0$ and $y_{\pi(0)}, \ldots, y_{\pi(k_2-1)}$ are replaced by $\beta_{k_2-1}, \ldots, \beta_0$. Then we have $f_{|\alpha,\beta}(x, y) = 1$ if and only if $a_{ij} = 1$ with

$$\begin{aligned} i &= \sum_{l=0}^{k_1-1} \alpha_l 2^{n-k_1+l} + \sum_{l=k_1}^{n-1} x_{\pi(l)} 2^{n-l-1} = |\alpha|_2 \cdot 2^{n-k_1} + \sum_{l=k_1}^{n-1} x_{\pi(l)} 2^{n-l-1} \text{ and} \\ j &= \sum_{l=0}^{k_2-1} \beta_l 2^{n-k_2+l} + \sum_{l=k_2}^{n-1} y_{\pi(l)} 2^{n-l-1} = |\beta|_2 \cdot 2^{n-k_2} + \sum_{l=k_2}^{n-1} y_{\pi(l)} 2^{n-l-1}. \end{aligned}$$

This means submatrix formed by the intersection of the rows with number $|\alpha|_2 \cdot 2^{n-k_1}, \ldots, |\alpha|_2 \cdot 2^{n-k_1} + 2^{n-k_1} - 1 = (|\alpha|_2 + 1) \cdot 2^{n-k_1} - 1$ and the columns with number $|\beta|_2 \cdot 2^{n-k_2}, \ldots, (|\beta|_2 + 1) \cdot 2^{n-k_2} - 1$ is the function table of $f_{|\alpha,\beta}$. By definition, this submatrix is equal to the

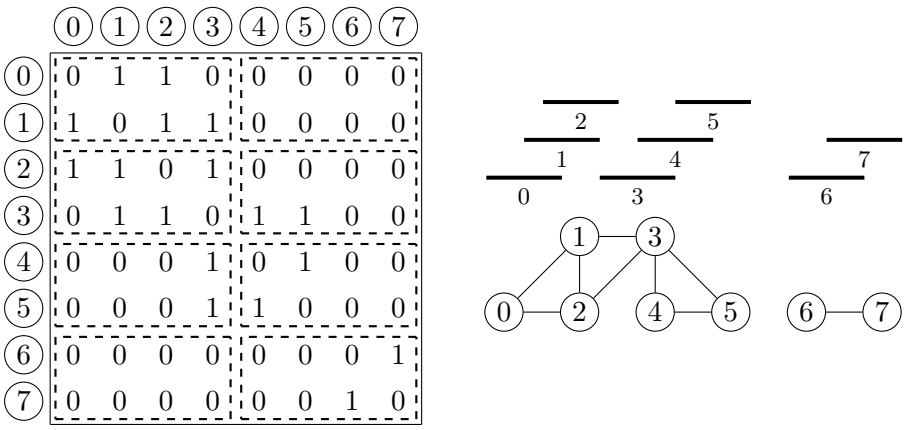


Figure 3.2.: The π-ordered adjacency matrix of a unit interval graph with $\pi = (2, 1, 0)$ and the framed blocks $B^{2,1}_{0,0}, B^{2,1}_{1,0}, B^{2,1}_{2,0}, B^{2,1}_{3,0}, B^{2,1}_{0,1}, B^{2,1}_{1,1}, B^{2,1}_{2,1}$ and $B^{2,1}_{3,1}$ which correspond to the subfunctions where x_2, y_2, and x_1 are replaced by constants.

block $B^{k_1,k_2}_{|\alpha|_2,|\beta|_2}$. A subfunction $f_{|\alpha,\beta}$ is essentially dependent on the next variable, if the two subfunctions where this variable is replaced by either 0 or 1 are different. This next variable is $x_{\pi(k_1+1)}$ if $k_1 = k_2$ and $y_{\pi(k_1)}$ if $k_2 = k_1 - 1$. In the first case the subfunctions are $f_{|[\alpha,0],\beta}$ and $f_{|[\alpha,1],\beta}$ whereas in the second case they are $f_{|\alpha,[\beta,0]}$ and $f_{|\alpha,[\beta,1]}$. In terms of blocks, this means that either the blocks $B^{k_1+1,k_2}_{i_1,j}, B^{k_1+1,k_2}_{i_2,j}$ with $i_1 = |[\alpha, 0]|_2 = 2i$ and $i_2 = |[\alpha, 1]|_2 = i_1 + 1$ or $B^{k_1,k_2+1}_{i,j_1}, B^{k_1,k_2+1}_{i,j_2}$ with $j_1 = |[\beta, 0]|_2 = 2j$ and $j_2 = |[\beta, 1]|_2 = j_1 + 1$ corresponding to these subfunctions have to be different which is exactly the definition of an asymmetric block. □

For instance, say that the variables are tested with decreasing significance. Then $a_{ij} = 1$ iff $\chi_E(x, y) = 1$ with $\sum_{l=0}^{n-1} x_l \cdot 2^l = |x|_2 = i$ and $\sum_{l=0}^{n-1} y_l \cdot 2^l = |y|_2 = j$, i. e., the π-ordered adjacency matrix A_π of G is the standard adjacency matrix where the labeling of the columns and rows is ordered by the node labels. Fig. 3.2 illustrate that for every k each subfunction of f where the first k bits (according to $\pi_{2,n}$) are replaced by constants corresponds to a block of this adjacency matrix.

Bollig and Wegener [22] use a similar approach to visualize subfunctions of a storage access function by building a matrix whose columns and rows are sorted according to the variable order and correspond to variables (not assignments as in our π-ordered matrix). Notice that A_π is not the communication matrix which is often used to show lower bounds of the OBDD size. For the communication matrix the variable order is split in two parts, the first k variables and the last $2n - k$ variables, and the rows and columns correspond to assignments to the first k variables and last $2n - k$ variables, respectively.

3.2.2. OBDD Size of (Unit) Interval Graphs

Now, we use the π-ordered adjacency matrix and count the number of different blocks to improve the bounds of the OBDD size of interval graphs from $\mathcal{O}(N^{3/2}\log^{3/4} N)$ [107] to $\mathcal{O}(N \log N)$. We also use a simpler labeling of the nodes. A characterization of the adjacency matrix of a (unit) interval graph has been shown by Mertzios [98] who introduced a normal interval representation. Here, it is sufficient to get the same structure by sorting the intervals according to their left endpoint.

Theorem 3.2.5. *Let $\pi_{2,n}$ be the interleaved variable order with decreasing significance and $G = (V, E)$ be an interval graph with $N := |V|$ nodes. The $\pi_{2,n}$-OBDD size of χ_E can be bounded above by $\mathcal{O}(N \log N)$.*

Proof. Let $f := \chi_E$, $1 \leq k \leq n$ and s_k be the number of different subfunctions $f_{|\alpha,\beta}$ of f where $\alpha \in \{0,1\}^k$ is an assignment to the variables $x_{n-1}, \ldots, x_{n-k}$ and $\beta \in \{0,1\}^k$ is an assignment to the variables $y_{n-1}, \ldots, y_{n-k}$, respectively. The number of different subfunctions where the variables $x_{n-1}, \ldots, x_{n-k}$ and $y_{n-1}, \ldots, y_{n-k-1}$ are replaced by constants can be bounded by $2 \cdot s_k$ because one additional variable can at most double the number of subfunctions.

We label the nodes according to their position in the sorted sequence of interval left endpoints (as for example in Fig. 3.2). Recall that the interleaved variable order with decreasing significance means that $a_{i,j}$ is one if and only if interval i intersects interval j. Now, notice that if $a_{i,j}$ is zero for $j > i$, i. e., interval j has a larger left endpoint than interval i and does not cut interval i, then no interval $j' > j$ with a larger left endpoint can cut interval i. Thus, for every column $i \in \{0, \ldots, N-1\}$, the sequence $(a_{i+1,i}, \ldots, a_{N-1,i})$ is zero or starts with a continuous sequence of ones followed by only zeros, i. e., there exists a j such that $a_{k,i} = 1$ for $i < k \leq j$ and $a_{k,i} = 0$ for $k > j$.

From Lemma 3.2.4 we know that every subfunction $f_{|\alpha,\beta}$ corresponds to a block of A_π. Let $\beta = 0^k$ and $|\alpha|_2 \geq 1$, i. e., we consider the blocks $B^k_{|\alpha|_2,0}$ of size $2^{n-k} \times 2^{n-k}$ (see Fig. 3.3). As we observed, every column of A_π has at most one possible *changing* position c such that $a_{c,i} = 1$ and $a_{c+1,i} = 0$ (below the diagonal). We say that a changing position c is inside a block if $a_{c,i}$ is in the block. Looking at the sequence $(B^k_{1,0}, \ldots, B^k_{2^k-1,0})$ of blocks, this fact implies that a block $B^k_{i,0}$ can only form a new block, i. e., all previous blocks in the sequence are different to this block, if there is a changing position in one column inside of $B^k_{i,0}$ or inside the block $B^k_{i-1,0}$. Of course, this holds for any assignment $\beta \in \{0,1\}^k$. Therefore, every changing position can induce at most two different blocks. Thus, we can bound the number of different blocks by two times the number of possible changing positions which is at most the number of columns of a block, i. e., $2 \cdot 2^{n-k}$. Since the graph is symmetric and the blocks containing the diagonal can only add 2^k additional distinct blocks, we can bound the overall number of different blocks by $\mathcal{O}(2^{n-k} \cdot 2^k + 2^k) = O(2^n)$ and thus $s_k = O(2^n)$. Summing this up over all possible values of k we get $\mathcal{O}(2^n \cdot n) = O(N \log N)$ as an upper bound on the size of the π-OBDD. □

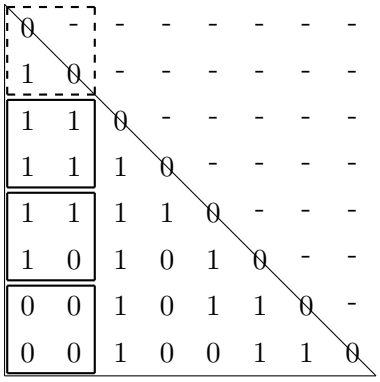


Figure 3.3.: Possible adjacency matrix with 8 nodes and framed subfunctions $f_{|\alpha,\beta}$ with $\beta = 0^k$, $|\alpha|_2 \geq 1$, and $k = 2$.

In the case of unit interval graphs we close the gap between $\Omega(N/\log N)$ and $\mathcal{O}(N/\sqrt{\log N})$ by using the π-ordered adjacency matrix to get a better upper bound of $\mathcal{O}(N/\log N)$. Nunkesser and Woelfel [107] showed that s_k can be bounded from above by $\min\{\mathcal{O}(2^k), 2^{2^{n-k}}\}$ which leads to an OBDD size of $\mathcal{O}(N/\sqrt{\log N})$. We improve the bound for large values of k but for the sake of completeness we also show the $\mathcal{O}(2^k)$ bound by using the π-ordered adjacency matrix. However, the ideas for the proof of the $\mathcal{O}(2^k)$ bound are similar to [107].

Theorem 3.2.6. *Let $\pi_{2,n}$ be the interleaved variable order with decreasing significance and $G = (V, E)$ be a unit interval graph with $N := |V|$ nodes. The $\pi_{2,n}$-OBDD size of χ_E can be bounded above by $\mathcal{O}(N/\log N)$.*

Proof. Again, let $f := \chi_E$ and s_k be the number of different subfunctions $f_{|\alpha,\beta}$ of f where α is an assignment to the variables $x_{n-1}, \ldots, x_{n-k}$ and β is an assignment to the variables $y_{n-1}, \ldots, y_{n-k}$, respectively. As we have seen in the last proof, the number of different subfunctions where the first k x-variables and $k+1$ y-variables are replaced by constants can be bounded by $2 \cdot s_k$.

We label the nodes according to their interval left endpoints. Let $|\alpha|_2 > |\beta|_2$. We know that $f_{|\alpha,\beta}$ corresponds to the block $B^k_{|\alpha|_2,|\beta|_2}$ of the π-ordered adjacency matrix of G. Let $c_j > j$ be the changing position (as in the proof of Theorem 3.2.6) of column j, i. e., the position (row) below the diagonal such that $a_{c_j,j} = 1$ and $a_{c_j+1,j} = 0$. If the entries of the column below the diagonal are all 0 or all 1 then we set $c_j = j$ and $c_j = N - 1$, respectively. Recall that the intervals are labeled according to their left endpoint. Since G is a unit interval graph, this is equivalent to labeling them according to their right endpoints, i. e., if $j > i$, then interval j starts *and* ends after interval i. This implies that the sequence $c_0, \ldots, c_{N-1}$ is monotonically increasing, i. e., $c_0 \leq c_1 \leq \ldots \leq c_{N-1}$. Now, for a fixed β, let $B^k_{|\alpha_1|_2,|\beta|_2}, \ldots, B^k_{|\alpha_l|_2,|\beta|_2}$ with $|\alpha_1|_2 < \ldots < |\alpha_l|_2$ be the distinct blocks among all blocks of the form $B^k_{\bullet,|\beta|_2}$. Since the sequence of c_i is monotonically increasing, for every β' with $|\beta'|_2 > |\beta|_2$ all non-constant

blocks $B^k_{|\alpha'|_2,|\beta'|_2}$ have to satisfy $|\alpha'|_2 \geq |\alpha_l|_2$ which means there cannot be more than $2 \cdot 2^k$ different blocks $B^k_{|\alpha|_2,|\beta|_2}$ with $|\alpha|_2 > |\beta|_2$. Since the matrix is symmetric and there are 2^k blocks on the diagonal, we can bound the overall number of different blocks by $5 \cdot 2^k$.

For bounding the number of blocks for large k, we observe that every column of a block $B^k_{|\alpha|_2,|\beta|_2}$ with $|\alpha|_2 > |\beta|_2$ consists of a beginning sequence of ones of length $l \geq 0$ and an ending sequence of zeros of length $C - l$, where $C = 2^{n-k}$ is the number of rows and columns of the block. Let $l_1, \ldots, l_C$ be the lengths of the beginning sequence consisting of ones of every column in the block $B^k_{|\alpha|_2,|\beta|_2}$. We know that the sequence $l_1, \ldots, l_C$ is monotonically increasing, i. e., $l_1 \leq l_2 \leq \ldots \leq l_C$. How many different blocks of this form can be constructed? We can construct such a block by drawing C numbers between 0 and C and sorting them, i. e., it is equivalent to drawing C numbers out of $\{0, \ldots, C\}$ with repetition, where order does not matter. The number of C-combinations with repetition is equal to $\binom{(C+1)+C-1}{C} = \binom{2C}{C}$ and this can be bounded above by 2^{2C}. Since G is symmetric, this is also a bound on the number of different blocks above the diagonal. The omitted blocks on the diagonal can be constructed in a similar way: At first, the diagonal of these blocks is zero and the blocks are symmetric. Below the diagonal the blocks also consist of a sequence of ones probably followed by a sequence of zeros. So the number of different blocks is bounded above by the number of different blocks, which are not on the diagonal, i. e., by 2^{2C}. Hence, for $C = 2^{n-k}$ we can bound s_k above by $3 \cdot 2^{2^{n-k+1}}$. Therefore, the OBDD size is at most

$$\begin{aligned}
\sum_{k=0}^{n-1} \min\{5 \cdot 2^k, 3 \cdot 2^{2^{n-k+1}}\} &\leq 5 \cdot \sum_{k=0}^{n-\log n+1} 2^k + 3 \cdot \sum_{k=n-\log n+2}^{n-1} 2^{2^{n-k+1}} \\
&\leq 5 \cdot 2^{n-\log n+2} + 3 \cdot 2^{2^{\log n-1}} \cdot (\log n - 2) \\
&= O(N/\log N) + O(\sqrt{N} \cdot \log\log N) \\
&= O(N/\log N).
\end{aligned}$$

□

The difference between unit and general interval graphs is that in general interval graphs there is no dependence between the columns of the π-ordered adjacency matrix, which will be important for our lower bound, while in unit interval graphs, the row number of the last 1-entry in a column is increasing from left to right. The proofs of the upper bounds suggest that the number of blocks $B^k_{i,j}$ with a changing position roughly determines the number of OBDD nodes labeled by x_{n-k-1}. We know that every layer of the OBDD, i. e., every set of OBDD nodes labeled by the same variable, has size $\mathcal{O}(N)$ which means that there have to be $\Omega(n)$ layers of the OBDD of size $\Omega(N)$ to show a lower bound of $\Omega(N \log N)$. Explicitly constructing a worst-case interval graph with OBDD size of $\Omega(N \log N)$ is difficult because $\Omega(n)$ layers correspond to $\Omega(n)$ values of k and, since the block $B^k_{i,j}$ results from dividing a block $B^{k-1}_{i',j'}$, many dependencies have to be considered to ensure that $\Omega(N)$ blocks are different for all the possible values of k.

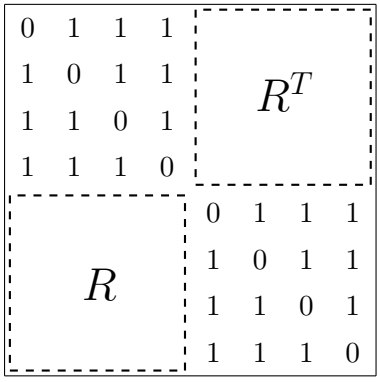


Figure 3.4.: Random interval graph where only R is generated randomly

In order to overcome these dependencies, we look at a random interval graph and compute the expected value of the number of different blocks for $\Omega(n)$ values of k. Intuitively, in the worst-case the lengths of the 1-sequences of the columns are uniformly distributed such that there are many blocks with a small number of changing positions inside which maximizes the possibility that there are many different blocks. Choosing an appropriate distribution on the set of interval graphs, we show that the expected number of different blocks with one changing position is $\Omega(N)$ for $\Omega(n)$ values of k. Due to the linearity of expectation, the expected value of the sum of the number of different blocks over all values of k is $\Omega(Nn) = \Omega(N \log N)$, i. e., there is an interval graph whose OBDD size is also $\Omega(N \log N)$. Since we also have an upper bound of $\mathcal{O}(N \log N)$, we can show even more: If we draw a random interval graph from our distribution, then the size of the OBDD is $\Omega(N \log N)$ with constant probability.

Theorem 3.2.7. *The worst-case $\pi_{2,n}$-OBDD size of an interval graph is $\Omega(N \log N)$ where the nodes are labeled according to the interval left endpoints and $\pi_{2,n}$ is an interleaved variable order with decreasing significance. Furthermore, there is a (nontrivial) distribution on the set of interval graphs such that the probability is $\Omega(1)$ that the OBDD size is at least $\Omega(N \log N)$.*

Proof. We describe a random process to generate an interval graph where the adjacency matrix is constant except the $N/2 \times N/2$ lower left submatrix which we denote by R (see Fig. 3.4). For this, we choose the length of the 1-sequence of column j for all $0 \leq j \leq N/2-1$ uniformly at random from $\{N/2-j, \ldots, N-1-j\}$ and for all $N/2 \leq j \leq N-1$ the length of column j is equal to $N-1-j$. As a result, the length of the 1-sequence of each column within R is uniformly distributed in $\{1, \ldots, N/2\}$.

Let $G = (V, E)$ be a random interval graph generated by the above process and $f := \chi_E$. Let $1 \leq k \leq n$ and s_k be the number of different subfunctions $f_{|\alpha,\beta}$ of f where $\alpha \in \{0,1\}^k$ is an assignment to the variables $x_{n-1}, \ldots, x_{n-k}$ and $\beta \in \{0,1\}^k$ is an assignment to the variables $y_{n-1}, \ldots, y_{n-k}$, respectively, and $f_{|a,b}$ is essentially dependent on x_{n-k-1}. We show

that the expected value of s_k with $n/2+1 \leq k \leq (3/4)n$ is $\Omega(2^n) = \Omega(N)$. Therefore, there has to be an interval graph with $\pi_{2,n}$-OBDD size $\Omega(N \log N)$.

We known that k induces a grid in R consisting of $2^{n-k} \times 2^{n-k}$ blocks. At first, we calculate the expected number of blocks with exactly one changing position. The probability that a fixed block of size $L \times L$ with $L \leq 2^{n/2-1}$ has exactly one changing position is

$$\begin{aligned}
\sum_{i=1}^{L} \frac{L-1}{2^{n-1}} \cdot \left(1 - \frac{L-1}{2^{n-1}}\right)^{L-1} &\geq \frac{L \cdot (L-1)}{2^{n-1}} \cdot \left(1 - \frac{L}{2^{n-1}}\right)^{L-1} \\
&\geq \frac{L \cdot (L-1)}{2^{n-1}} \cdot \left(1 - \frac{2^{n/2}}{2^{n-1}}\right)^{2^{n/2-1}-1} \\
&\geq \frac{L \cdot (L-1)}{2^{n-1}} \cdot e^{-1}.
\end{aligned}$$

Let $n/2+1 \leq k \leq (3/4)n$ be fixed. Since we have $2^{k-1} \cdot 2^{k-1}$ blocks of size $2^{n-k} \times 2^{n-k}$ in R, the expected value of the number of blocks with exactly one changing position is at least

$$\frac{1}{2e} \cdot 2^{k-1} \cdot 2^{k-1} \cdot \frac{2^{n-k} \cdot (2^{n-k}-1)}{2^n} = \frac{1}{8e} \cdot 2^k \cdot (2^{n-k}-1) = \Omega(2^n).$$

Now, we have to ensure that these blocks correspond to different subfunctions which are also essentially dependent on x_{n-k-1}, i. e., these blocks have to be asymmetric. Due to the one changing position in each block, this is always the case. Blocks $B^k_{i,j}$ and $B^k_{i',j}$ with exactly one changing position and $i \neq i'$ clearly correspond to different subfunctions because they are in the same block column. But blocks $B^k_{i,j}$ and $B^k_{i',j'}$ with $j \neq j'$, i. e., from different block columns, do not have to be different. By replacing some columns of the matrix by constants, we ensure that this also holds. Consider the case $k = (3/4)n$, i. e., the finest grid of R made by $2^{n-k} \times 2^{n-k}$ blocks with $n/2+1 \leq k \leq (3/4)n$. For every block column $0 \leq j \leq 2^k - 1$ we fix the first k columns of $B^k_{i,j}$ with $0 \leq i \leq 2^k - 1$ such that they represent the binary number $[j]_2$ of the column index. Thus, we have that blocks $B^k_{i,j}$ and $B^k_{i',j'}$ with $j \neq j'$ are always different. Since we looked at the finest grid, this also holds for smaller values of k because every larger block is equal to a union of small blocks. The probability that a block contains exactly one changing position is smaller than before, since we fix some columns. For $k = (3/4)n$ the number of fixed columns is $(3/4)n$ and in each $k \to k-1$ step this number is doubled, i. e., for $n/2+1 \leq k \leq (3/4)n$ the number of "free" columns is

$$2^{n-k} - 2^{(3/4)n-k} \cdot (3/4)n = 2^{n-k} - 2^{(3/4n-k+\log((3/4)n))} = \Omega(2^{n-k})$$

for n large enough. Replacing $L = 2^{n-k}$ by $\Omega(2^{n-k})$ in the calculation of the expectation does not change the asymptotic behavior. Thus, the expected number of blocks with exactly one changing position remains $\Omega(2^n)$ for every $n/2+1 \leq k \leq (3/4)n$.

Let X be the OBDD size of an interval graph randomly drawn from our construction.

From Theorem 3.2.5 we know that $X = \mathcal{O}(N \log N)$ with probability 1. We proved that $\mathbf{E}[X] = \Omega(N \log N)$. Now, we can use the reverse Markov inequality (see Theorem A.2.2 in the appendix) that says that for $\delta > 0$ the probability that X is larger than $(1-\delta) \cdot \mathbf{E}[X]$ is at least $\Omega(\delta)$. □

The upper bound for interval graphs can be easily adapted to interval bigraphs.

Theorem 3.2.8. *Let $\pi_{2,n}$ be the interleaved variable order with decreasing significance and $G = (A, B, E)$ be an interval bigraph with $N := |A| + |B|$ nodes. The $\pi_{2,n}$-OBDD size of χ_E can be bounded above by $\mathcal{O}(N \log N)$ and the worst-case OBDD size of χ_E is at least $\Omega(N)$.*

Proof. We label the nodes from A by $0, \ldots, |A| - 1$ according to their left interval endpoint and the nodes from B by $|A|, \ldots, |A| + |B| - 1$ also according to their left interval endpoint. The π-ordered adjacency matrix consists of a matrix U where the rows correspond to nodes from A and the columns correspond to nodes from B, the transpose U^T of U and two complete 0 matrices. We divide U into two part U_1, U_2 such that $U_1 + U_2 = U$. Every entry u_{ij} where the interval corresponding to the i-the row in U has a smaller left endpoint than the interval corresponding to the j-the column in U belongs to U_1. Similarly, in U_2 we put every entry u_{ij} where the interval corresponding to the i-the row in U has a larger left endpoint than the interval corresponding to the j-the column in U. This separation defines a clear cut of the matrix U similar to the diagonal of the adjacency matrix of an interval graph. Now, we can apply the same analysis as for interval graphs separately to U_1 and U_2 getting an upper bound of $\mathcal{O}(N)$ on the number of different blocks for every k. Since there are only 2^k blocks containing entries of both matrices U_1 and U_2, this gives us the desired bound of $\mathcal{O}(N \log N)$. The number of unlabeled interval bigraphs is bounded from below by $2^{\omega(N \log N)}$ (Theorem 2.2.20). Applying Corollary 2.3.11, gives a lower bound of $\Omega(N)$ for the worst-case OBDD size of χ_E. □

3.2.3. OBDD Size of Bipartite Permutation Graphs

In the next sections, we take advantage of the neighborhood structure of some graph classes to show good upper bounds on the OBDD size of the graphs as in the case of (unit) interval graphs. We show an upper bound of $\mathcal{O}(N/\log N)$ for bipartite permutation graphs in a similar way to unit interval graphs by using the strong order property from Theorem 2.2.12. Actually, we use an order with the so-called *forward-convex property* due to Lai and Wei [84] which coincides with the strong order if there is no isolated node. A *forward-convex labeling* of a bipartite graph $G = (A, B, E)$ is a labeling of A and B such that the order on A induced by the labeling fulfills the adjacency property and for every pair of nodes $b_j, b_{j'} \in B$ it holds that if $j < j'$ then $first(b_j) \leq first(b_{j'})$ and $last(b_j) \leq last(b_{j'})$ where $first(b)$ and $last(b)$ denotes the index of the first and last element of A, respectively, which is adjacent to b.

Theorem 3.2.9. *Let $\pi_{2,n}$ be the interleaved variable order with decreasing significance and $G = (A, B, E)$ be a bipartite permutation graph with $N := |A| + |B|$ nodes. The $\pi_{2,n}$-OBDD size of χ_E can be bounded above by $\mathcal{O}(N/\log N)$ and the worst-case OBDD size of χ_E is at least $\Omega(N/\log N)$.*

Proof. Consider the π-ordered adjacency matrix where we adapt the labeling from the forward-convex labeling of the graph while all nodes in A have smaller labels than all nodes in B. The matrix consists of a matrix U where the rows correspond to nodes from A and the columns correspond to nodes from B, the transpose U^T of U and two complete 0 matrices. Then for $b_j \in B$ we know that $first(b_j)$ and $last(b_j)$ is the first and last row, respectively, containing a 1 in the j-th column of U. Due to the forward-convex property, we also have that the sequence of "first" rows $\Big(first(b_0), \ldots, first(b_{|B|-1})\Big)$ and "end" rows $\Big(last(b_0), \ldots, last(b_{|B|-1})\Big)$ are monotonically increasing. For a fixed k, we look at the different blocks contained in the U submatrix of the π-ordered adjacency matrix. For a fixed j, let $(i_1, \ldots, i_s)$ be the indices of the different blocks of the form $B^k_{\bullet,j}$. Every changing position can cause at most two different blocks. Therefore, if $s \geq 4$ then there has to be at least one changing position in $B^k_{i_3,j}$. This means that the sequence of first or last rows is increasing within the columns of $B^k_{\bullet,j}$ by an additive term of at least 2^{n-k}. Thus, there can only be at most $\mathcal{O}(2^k)$ different blocks because otherwise the elements of one sequence would be greater than 2^n.

Every bipartite permutation matrix can be described by two monotone sequences of the first and last rows of the columns. This also holds for every block $B^k_{\bullet,\bullet}$. In the proof of the upper bound for unit intervals we have shown that there are at most $2^{2^{n-k+1}}$ monotone sequences of the form $(a_1, \ldots, a_{2^{n-k}})$ with $0 \leq a_i \leq 2^{n-k}$ for every i. Since here we have two monotone sequences, the number of different blocks is bounded from above by $\left(2^{2^{n-k+1}}\right)^2 = 2^{2^{n-k+2}}$. Similarly as for unit interval graphs, summing up everything gives us a $\mathcal{O}(N/\log N)$ upper bound on the $\pi_{2,n}$-OBDD size of χ_E.

For the lower bound, we know from Theorem 2.2.22 that the number $BP(N)$ of unlabeled bipartite permutation graphs of size N is given by

$$BP(N) = \begin{cases} \frac{1}{4}\left(C(N-1) + C(N/2-1) + \binom{N}{N/2}\right) & \text{if } N \text{ is even,} \\ \frac{1}{4}\left(C(N-1) + \binom{N-1}{(N-1)/2}\right) & \text{otherwise} \end{cases}$$

where $C(N) := \frac{1}{N+1}\binom{2N}{N}$ is the N-th Catalan number. Since $\binom{n}{k} \geq (\frac{n}{k})^k$, we have $BP(N) = 2^{\Omega(N)}$. Using Corollary 2.3.11, this implies that the worst-case OBDD size is bounded from below by $\Omega(N/\log N)$. □

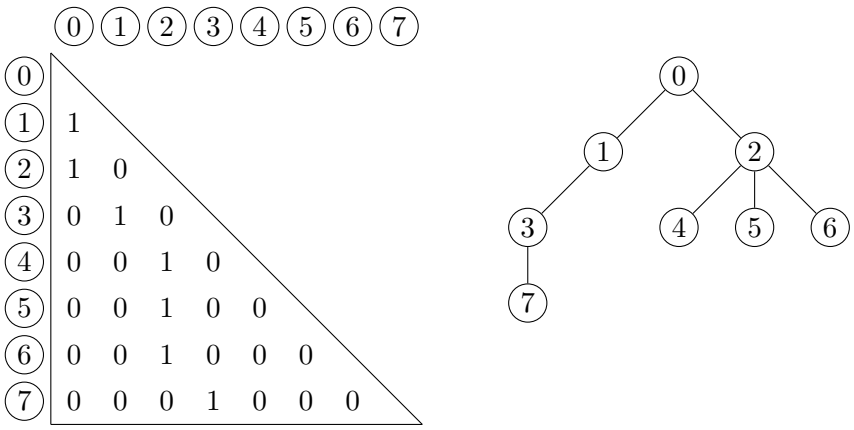


Figure 3.5.: A tree with a breadth first search labeling and the corresponding adjacency matrix

3.2.4. OBDD Size of Trees

An easy upper bound of trees is $\mathcal{O}(N \log N)$ because the number of edges is linear and the size of OBDDs is bounded by $\mathcal{O}(n \cdot |\chi_E^{-1}(1)|) = \mathcal{O}(|E| \log N)$. To improve this bound, we want to label the nodes of the tree such that the neighborhood restricted to nodes with a larger labeling is consecutive. In addition, we want that for two nodes v_i and $v_{i'}$ with $i' > i$ the largest label of the neighbors is increasing. For this, we root the tree at an arbitrary node and label this node by 0. Now, every node gets a label which is corresponding to the visit time of the node in a breadth first search starting in the root node. This means that the children of a node v_i get consecutive labels greater than i. This property allows us to apply similar arguments as for unit interval graphs to obtain an upper bound of $\mathcal{O}(N/\log N)$. The counting result by Otter (Theorem 2.2.17) implies that this upper bound is also asymptotically optimal.

Theorem 3.2.10. *Let $\pi_{2,n}$ be the interleaved variable order with decreasing significance and $G = (V, E)$ be a tree with N nodes. The $\pi_{2,n}$-OBDD size of χ_E can be bounded above by $\mathcal{O}(N/\log N)$. The worst-case OBDD size of χ_E is at least $\Omega(N/\log N)$.*

Proof. Let A_π be the π-ordered adjacency matrix of G where the nodes are labeled as described above: We root the tree at an arbitrary node and each node gets the label that corresponds to the visit time in a breadth first search starting in the root. For a fixed k and $\beta \in \{0,1\}^{n-k}$, we look at the blocks of the form $B^k_{|\alpha|_2,|\beta|_2}$ with $|\alpha|_2 > |\beta|_2$. We know that the neighborhood of every node restricted to larger labels is consecutive due to the BFS labeling. Since G is a tree, such neighborhoods of two different nodes v_i and $v_{i'}$ are disjoint and if $i' > i$ the labels of the children of v_i are smaller than the labels of the children of $v_{i'}$ (see Fig. 3.5). Let $B^k_{|\alpha_{max}|_2,|\beta|_2}$ be the non-zero block $B^k_{|\alpha|_2,|\beta|_2}$ with the largest value of α. The structure of the labels implies that the blocks $B^k_{|\alpha|_2,|\beta'|_2}$ with $|\beta'|_2 > |\beta|_2$ and $|\alpha_{max}|_2 > |\alpha|_2 > |\beta'|_2$ are completely 0. This means we can only have at most $\mathcal{O}(2^k)$ different blocks below the diagonal. Since the matrix is symmetric and the diagonal is part of exactly 2^k blocks, we have $\mathcal{O}(2^k)$

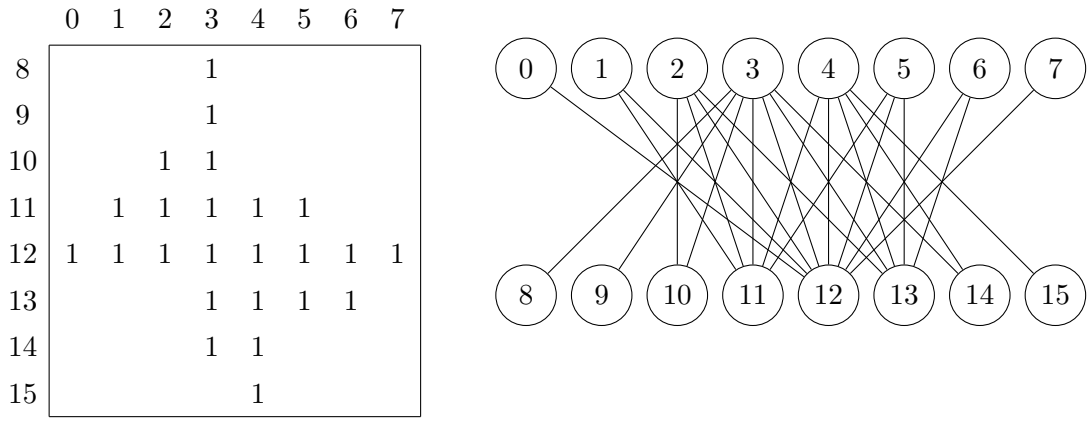


Figure 3.6.: A biconvex graph and the matrix U of the adjacency matrix with indices $j_f = 3$ and $j_l = 4$

different blocks in total. For large k, we can also use the bound for the number of different monotone sequences $(\alpha_1, \ldots, \alpha_{2^{n-k}})$ with $0 \leq \alpha_i \leq 2^{n-k}$ for every i which is $2^{2^{n-k+1}}$: The largest neighbor (with respect to the labeling) of node v_i (restricted to neighbors with larger label than i) is strictly increasing with ascending i if such a neighbor exists. Recall that these neighborhoods are disjoint. This means we can represent a block of the matrix by a monotone sequence $(\alpha_0, \ldots, \alpha_{2^{n-k}-1})$ where α_i is the largest neighbor of node v_i within the block. If such a neighbor does not exist we set $\alpha_i = \max\{\alpha_{i'} \mid i' < i \text{ and } v_{i'} \text{ has a neighbor in the block}\}$ (assume that v_0 has a neighbor in the block otherwise set $\alpha_0 = 0$). This gives us an upper bound of $\mathcal{O}(2^{2^{n-k+1}})$ which implies the $\mathcal{O}(N/\log N)$ bound of the OBDD size.

For the lower bound on the OBDD size of trees, we can use the result by Otter (Theorem 2.2.17) stating that there are at least $2^{\Omega(N)}$ unlabeled trees together with Corollary 2.3.11 to show that the worst-case OBDD size is at least $\Omega(N/\log N)$. □

3.2.5. OBDD Size of (Bi)Convex Graphs

Recall that the definition of a convex graph $G = (A, B, E)$ says that w.l.o.g. A fulfills the adjacency property, which means that nodes can be ordered such that the neighborhood of every node $u \in B$ is consecutive in this order. If we label the nodes from A according to this order, then each row in the matrix U, where the rows correspond to nodes from A and the columns correspond to nodes from B, has at most two changing positions. As we have seen in one of the last subsections, the actual adjacency matrix of G consists of two 0 submatrices, U, and U^T. Now, we can use the same proof idea as for interval graphs with 2 instead of 1 changing positions for every column.

Using the counting results from Theorem 2.2.19 we know that there are at least $2^{\Omega(N \log N)}$ unlabeled convex graphs with $\Theta(N)$ nodes. Plugging this into Corollary 2.3.11, we get a lower

bound of $\Omega(N)$. This gives us the following result.

Theorem 3.2.11. *Let $\pi_{2,n}$ be the interleaved variable order with decreasing significance and $G = (A, B, E)$ be a convex graph with $N := |A| + |B|$ nodes. The $\pi_{2,n}$-OBDD size of χ_E can be bounded above by $\mathcal{O}(N \log N)$. The worst-case OBDD size of χ_E is at least $\Omega(N)$.*

In a biconvex graph $G = (A, B, E)$ there is also an order of the nodes from B such that every neighborhood of a node from A is consecutive in this order. Now, if we label the nodes in B according to this order, we have that both rows and columns of the adjacency matrix have at most two changing positions. As for bipartite permutation graphs, we look at the sequence of first rows $(first(b_0), \ldots, first(b_{|B|-1})$ and last rows $(last(b_0), \ldots, last(b_{|B|-1})$. Since the neighborhoods of every node from A and B are consecutive in the corresponding order of A and B, there are indices $0 \leq j_f \leq |B| - 1$ and $0 \leq j_l \leq |B| - 1$ such that

$$\begin{aligned}
first(b_j) &\geq first(b_{j'}) \quad \text{for} \quad 0 \leq j < j' \leq j_f, \\
first(b_j) &\leq first(b_{j'}) \quad \text{for} \quad j_f \leq j < j' \leq |B| - 1, \\
last(b_j) &\geq last(b_{j'}) \quad \text{for} \quad 0 \leq j < j' \leq j_l, \text{ and} \\
last(b_j) &\leq last(b_{j'}) \quad \text{for} \quad j_l \leq j < j' \leq |B| - 1.
\end{aligned}$$

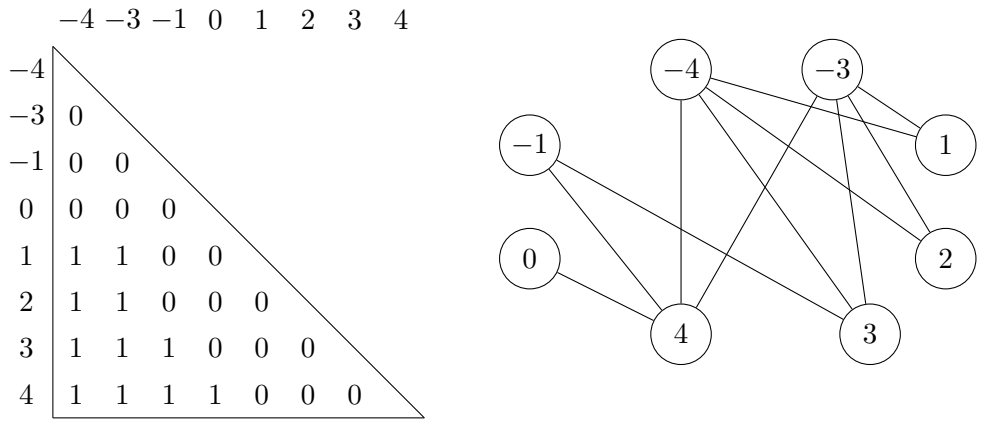
Fig. 3.6 shows an example of a biconvex graph and the values of these indices. This property was also observed by Lai and Wei [84] who used this to split the graph into two chain graphs and one bipartite permutation graph. Here, we directly use the sequence to bound the OBDD size of a biconvex graph. We can apply the same arguments as in the proof for bipartite permutation graphs for each possible pair of the four monotone sequences: If we have four or more different blocks of the form $B^k_{\bullet,j}$ for a fixed j then the values of at least one of the two sequences have to increase or decrease by an additive term of at least 2^{n-k}. This means that there cannot be more than $\mathcal{O}(2^k)$ different blocks because otherwise the elements of a sequence would be too large or too small.

Since bipartite permutation graphs are biconvex, we can bound the number of unlabeled biconvex graphs by the number of unlabeled bipartite permutation graphs. Thus, we get also a lower bound of $\Omega(N/\log N)$.

Theorem 3.2.12. *Let $\pi_{2,n}$ be the interleaved variable order with decreasing significance and $G = (A, B, E)$ be a biconvex graph with $N := |A| + |B|$ nodes. The $\pi_{2,n}$-OBDD size of χ_E can be bounded above by $\mathcal{O}(N/\log N)$. The worst-case OBDD size of χ_E is at least $\Omega(N/\log N)$.*

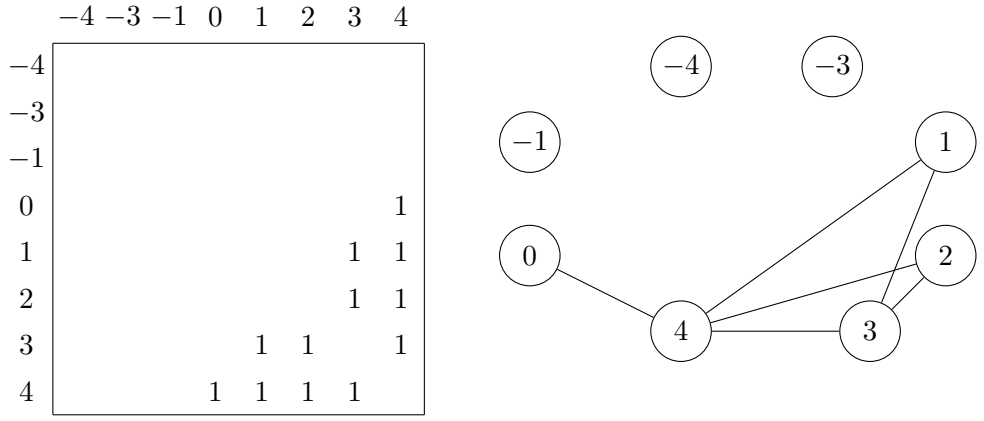
3.2.6. OBDD Size of Threshold and Chain Graphs

Bounding the size of an OBDD representing a chain or threshold graph is easy with the tools which we have developed in the last subsections. Recall that in both graph classes we have a



	−4	−3	−1	0	1	2	3	4
−4								
−3	0							
−1	0	0						
0	0	0	0					
1	1	1	0	0				
2	1	1	0	0	0			
3	1	1	1	0	0	0		
4	1	1	1	1	0	0	0	

Figure 3.7.: A chain graph with parameter $T = 4$ and the corresponding adjacency matrix



	−4	−3	−1	0	1	2	3	4
−4								
−3								
−1								
0								1
1							1	1
2							1	1
3					1	1		1
4				1	1	1	1	

Figure 3.8.: A threshold graph with parameter $T = 4$ and the corresponding adjacency matrix

node weight w_v for every $v \in V$ and two nodes u and v are adjacent if and only if $w_u + w_v \geq T$ (threshold graph) or $|w_u - w_v| \geq T$ (chain graph) for a real number T. We label the nodes according to their rank in the ascending order of node weights, i. e., for $v_i, v_j \in V$ with $i < j$ we have $w_{v_i} \leq w_{v_j}$. As before, we look at the column of the adjacency matrix corresponding to a fixed node v_j. In the case of threshold graphs, we observe the following property: Let $i > j$ which means $w_{v_i} \geq w_{v_j}$. Let $first(v_j)$ be the first row in the column of v_j such that the entry is 1. If such a row does not exist, we set $first(v_j) = N$. Since the node weights are increasing with larger node indices, we have $a_{ij} = 1$ for all $first(v_j) \leq i \leq N - 1$ with $i \neq j$ and $a_{ij} = 0$ otherwise (see Fig. 3.8). Furthermore, since $w_{v_{j'}} \geq w_{v_j}$ for every $j' > j$, if $w_{v_j} + w_u \geq T$ holds, then it is also $w_{v_{j'}} + w_u \geq T$ which implies that $first(v_{j'}) \leq first(v_j)$. Overall, we have $first(v_0) \geq first(v_1) \geq \ldots \geq first(v_{N-1})$ and we can use similar arguments as before to bound the OBDD size by $\mathcal{O}(N/\log N)$.

In the case of chain graphs, the value $|w_{v_j} - w_{v_i}|$ is increasing in i for a fixed $j < i$ because

$w_{v_j} \leq w_{v_i}$. This means that $first(v_0) \leq first(v_1) \leq \ldots \leq first(v_{N-1})$ where here $first(v_j)$ is the first row greater than j in the column of v_j such that the entry is 1 (see Fig. 3.7). Every entry after the $first(v_j)$-th row in the column of v_j is 1. Again, this gives us an upper bound of $\mathcal{O}(N/\log N)$ on the OBDD size.

A matching lower bound for the OBDD representation of chain and threshold graphs can be easily obtained by using the counting results from Theorem 2.2.24 and 2.2.23 that the number of unlabeled chain and threshold graphs of size N is bounded from below by $2^{\Omega(N)}$.

Theorem 3.2.13. *Let $\pi_{2,n}$ be the interleaved variable order with decreasing significance and $G = (V, E)$ be a threshold graph with N nodes. The $\pi_{2,n}$-OBDD size of χ_E can be bounded above by $\mathcal{O}(N/\log N)$. The worst-case OBDD size of χ_E is at least $\Omega(N/\log N)$.*

Theorem 3.2.14. *Let $\pi_{2,n}$ be the interleaved variable order with decreasing significance and $G = (V, E)$ be a chain graph with N nodes. The $\pi_{2,n}$-OBDD size of χ_E can be bounded above by $\mathcal{O}(N/\log N)$. The worst-case OBDD size of χ_E is at least $\Omega(N/\log N)$.*

3.3. OBDD-Based Algorithms on Interval Graphs

In this section, we want to develop a maximum matching algorithm on unit interval graphs and a coloring algorithm on unit and general interval graphs. Before we start with the algorithms, we have to investigate a special function class, which we will use in our algorithms, so-called multivariate threshold functions. This function class was first investigated in [131] to analyze the running time of an implicit topological sorting algorithm on grid graphs. Woelfel [131] looked at the OBDD size of these functions for the interleaved variable order with increasing significance, i. e., just the reverse of our variable order. Hosaka et al. [62] showed that the difference of the OBDD sizes for this two orders is at most $n - 1$. We can show that an OBDD using our variable order is not only small but can also be constructed efficiently which is important in view of the implementation.

3.3.1. Constructing OBDDs for Multivariate Threshold Functions

We start with the definition of multivariate threshold functions.

Definition 3.3.1 ([131])**.** A Boolean function $f : \{0,1\}^{kn} \to \{0,1\}$ with k input variable vectors $x^{(1)}, \ldots, x^{(k)} \in \{0,1\}^n$ of length n is called k-variate threshold function, if there exist a threshold $T \in \mathbb{Z}$ and $W \in \mathbb{N}$ and weights $w_1, \ldots, w_k \in \{-W, \ldots, W\}$ such that

$$f(x^{(1)}, \ldots, x^{(k)}) = 1 \Leftrightarrow \sum_{j=1}^{k} w_j \cdot |x^{(j)}|_2 \geq T.$$

The set of k-variate threshold functions $f \in B_{kn}$ with weight parameter W is denoted by $\mathbb{T}^W_{k,n}$.

Woelfel [131] showed that there exists an OBDD representing a multivariate threshold function $f \in \mathbb{T}^W_{k,n}$ of size $\mathcal{O}(k^2Wn)$ and such an OBDD can be constructed efficiently. Our proof for our variable order is similar to his proof: It is sufficient to look at the carry values of the sum $\sum_{j=1}^{k} w_j \cdot |x^{(j)}|_2 - T$ and, especially, at the carry value generated at the position with the most significance. Reading the bits with increasing significance, Woelfel showed that after each bit it is enough to store a number with absolute value $\mathcal{O}(kW)$ to compute the carry values. Here, we show that the influence of input bits with lower significance is small such that we can also bound the number which we have to store after each bit while we read the bits with decreasing significance.

Theorem 3.3.2. *Let $f \in \mathbb{T}^W_{k,n}$ be a k-variate threshold function with weight parameter $W \in \mathbb{N}$ and $\pi_{k,n}$ be the k-interleaved variable order where the variables are tested with decreasing significance. Then we can construct a $\pi_{k,n}$-OBDD representing f with width $\mathcal{O}(kW)$ and size $\mathcal{O}(k^2Wn)$ in time $\mathcal{O}(k^2Wn)$.*

Proof. Similar to the proof in [131], we choose $T_0, \ldots, T_{n-1} \in \{0,1\}$ and $T_n \in \mathbb{Z}$ such that $-T = \sum_{i=0}^{n} T_i \cdot 2^i$. Notice that the T_i are unique, and that $T_0, \ldots, T_{n-1}$ are the n least significant coefficients of $|T|$ in binary representation and T_n is the number of times that we need to add 2^n in order to make up for the missing coefficients in this binary representation. The function value of f is determined by the sign of $S := -T + \sum_{j=1}^{k} w_j \cdot |x^{(j)}|_2 = \sum_{i=0}^{n-1}(T_i + \sum_{j=1}^{k} w_j \cdot x_i^{(j)}) \cdot 2^i + T_n \cdot 2^n$. Now, we represent S in the same way as T, i. e., we define $S_0, \ldots, S_{n-1} \in \{0,1\}$ and $S_n \in \mathbb{Z}$ as the unique coefficients satisfying $S = \sum_{i=0}^{n} S_i \cdot 2^i$. We want to compute S_i step-by-step: Notice that S_i results from adding $T_i + \sum_{j=1}^{k} w_j \cdot x_i^{(j)}$ and the carry value which is generated at position $i-1$, and taking the remainder of the division of this sum by two. In particular, S_i is only influenced by factors of 2^j for $j \leq i$, and it holds that for $0 \leq i \leq n-1$

$$S_i := \left(c_{i-1} + T_i + \sum_{j=1}^{k} w_j x_i^{(j)} \right) \bmod 2 \quad \text{and}$$
$$c_i := \left\lfloor \left(c_{i-1} + T_i + \sum_{j=1}^{k} w_j x_i^{(j)} \right) /2 \right\rfloor$$

with $c_{-1} = 0$. Finally, we compute $S_n = c_{n-1} + T_n$. Now, we have $f(x^{(1)}, \ldots, x^{(k)}) = 1 \Leftrightarrow c_{n-1} \geq -T_n$, i. e., it is sufficient to compute the c_i values.

We rewrite the c_i to have them in a more convenient form. Notice that for $m, n \in \mathbb{N}$ and $x \in \mathbb{R}$ it holds that $\left\lfloor \frac{\lfloor x \rfloor + m}{n} \right\rfloor = \left\lfloor \frac{x+m}{n} \right\rfloor$ (see, e. g., [79]). So we have

$$\begin{aligned} c_1 &= \left\lfloor \left(c_0 + T_1 + \sum_{j=1}^{k} w_j x_1^{(j)} \right) /2 \right\rfloor \\ &= \left\lfloor \left(\left\lfloor \left(T_0 + \sum_{j=1}^{k} w_j x_0^{(j)} \right) /2 \right\rfloor + T_1 + \sum_{j=1}^{k} w_j x_1^{(j)} \right) /2 \right\rfloor \\ &= \left\lfloor \left(\left(T_0 + \sum_{j=1}^{k} w_j x_0^{(j)} \right) /2 + T_1 + \sum_{j=1}^{k} w_j x_1^{(j)} \right) /2 \right\rfloor \\ &= \left\lfloor \left(T_0 + \sum_{j=1}^{k} w_j x_0^{(j)} \right) /4 + \left(T_1 + \sum_{j=1}^{k} w_j x_1^{(j)} \right) /2 \right\rfloor . \end{aligned}$$

Let $c'_i = \frac{T_i + \sum_{j=1}^{k} w_j x_i^{(j)}}{2^{n-i}}$. Applying the above observation iteratively, we have

$$c_{n-1} = \left\lfloor \sum_{i=0}^{n-1} c'_i \right\rfloor .$$

According to our variable order, we have to compute c'_i backwards from $n-1$ to 0. This is possible because each c'_i only depends on i. We describe an algorithm that is divided into phases. In each phase, the algorithm is in a state Q, reads all k bits of the input variable vectors of the same significance and changes the state depending on the former state and the read bits. After phase i, the algorithm has the correct sum of the summands from $n-1$ to i. Notice that the bits with lesser significance can only add a value to S with bounded absolute value, so if the accumulated sum has a large enough absolute value, then we can already decide which sign S has.

Let us start with phase $n-1$ and state $Q = 0$. In phase $1 \leq i \leq n-1$ we compute the value of c'_i by reading $x_i^{(1)}, \ldots, x_i^{(k)}$:

1. If $c'_i + Q \geq -T_n + (kW+1)/2^{n-i}$ then change into the accepting state Q_{acc}.
2. If $c'_i + Q < -T_n - (kW+1)/2^{n-i}$ then change into the rejecting state Q_{rej}.
3. Otherwise update the state $Q = c'_i + Q$ and go to phase $i-1$.

In phase 0 we compute c'_0 and accept iff $\lfloor c'_0 + Q \rfloor \geq -T_n$.

If we reach phase 0 then the output is correct due to our above observations. So we have to show that we correctly accept/reject within phase i with $1 \leq i \leq n-1$. For $i = 0, \ldots, n-1$ it is $|c'_i| \leq \frac{kW+1}{2^{n-i}}$ because $T_i \in \{0,1\}$ and all weights are bounded by W and therefore

$$\left| \sum_{l=0}^{i} c'_l \right| \leq \sum_{l=0}^{i} \frac{kW+1}{2^{n-l}} = \frac{kW+1}{2^{n-i}} \sum_{l=0}^{i} \frac{1}{2^l} \leq \frac{kW+1}{2^{n-i}} \cdot 2 = \frac{kW+1}{2^{n-i-1}}.$$

I. e., if in phase i it is either $c'_i + Q \geq -T_n + (kW+1)/2^{n-i}$ or $c'_i + Q < -T_n - (kW+1)/2^{n-i}$, then we know that $\left\lfloor \sum_{i=0}^{n-1} c'_i \right\rfloor \geq -T_n$ or $\left\lfloor \sum_{i=0}^{n-1} c'_i \right\rfloor < -T_n$ respectively. So our algorithm works correctly.

Based on this algorithm, the construction of the $\pi_{k,n}$-OBDD is easy: Assume that we update the state immediately after reading an input variable. Then each state is represented by an OBDD node labeled by the variable which the algorithm will read next. The states for accepting and rejecting are represented by the sinks. The edges correspond to the state transition of the algorithm. If we are not in an accepting or rejecting state, we know that the state value is between $-T_n - \frac{kW+1}{2^{n-i-1}}$ and $-T_n + \frac{kW+1}{2^{n-i-1}} - 1$. We also know that $|c'_i| \leq \frac{kW+1}{2^{n-i}}$, i. e., the values computed in phase i have to be between

$$-T_n - \frac{kW+1}{2^{n-i-1}} - \frac{kW+1}{2^{n-i}} \text{ and } -T_n + \frac{kW+1}{2^{n-i-1}} - 1 + \frac{kW+1}{2^{n-i}}.$$

So all values are in the interval $I = [-T_n - \frac{3}{2}\frac{kW+1}{2^{n-i-1}}, -T_n + \frac{3}{2}\frac{kW+1}{2^{n-i-1}})$. The denominator of c'_i is an integer, i. e., only at most $2^{n-i} \cdot |I| = O(kW)$ values of I are possible during the computation. Therefore, we have an OBDD width of $\mathcal{O}(kW)$ and overall an OBDD size of $\mathcal{O}(k^2Wn)$. The construction algorithm is straightforward and has a running time which is linear to the OBDD size. □

The proof of Theorem 3.3.2 also showed that the complete OBDD width is bounded by $\mathcal{O}(kW)$. A binary synthesis of two functions with complete-OBDD width w_1 and w_2 has a complete-OBDD width of at most $w_1 \cdot w_2$ [117]. Since the complete OBDD size is an upper bound on the general OBDD size, we can compute a sequence of $\mathcal{O}(1)$ binary synthesis of multivariate threshold functions efficiently using the interleaved variable order with decreasing significance if k and W are constants.

We use the arithmetic notation in our algorithm instead of the functional notation whenever we use multivariate threshold functions or simple combination of multivariate threshold functions, e. g., we denote the conjunction of the multivariate threshold functions $f(x,y) = 1 \Leftrightarrow |x|_2 - |y|_2 \geq 1$ and $g(x,y) = 1 \Leftrightarrow |y|_2 - |x|_2 \geq -1$ by $|x|_2 - |y|_2 = 1$.

3.3.2. Maximum Matching on Unit Interval Graphs

Let $G = (V, E)$ be a unit interval graph where the nodes are labeled according to the sorted sequence of left endpoints. Our maximum matching algorithm is based on a simple observation that was also used in a parallel algorithm for this problem [32]: Assume that the unit interval graph is connected (otherwise this observation holds for every connected component). Then we have $\{v_i, v_{i+1}\} \in E$ for $i = 0, \ldots, N-2$. Assume that there is an $i \in \{0, \ldots, N-2\}$ such that $\{v_i, v_{i+1}\} \notin E$, then due to the connectivity there has to be another interval with left endpoint left of v_i or right of v_{i+1}, which intersects both intervals v_i and v_{i+1}. The length of

this interval would be larger than 1 which is a contradiction.

Algorithm 4 uses the characteristic function of the set of nodes besides the characteristic function χ_E. This is important if the number of nodes is not a power of two and we have assignments to the input variables which do not represent a node. Since we label our nodes according to their left endpoints, we have that the characteristic function of the node set is equal to $f(x) = 1 \Leftrightarrow |x|_2 < N$.

Algorithm 4 Implicit maximum matching algorithm for unit interval graphs

Input: Unit interval graph χ_E

Output: Matching χ_M

$\triangleright$ Compute path graph

$\chi_{\overrightarrow{E}}(x,y) = \chi_E(x,y) \wedge (|y|_2 - |x|_2 = 1)$

$\triangleright$ Compute set of starting nodes

$First(z) = (|z|_2 < N) \wedge \forall x : \overline{\chi_{\overrightarrow{E}}(x,z)}$

$\triangleright$ Compute set of reachable nodes

$S(z) = \exists z' : \chi_{\overrightarrow{E}}(z,z')$

$Reachable(x,y) = (|x|_2 \leq |y|_2) \wedge \forall z : (|x|_2 \leq |z|_2 < |y|_2) \Rightarrow S(z)$

$Reachable(x,y) = Reachable(x,y) \wedge (|x|_2 < N) \wedge (|y|_2 < N)$

$\triangleright$ Compute matching

$F(x) = \exists z, d : First(z) \wedge Reachable(z,x) \wedge (|x|_2 - |z|_2 = 2|d|_2)$

$M(x,y) = \chi_{\overrightarrow{E}}(x,y) \wedge F(x)$

$\chi_M(x,y) = M(x,y) \vee M(y,x)$

return $\chi_M(x,y)$

At first, the algorithm computes a directed path graph, i. e., a union of paths, which is a subgraph of the input graph and consists of the edges (x,y) with $|x|_2 - |y|_2 = 1$. For every connected component this path consists of all nodes within the component, as we have seen above. In general, maximum matchings on vertex disjoint paths can be computed with $\mathcal{O}(\log^2 N)$ functional operations [20]. Here, we know that every path P consists of a consecutive sequence of nodes, i. e., $P = (v_i, \ldots, v_k)$ for $0 \leq i \leq k \leq N-1$. We can use this information to lower the number of functional operations: We compute the set of nodes which are starting nodes of the paths. Then we want to compute the connected components of the graph. Usually, this is done by computing the transitive closure, which needs $\mathcal{O}(\log^2 N)$ operations. Again, we can do it better: Two nodes x and y of the unit interval graph are connected iff every node z with $|x|_2 \leq |z|_2 < |y|_2$ has a successor, i. e., there is an edge $(v_{|z|_2}, v_{|z|_2+1}) \in \overrightarrow{E}$. Having this information, we can compute the matching by adding every second edge of a path to the matching beginning with the first edge. To compute this set of edges on general paths, the distance of every node to the first node has to be computed. This can be done by an iterative squaring approach with $\mathcal{O}(\log^2 N)$ functional operations [20]. We

can easily determine the set of edges by comparing the difference of two node labels due to the structure of the paths.

Theorem 3.3.3. *Algorithm 4 computes a maximum matching for unit interval graphs using $\mathcal{O}(\log N)$ functional operations.*

Proof. As we have seen in the beginning of this subsection, every connected component has always a path which consists of consecutive nodes and visits every node in this component. The algorithm computes such paths and constructs a maximum matching of each path. Clearly, the union of these matchings is a maximum matching of the complete graph.

The number of functional operations is determined by the lines 2-4 where we use quantifications over $\mathcal{O}(\log N)$ variables. Otherwise, there is only a constant number of operations. □

3.3.3. Implicit Coloring of Interval Graphs

Coloring refers to the task to color the nodes of a graph using the least number of colors, such that all adjacent nodes have different colors. In the case of interval graphs, there is an easy greedy coloring algorithm: Sort the set of endpoints of the intervals (i. e., the set consists of both left and right endpoints) in ascending order. At the beginning all colors are on a stack. Then color the intervals sequentially by traversing the sorted list and using the first color available on the stack when the current element is a left endpoint. As soon as we visit a right endpoint, we push the used color onto the top of the stack. This greedy algorithm is optimal and can be implemented to run in linear time by determining the order without sorting [108]. The parallel algorithm in [134] assigns weights to the endpoints and computes prefix sums to simulate the stack. In our implicit algorithm we can do the simulation in a more direct manner: We call two intervals $I_i = [a_i, b_i]$ and $I_j = [a_j, b_j]$ *related* iff $b_i < a_j$ and I_j is the first interval with the same color as I_i in the greedy algorithm. The following easy observation helps us to compute this "related" relation implicitly.

Observation 3.3.4. *The intervals $I_i = [a_i, b_i]$ and $I_j = [a_j, b_j]$ are related iff the number of right endpoints r with $b_i < r < a_j$ is equal to the number of left endpoints l with $b_i < l < a_j$ and for all intervals $I_{j'} = [a_{j'}, b_{j'}]$ with $b_i < a_{j'} < a_j$ the number of right endpoints r with $b_i < r < a_{j'}$ is not equal to the number of left endpoints l with $b_i < l < a_{j'}$*

The first property in the observation ensures that the intervals get the same color by the greedy algorithm while the second property means that there is not another interval with a smaller left endpoint with the same color. Now, in the case of unit intervals we want to show how we can compute a function $RELATED(x, y)$, which is 1 iff the intervals $I_{|x|_2}$ and $I_{|y|_2}$ are related. The general case is discussed later in this section. As before, the intervals are labeled according to their left endpoints. Let $RE(x, y, l) = 1$ iff $|x|_2 \leq |y|_2$ and the number of right endpoints between $b_{|x|_2}$ and $a_{|y|_2}$ is equal to $|l|_2$. Similarly, let $LE(x, y, l) = 1$ iff $|x|_2 \leq |y|_2$ and the number of left endpoints between $b_{|x|_2}$ and $a_{|y|_2}$ is equal

to $|l|_2$. Let $\chi_E(x,y)$ be the characteristic function of the edge set of a unit interval graph $G=(V,E)$ and $\chi_{E^c}(x,y)$ the characteristic function of the edge set of the complement graph, i. e., $E^c = \{(u,v) \mid u \neq v \text{ and } (u,v) \notin E\}$. Then we can compute $RE(x,y,l)$ and $LE(x,y,l)$ in the following way:

$$\begin{aligned} H_1(x,y,z) &= (|x|_2 \leq |z|_2 < |y|_2) \wedge \chi_{E^c}(z,y) \\ RE(x,y,l) &= (|x|_2 \leq |y|_2) \wedge \\ &\quad \Big[\exists z : H_1(x,y,z) \wedge (|z|_2 - |x|_2 = |l|_2) \wedge \overline{\exists z' : H_1(x,y,z') \wedge (|z'|_2 > |z|_2)}\Big] \end{aligned}$$

$$\begin{aligned} H_2(x,y,z) &= (|x|_2 < |z|_2 \leq |y|_2) \wedge \chi_{E^c}(x,z) \\ LE(x,y,l) &= (|x|_2 \leq |y|_2) \wedge \\ &\quad \Big[\exists z : H_2(x,y,z) \wedge (|y|_2 - |z|_2 = |l|_2) \wedge \overline{\exists z' : H_2(x,y,z') \wedge (|z'|_2 < |z|_2)}\Big]. \end{aligned}$$

The right endpoint of an interval $I_{|z|_2}$ is greater than or equal to $b_{|x|_2}$ and less than $a_{|y|_2}$ iff $|x|_2 \leq |z|_2 < |y|_2$ and $I_{|z|_2}$ does not intersect $I_{|y|_2}$. Since we are dealing with unit interval graphs, if for some z with $|x|_2 \leq |z|_2 < |y|_2$ the intervals $I_{|z|_2}$ and $I_{|y|_2}$ do not intersect, then this also holds for all z' with $|x|_2 \leq |z'|_2 < |z|_2$. I. e., the maximal value of $|z|_2 - |x|_2$ over all z with the above property is equal to the number of right endpoints between $b_{|x|_2}$ and $a_{|y|_2}$ and, therefore, we compute the function $RE(x,y,l)$ correctly. Similar arguments show that $LE(x,y,l)$ is computed correctly, too. Together with Observation 3.3.4, we can compute the function $RELATED(x,y)$ as follows:

$$\begin{aligned} RELATED(x,y) &= \exists l : RE(x,y,l) \wedge LE(x,y,l) \wedge \\ &\quad \overline{\exists z, l' : (|z|_2 < |y|_2) \wedge RE(x,z,l') \wedge LE(x,z,l')} \end{aligned}$$

Now, we have to compute the sequence of related intervals, which is nothing more than the transitive closure of the related relation. This can be computed with $\mathcal{O}(\log^2 N)$ functional operations. Finally, we have to assign a color to each interval, such that all intervals in a sequence of related intervals are getting the same color. In order to do this, we compute an order on the sequences of related intervals and assign the colors to the sequences according to that order by using the *EnumerateOrder* procedure from Section 2.3. The order on the sequences is given by the order on the minimal interval number within the sequences. Putting all together, Algorithm 5 computes a coloring on a unit interval graph.

Theorem 3.3.5. *Algorithm 5 computes a coloring of a unit interval graph using the minimal number of colors and $\mathcal{O}(\log^2 N)$ functional operations.*

Proof. That the output is a coloring with the minimal number of colors follows directly from correctness of the greedy algorithm. The number of functional operations is dominated by the *TransitiveClosure* and *EnumerateOrder* procedures. As we have seen in Section 2.3, both procedures need $\mathcal{O}(\log^2 N)$ functional operations. □

Algorithm 5 Implicit coloring algorithm for unit interval graphs

Input: Unit interval graph (χ_E, χ_V)

Output: Coloring $COLOR(x,l)$ with $COLOR(x,l) = 1$ iff $I_{|x|_2}$ has color $|l|_2$

$\triangleright$ Complement graph

$\chi_{E^c}(x,y) = \chi_V(x) \wedge \chi_V(y) \wedge \overline{\chi_E(x,y)} \wedge (|x|_2 \neq |y|_2)$

$\triangleright$ Auxiliary functions to compute the number of right/left endpoints

$\triangleright$ between two intervals

$H_1(x,y,z) = (|x|_2 \leq |z|_2 < |y|_2) \wedge \chi_{E^c}(z,y)$

$H_2(x,y,z) = (|x|_2 < |z|_2 \leq |y|_2) \wedge \chi_{E^c}(x,z)$

$\triangleright$ Number of right endpoints between $b_{|x|_2}$ and $a_{|y|_2}$

$$RE(x,y,l) = (|x|_2 \leq |y|_2) \wedge \left[\exists z : H_1(x,y,z) \wedge (|z|_2 - |x|_2 = |l|_2) \wedge \overline{\exists z' : H_1(x,y,z') \wedge (|z'|_2 > |z|_2)}\right]$$

$\triangleright$ Number of left endpoints between $b_{|x|_2}$ and $a_{|y|_2}$

$$LE(x,y,l) = (|x|_2 \leq |y|_2) \wedge \left[\exists z : H_2(x,y,z) \wedge (|y|_2 - |z|_2 = |l|_2) \wedge \overline{\exists z' : H_2(x,y,z') \wedge (|z'|_2 < |z|_2)}\right]$$

$\triangleright$ Compute related intervals

$$RELATED(x,y) = \exists l : RE(x,y,l) \wedge LE(x,y,l) \wedge \overline{\exists z, l' : (|z|_2 < |y|_2) \wedge RE(x,z,l') \wedge LE(x,z,l')}$$

$\triangleright$ Compute set of intervals with the same color

$SAMECOLOR(x,y) = TransitiveClosure(RELATED(x,y) \vee (|x|_2 = |y|_2))$

$\triangleright$ Order these sets

$FIRST(x) = \overline{\exists x' : SAMECOLOR(x',x) \wedge (|x'|_2 < |x|_2)}$

$$COLORORDER(x,y) = \exists x', y' : SAMECOLOR(x',x) \wedge FIRST(x') \wedge SAMECOLOR(y',y) \wedge FIRST(y') \wedge (|x'|_2 < |y'|_2)$$

$\triangleright$ Assign the colors

$COLOR(x,l) = EnumerateOrder(COLORORDER(x,y))$

return $COLOR(x,l)$

The only difference between the unit interval and the general case is the computation of the functions LE and RE (this is the only place where we need the unity property). What we actually need is an order on the sequence of right endpoints to compute RE and an order on the left endpoints of the intervals to compute LE (and in the case of unit intervals both orders are the same). Assuming that we label the intervals according to their left endpoints, we only need to compute the order on the right endpoints. Let $EO(x,y)$ be this order, i. e., $EO(x,y) = 1$ iff $b_{|x|_2} \leq b_{|y|_2}$. Assume that the left endpoints of the intervals are the integers $0, \ldots, N-1$. Recall the structure of the adjacency matrix of an interval graph from section 3.2.2: We know that the interval with left endpoint $i \in \{0, \ldots, N-1\}$ has a maximal value j such that I_i and I_k intersect for all $i \leq k \leq j$. Therefore, the right endpoint of I_i has to be in $[j, j+1)$. Let j and j' be the maximal values such that I_i intersects all I_k with $i \leq k \leq j$ and $I_{i'}$ intersects all I_k with $i' \leq k \leq j'$, respectively. If $j < j'$ ($j > j'$), then $b_i < b_{i'}$ ($b_i > b_{i'}$). If $j = j'$, then we can break ties arbitrary (e. g., $b_i \leq b_{i'}$ iff $i \leq i'$). Now, we can compute

$EO(x, y)$ as follows:

$$\begin{aligned} H(x, y, x', y') &= (|x|_2 \leq |x'|_2) \wedge (|y|_2 \leq |y'|_2) \wedge \chi_E(x, x') \wedge \chi_E(y, y') \\ EO(x, y) &= \exists x', y' : H(x, y, x', y') \wedge (|x'|_2 < |y'|_2 \vee (|x'|_2 = |y'|_2 \wedge |x|_2 < |y|_2)) \wedge \\ &\quad \overline{\exists x'', y'' : H(x, y, x'', y'') \wedge ((|x''|_2 > |x'|_2) \vee (|y''|_2 > |y'|_2))} \end{aligned}$$

Notice that this order on the right endpoints does not have to be the same order on the original right endpoints. But, as we have shown, there is an interval representation of the graph, such that the left and right endpoints are ordered according to labels of the nodes and $EO(x, y)$, respectively. Finally, we have to compute $EEO(x, l) = EnumerateOrder(EO(x, y))$ and, with it, we get $RE(x, y, l)$ for general interval graphs:

$$\begin{aligned} H_1(x, y, z) &= \overline{EO(z, x)} \wedge EO(z, y) \wedge \chi_{E^c}(z, y) \\ RE(x, y, l) &= (|x|_2 < |y|_2) \wedge [\exists z, l_1, l_2 : H_1(x, y, z) \wedge EEO(x, l_1) \wedge \\ &\quad EEO(z, l_2) \wedge (|l_2|_2 - |l_1|_2 = |l|_2) \wedge \overline{\exists z' : H_1(x, y, z') \wedge EO(z, z')}\big] \end{aligned}$$

Since all additional operations are dominated by the *EnumarateOrder* procedure, we get the same result as for unit intervals.

Theorem 3.3.6. *Algorithm 5 with the modified computation of $RE(x, y, l)$ outputs a coloring of an interval graph using the minimal number of colors and $\mathcal{O}(\log^2 N)$ functional operations.*

4. Randomized OBDD-Based Algorithms

In this chapter, we introduce the new concept of randomized OBDD-based algorithms. In the first section, we start with an overview of related work regarding results on the OBDD size of random functions. We also discuss known explicit randomized algorithms that use random bits which are not completely independent. In Section 4.2, we investigate the OBDD size of random functions where the function values are subject to limited independence, namely the well known notion of k-wise independence. Then we give a construction of a random OBDD representing a random function which exhibits pseudorandom properties also known as almost k-wise independence. In Section 4.4, we briefly discuss the extension of Sawitzki's structural result on the relation between OBDD-based algorithms and parallel algorithms to the randomized setting. Finally, we present randomized OBDD-based algorithms for the maximal matching problem and for the minimum spanning tree problem.

4.1. Related Work and Contribution

Recall that a random function $f : S \to B_n$ over a sample space S induces 2^n random variables $X_0(s) := f_s([0]_2), \ldots, X_{2^n-1}(s) = f_s([2^n - 1]_2)$ and we say that f is (almost) k-wise independent if the random variables are (almost) k-wise independent. If we want to use f in an OBDD-based algorithm, we need an efficient OBDD construction of f_s for every $s \in S$. In particular, this implies that f_s has to be representable by an OBDD of small size for every $s \in S$. But, obviously, if the function values are completely independent, i. e., $k = 2^n$, then the OBDD (and even the more general FBDD) size of f_s is exponentially large with an overwhelming probability [127]. Kabanets [71] constructed simple Boolean functions which are hard for FBDDs by using (almost) k-wise independent random functions with $k = \Theta(n)$ and showed that the probability tends to 1 as n grows that the size is $\Omega(2^n/n)$. In terms of upper bounds, we have seen in Section 2.4 that (ε, k)-wise independent functions can be represented by Boolean formulas of size $\mathcal{O}(n \log^2 k \log \frac{1}{\varepsilon})$ [114].

Small probability spaces as k-wise independent random variables can also be used for a succinct representation of a random string of length 2^n, e. g., in streaming algorithms [6], or for derandomization [3, 89]. The randomized parallel algorithms from [3, 89] compute a maximal independent set (MIS) of a graph using only pairwise independent random variables ($k = 2$). An independent set is a set of nodes where no two nodes are adjacent. These algorithms can also be used to compute a maximal matching. The computation of a MIS has been exten-

sively studied in the area of distributed algorithms [10, 86]. In the distributed setting a set of processors can exchange messages over channels. The communication network is modeled as a graph where the nodes correspond to the processors and the possible communication channels are represented by edges. An optimal randomized distributed MIS algorithm that uses completely independent random bits was presented in [99] where the time and bit complexity (bits per channel) is $\mathcal{O}(\log N)$. Using completely independent random bits, Israeli and Itai [66] gave a randomized parallel algorithm computing a maximal matching in time $\mathcal{O}(\log N)$.

In Section 4.2, we show that the OBDD and FBDD size for k-wise independent functions with $k \geq 4$ is at least $2^{\Omega(n+\log(p'))}$ with $p = \mathbf{Pr}[f_s(x) = 1]$ and $p' = 2p(1-p)$. We give an efficient construction of OBDDs for 3-wise independent random functions which is based on the known construction of 3-wise independent random variables using BCH-schemes [3] from Section 2.4. In Section 4.3, we investigate a simple construction of a random OBDD which generates almost k-wise independent random functions and has size $\mathcal{O}((kn)^2/\varepsilon)$. Reading the actual value of the i-th random bit is an evaluation of the function on input i which can be done in $\mathcal{O}(n)$ time, i. e., the time is independent of both k and ε. This construction can be used as a distribution on graphs representable by OBDDs of small size what enables us to use it as an input distribution for our implicit algorithm in the experimental evaluation in Chapter 5. In Section 4.4, we start with the extension of Sawitzki's result [116, 118] on the equivalence of parallel algorithms and implicit algorithms to the randomized setting. We show that randomized parallel algorithms are equivalent to randomized implicit algorithms. Then we continue with a discussion how to use random functions in OBDD-based algorithms and what are the strengths and weaknesses of this model. Motivated by this discussion, we see that we can design a simple randomized OBDD-based algorithm for computing a minimum spanning tree in a weighted graph by using the well-known randomized algorithm by Karger, Klein, and Tarjan [76] which runs in expected linear time. Nevertheless, we cannot give a small bound on the number of functional operations and we leave it to Chapter 5 to see whether it can outperform the known deterministic algorithm by Bollig [15] using $\mathcal{O}(\log^3 N)$ functional operations. Then we present a simple randomized maximal matching algorithm that uses only $\mathcal{O}(\log^3 N)$ functional operations in expectation and functions with at most $3 \log N$ variables. As we know from Section 2.3, this is better than the algorithms by Hachtel and Somenzi [57] with $\mathcal{O}(N \log N)$ operations and $3 \log N$ variables as well as the algorithm by Bollig and Pröger [20] with $\mathcal{O}(\log^4 N)$ operations and $6 \log N$ variables. This algorithm can easily be extended to the MIS problem and can be implemented as a parallel algorithm using $\mathcal{O}(\log N)$ time in expectation or as a distributed algorithm with $\mathcal{O}(\log N)$ expected time and bit complexity. To the best of the author's knowledge, this is the first (explicit or implicit) maximal matching algorithm that does not need any knowledge about the graph (like size or node degrees) while only using pairwise independent random variables.

4.2. OBDD Size of k-Wise Independent Random Functions

We start with an upper bound on the OBDD size of 3-wise independent random functions using the BCH scheme by Alon, Babai, and Itai [3]: For a random vector $r = \left[r_0, r^{(1)}\right]$ with $r_0 \in \{0,1\}$ and $r^{(1)} \in \{0,1\}^n$ and for $1 \leq i \leq 2^n - 1$, the BCH scheme defines $X_i = IP_{n+1}(r, [1, [i]_2])$ which are 3-wise independent random variables. We want to give a construction of a 3-wise independent random function $f : S \to B_n$ such that for every $s \in S$ the OBDD representing f_s can be efficiently constructed. We cannot use the BCH scheme directly because we need 2^n random variables for the random function whereas the BCH scheme defines only $2^n - 1$ variables. However, adding the variable $X_0 = r_0 = IP_{n+1}(r, [1, [0]_2])$ yields the desired construction what is easy to prove directly.

Lemma 4.2.1. *Let $r = \left[r_0, r^{(1)}\right] \in \{0,1\}^{n+1}$ be a random vector. For $0 \leq i \leq 2^n - 1$ the random variables defined by*

$$X_i = IP_{n+1}(r, [1, [i]_2])$$

are 3-wise independent.

Proof. Let $Y_i = IP(r^{(1)}, [i]_2) = X_i \oplus r_0$. Due to [3] we know that three variables X_i, X_j, X_k with non-zero indices are independent. Let X_0, X_i, X_j be three arbitrary and random variables with $i \neq j$, $i \neq 0$, and $j \neq 0$. For every choice of $b_1, b_2, b_3 \in \{0,1\}$, we have

$$\begin{aligned}
\mathbf{Pr}[X_0 = b_1, X_i = b_2, X_j = b_3] &= \mathbf{Pr}[X_0 = b_1 \mid X_i = b_2, X_j = b_3] \cdot \mathbf{Pr}[X_i = b_2, X_j = b_3] \\
&= \mathbf{Pr}[X_0 = b_1 \mid X_i = b_2, X_j = b_3] \cdot \frac{1}{4} \\
&= \frac{1}{4} \cdot \mathbf{Pr}[Y_0 = b_1 \mid Y_i = b_2, Y_j = b_3, r_0 = 0] \cdot \mathbf{Pr}[r_0 = 0] \\
&\quad + \frac{1}{4} \cdot \mathbf{Pr}\left[Y_0 = \overline{b_1} \mid Y_i = \overline{b_2}, Y_j = \overline{b_3}, r_0 = 1\right] \cdot \mathbf{Pr}[r_0 = 1] \\
&= \frac{1}{8}
\end{aligned}$$

where the last equality follows from the fact that $Y_0 = 0$ and either $b_1 = 0$ or $\overline{b_1} = 0$. Since $\mathbf{Pr}[X_0 = b_1] = \mathbf{Pr}[r_0 = b_1] = 1/2$ the probability $\mathbf{Pr}[X_0 = b_1, X_i = b_2, X_j = b_3]$ is equal to $\mathbf{Pr}[X_0 = b_1] \cdot \mathbf{Pr}[X_i = b_2] \cdot \mathbf{Pr}[X_j = b_3]$. □

Now, it is straightforward to use this as a 3-wise independent random function and to give an efficient OBDD construction for every choice of the random vector r.

Algorithm 6 RandomFunc(x, n)

Input: Variable vector x of length $n \in \mathbb{N}$
Output: 3-wise independent function $f_r(x)$

Let $r_0, \ldots, r_n$ be $n+1$ independent random bits
$f_r(x) = \bigoplus_{i=0}^{n-1}(r_i \wedge x_i) \oplus r_n$
return $f_r(x)$

Theorem 4.2.2. *Let $\varepsilon > 0$, $n \in \mathbb{N}$, p be a probability with $0 < p \leq 1/2$, and let π be a variable order on the variables $\{x_0, \ldots, x_{n-1}\}$.*

1. *We can construct a π-OBDD representing a 3-wise independent function $f : S \to B_n$ with $S = \{0,1\}^{n+1}$ in time $\mathcal{O}(n)$ such that for every $x \in \{0,1\}^n$*

$$\mathbf{Pr}_{r \in \{0,1\}^{n+1}}[f_r(x) = 1] = 1/2,$$

and the size of the π-OBDD representing f_r is $\mathcal{O}(n)$ with width 2 for every $r \in \{0,1\}^{n+1}$.

2. *We can construct a π-OBDD representing a 3-wise independent function $f : S \to B_n$ with $S = \{0,1\}^{t(n+1)}$ where $t = \lceil -\log p - \log \varepsilon \rceil$ in time $\mathcal{O}(\frac{n}{p \cdot \varepsilon})$ such that for every $x \in \{0,1\}^n$*

$$p \leq \mathbf{Pr}_{r \in \{0,1\}^{n+1}}[f_r(x) = 1] \leq (1+\varepsilon) \cdot p,$$

and the size of the π-OBDD representing f_r is bounded above by $\mathcal{O}(\frac{n}{p \cdot \varepsilon})$ for every $r \in \{0,1\}^{n+1}$.

Proof. 1. This is an easy implication of the construction from Lemma 4.2.1 which says that for random $r = (r_0, \ldots, r_n) \in \{0,1\}^{n+1}$ the random variables $X_i(r) = IP(r, [1, [i]_2])$ for $0 \leq i \leq 2^n - 1$ are 3-wise independent and $\mathbf{Pr}[X_i = 1] = 1/2$ for every i. We define $f_r(x) = X_{|x|_2}(r)$ (see Algorithm 6). As described in the preliminaries, the function $IP(x, y)$ can be represented by an OBDD of width 2 and size $\mathcal{O}(n)$ when one input vector is replaced by a constant. The construction of the OBDD is straightforward (see, e. g., Fig. 2.3) and can be done in time $\mathcal{O}(n)$.

2. We round the binary representation of p to $t = \lceil -\log p - \log \varepsilon \rceil$ decimal places and call the result p'. Let $p'_1, \ldots, p'_t$ be the decimal places of p'. Then it holds $p \leq p' \leq p + 2^{-t} \leq (1+\varepsilon)p$. Let $f_{r_1}, \ldots f_{r_t}$ be t independent random functions which are drawn according to the construction in 1. We can construct an OBDD which evaluates these functions simultaneously on an input x and ends in the 1-sink iff $(f_{r_1}(x), \ldots, f_{r_t}(x)) \leq (p'_1, \ldots, p'_t)$. Since a single random function has width 2, this OBDD has size $\mathcal{O}(n \cdot 2^t) = \mathcal{O}(\frac{n}{p \cdot \varepsilon})$. The probability that $(f_{r_1}(x), \ldots, f_{r_t}(x)) \leq (p'_1, \ldots, p'_t)$ is $|(p'_1, \ldots, p'_t)|_2 / 2^t = p'$. □

Can we also construct small OBDDs for k-wise independent random variables with $k \geq 4$? The BCH scheme needs multiplication in a finite field to generate k-wise independent variables with

$k \geq 4$ which is unlikely to be representable by OBDDs of small size because computing specific bits of the result of a binary multiplication is already hard for OBDDs (see, e. g., [128]). In the next theorem we show that for a fixed variable order π, every k-wise independent random function with $k \geq 4$ inherently has large π-OBDD size regardless of the used construction. In order to show this we need a technical lemma that proves some properties of the subfunctions of a k-wise independent function.

Lemma 4.2.3. *Let $f : S \to B_n$ be a k-wise independent random function over a sample space S with $k \geq 4$ and $\mathbf{Pr}[f_s(x) = 1] = p$ for all $x \in \{0,1\}^n$. Let π be a fixed variable order. For $s \in S$, $l \in [n]$, and $\alpha \in \{0,1\}^l$ let $f_{s|\alpha}\{0,1\}^{n-l} \to \{0,1\}$ be the subfunction of f_s where the first l variables $x_{\pi(0)}, \ldots, x_{\pi(l-1)}$ are fixed according to α, i.e. $f_{s|\alpha}(z) = f_{s|x_0=\alpha_0,\ldots,x_{l-1}=\alpha_{l-1}}(z)$. Further, let C_l be the number of collisions of the form $f_{s|\alpha} = f_{s|\alpha'}$ with $\alpha \neq \alpha'$, i. e.,*

$$C_l := \left|\left\{(\alpha,\alpha') \mid \alpha,\alpha' \in \{0,1\}^l, \alpha \neq \alpha', \text{ and } f_{s|\alpha} = f_{s|\alpha'}\right\}\right|$$

and D_l be the number of different subfunctions $f_{s|\alpha}$. This means

$$D_l := \left|\left\{\left[f_{s|\alpha}\right] \mid \alpha \in \{0,1\}^l\right\}\right|$$

where $\left[f_{s|\alpha}\right] := \left\{\alpha' \in \{0,1\}^l \mid f_{s|\alpha} = f_{s|\alpha'}\right\}$ is the equivalence class of f_α with respect to function equality. Then we have $D_l \geq \dfrac{2^l}{\sqrt{2C_l+1}}$ and it holds

$$\mathbf{E}[C_l] \leq \frac{2^{2l}}{2^{n-l}\cdot p'} \qquad \text{and} \qquad \mathbf{E}[D_l] \geq \frac{2^l}{\sqrt{2\cdot\mathbf{E}[C_l]+1}} \geq \frac{1}{\sqrt{\frac{2}{2^n\cdot p'}2^{l/2}+\frac{1}{2^l}}}$$

where $p' = 2p(1-p)$.

Proof. For the sake of simplicity we omit the index s of f_s and write f_α to denote $f_{s|\alpha}$. Now we fix two different assignments α, α' and define 2^{n-l} random variables $D(z) := D_{\alpha,\alpha'}(z)$ such that $D(z) = 1$ iff $f_\alpha(z) \neq f_{\alpha'}(z)$. Since the function values of the subfunctions are also k-wise independent, we have for every $z \in \{0,1\}^{n-l}$

$$\mathbf{E}[D(z)] = \mathbf{Pr}[D(z) = 1] = 2p(1-p) := p' \text{ and}$$

$$\mathbf{Var}[D(z)] = \mathbf{E}\left[D(z)^2\right] - \mathbf{E}[D(z)]^2 = \mathbf{E}[D(z)] - \mathbf{E}[D(z)]^2 = p'(1-p').$$

Let $D = \sum_z D(z)$. By definition of D, we have $\mathbf{Pr}[f_\alpha = f_{\alpha'}] = \mathbf{Pr}[D = 0]$ for a fixed pair (α,α') and the latter term can be bounded from above by the probability that the difference between D and $\mathbf{E}[D]$ is at least $\mathbf{E}[D]$, i. e.,

$$\mathbf{Pr}[D = 0] \leq \mathbf{Pr}[|D - \mathbf{E}[D]| \geq \mathbf{E}[D]].$$

Each random variable $D(z)$ depends on two function values, i. e., these variables are $k' = \lfloor k/2 \rfloor$-wise independent. Since $k' \geq 2$, we can use Chebyshev´s inequality (Theorem 2.4.3) to get

$$\mathbf{Pr}\left[|D - E[D]| \geq \mathbf{E}[D]\right] \leq \frac{\mathbf{Var}[D]}{\mathbf{E}[D]^2} = \frac{\sum_z \mathbf{Var}[D(z)]}{(2^{n-l} \cdot p')^2} = \frac{2^{n-l} \cdot p' \cdot (1-p')}{(2^{n-l} \cdot p')^2} \leq \frac{1}{2^{n-l} \cdot p'}.$$

This implies that

$$\mathbf{E}[C_l] \leq \frac{\binom{2^l}{2}}{2^{n-l} \cdot p'} \leq \frac{2^{2l}}{2^{n-l} \cdot p'}. \tag{4.1}$$

Recall that $[f_\alpha] := \{\alpha' \in \{0,1\}^l \mid f_\alpha = f_{\alpha'}\}$ is the equivalence class of f_α with respect to function equality which consists of the assignments α' with $f_\alpha = f_{\alpha'}$. Let $M_l = \max_\alpha |[f_\alpha]|$ be the random variable for the size of the largest equivalence class. Since every equivalence class of size $s \geq 2$ causes $\binom{s}{2}$ collisions and no collisions otherwise, we have

$$\begin{aligned} C_l &\geq \frac{M_l(M_l-1)}{2} \geq \frac{(M_l-1)^2}{2} \\ \Leftrightarrow \sqrt{2 \cdot C_l} + 1 &\geq M_l. \end{aligned} \tag{4.2}$$

Let $\alpha_1, \ldots, \alpha_{D_l} \in \{0,1\}^l$ be representative assignments for the different equivalence classes. Then it holds that

$$\sum_{i=1}^{D_l} |[f_{\alpha_i}]| = 2^l.$$

Therefore, we have $M_l \cdot D_l \geq 2^l$ which is equivalent to $D_l \geq 2^l / M_l$ (note that $M_l \geq 1$). This means that we can bound the expected value of D_l by

$$\mathbf{E}[D_l] \geq \mathbf{E}\left[\frac{2^l}{M_l}\right] \geq \frac{2^l}{\mathbf{E}[M_l]} \tag{4.3}$$

$$\geq \frac{2^l}{\mathbf{E}\left[\sqrt{2 \cdot C_l} + 1\right]} \geq \frac{2^l}{\sqrt{2 \cdot \mathbf{E}[C_l]} + 1} \tag{4.4}$$

$$\geq \frac{2^l}{\sqrt{2} \cdot \frac{2^l}{\sqrt{2^{n-l} \cdot p'}} + 1} = \frac{1}{\sqrt{\frac{2}{2^n \cdot p'}} 2^{l/2} + \frac{1}{2^l}} \tag{4.5}$$

where (4.3) is due to Jensen's inequality (see Theorem A.2.4 in the appendix), (4.4) uses (4.2) and Jensen, and (4.5) follows from (4.1). □

Now, we can apply this lemma to show a lower bound on the expected π-OBDD size of a k-wise independent random function.

Theorem 4.2.4. *Let $f : S \to B_n$ be a k-wise independent random function over a sample space S with $k \geq 4$ and $\mathbf{Pr}[f_s(x) = 1] = p$ for all $x \in \{0,1\}^n$. Then, for a fixed variable order π, the expected π-OBDD size of f_s is bounded below by $\Omega(2^{n/3} \cdot (p')^{(1/3)})$ with $p' = 2p(1-p)$.*

Proof. Recall that D_l is the number of different subfunctions $f_{s|\alpha}$ defined as $D_l := \left|\left\{\left[f_{s|\alpha}\right] \mid \alpha \in \{0,1\}^l\right\}\right|$. We cannot just sum over the expected values of D_l to get a lower bound of the OBDD size because we do not know whether the D_l pairwise different subfunctions resulting from replacing the first l bits by constants are dependent on the next variable. But for a fixed l, each of the D_l subfunctions needs one node in the OBDD representing f_s (but not necessarily labeled by the variable $x_{\pi(l+1)}$). Therefore, we can bound the expected OBDD size from below by every choice of $\mathbf{E}[D_l]$. Thus, we need a lower bound on $\mathbf{E}[D_l]$. Due to Lemma 4.2.3 we have

$$\mathbf{E}[D_l] \geq \frac{1}{\sqrt{\frac{2}{2^n \cdot p'}}2^{l/2} + \frac{1}{2^l}}.$$

For $l = 1$, the denominator is roughly $1/2$. Now, increasing l results in an increase of the first summand and in a decrease of the second summand. Thus, computing the value of l with $\sqrt{\frac{2}{2^n \cdot p'}}2^{l/2} = \frac{1}{2^l}$ should give a large lower bound of $\mathbf{E}[D_l]$:

$$\begin{aligned} & \sqrt{\tfrac{2}{2^n \cdot p'}}2^{l/2} &=& \tfrac{1}{2^l} \\ \Leftrightarrow \quad & 2^{3l/2} &=& \sqrt{2^{n-1} \cdot p'} \\ \Leftrightarrow \quad & l &=& 1/3(n - 1 + \log p') \end{aligned}$$

W.l.o.g. assume that $l^* = 1/3(n - 1 + \log p')$ is an integer. Then the expected π-OBDD size is at least

$$\frac{1}{\sqrt{\frac{2}{2^n \cdot p'}}2^{l^*/2} + \frac{1}{2^{l^*}}} = \frac{1}{\frac{2}{2^{l^*}}} = \Omega(2^{n/3} \cdot (p')^{1/3}).$$

□

The last theorem states that we should not use k-wise independent random functions in an implicit algorithm with a fixed variable order and $k \geq 4$. But it is still possible that for every function f_s there is a variable order π such that the π-OBDD representing f_s is small. But the following theorem shows that representing k-wise independent random functions with $k \geq 4$ is infeasible even for FBDDs. The general strategy of the proof of this is similar to the proof as in Wegener's analysis [127] where the OBDD size of completely independent random functions is analyzed: We bound the probability p_l that there is a set of l variables such that the number of different subfunctions obtained by replacing these variables by constants deviates too much from the expected value. Then we show that there is no such deviation in any set of l variables with nonzero probability. While in [127] the function values are completely independent and, therefore, the calculation can be done more directly and with better estimations, we have to take a detour over the number of subfunctions which are equal

(as in Lemma 4.2.3 and Theorem 4.2.4) and can only use Markov's inequality to calculate the deviation of the expectation. Furthermore, because of the independence, Wegener [127] was able to do a more subtle analysis of the OBDD size by investigating the effects of the OBDD minimization rules separately.

Theorem 4.2.5. *Let $f : S \to B_n$ be a k-wise independent random function over a sample space S with $k \geq 4$ and $\mathbf{Pr}[f_s(x) = 1] = p$ for all $x \in \{0,1\}^n$. Then, there is an $s \in S$ such that f_s is an r-mixed function with $r = \Omega(n + \log(p'))$ and $p' = 2p(1-p)$.*

Proof. We prove that there is a function f_s such that for all subsets of r variables, the 2^r assignments of these variables lead to different subfunctions. We start with a sketch of the proof: First, we fix a set of l variables and prove an upper bound on the probability that the number C_l of collisions $f_{s|\alpha} = f_{s|\alpha'}$ deviates by a factor of δ_l from the expectation μ_l. Then we choose δ_l in such a way that the probability is smaller than 1 that there exists a set of l variables where the number of collisions is greater than $\delta_l \cdot \mu_l$. Now, we can condition on the event that $C_l \leq \delta_l \cdot \mu_l$ for every choice of l variables: By means of Lemma 4.2.3 we calculate a value of r such that $D_r > 2^r - 1$. Since D_r is an integer for every r, this implies that $D_r = 2^r$. Thus, all 2^r possible subfunctions are different for all choices of r variables which concludes the proof.

For a fixed set of l variables and a variable order whose set of first l variables coincides with these variables, we know from Lemma 4.2.3 that the expected value of C_l is at most $\frac{2^{2l}}{2^{n-l} \cdot p'}$. Due to the dependencies, using Markov's inequality is the best we can do to bound the deviation from the expectation. Thus, for $\delta_l > 1$ we have

$$\mathbf{Pr}[C_l \geq \delta_l \cdot \mathbf{E}[C_l]] \leq \frac{1}{\delta_l}.$$

We have to distinguish $\binom{n}{l}$ possibilities to choose the l variables (and the corresponding variable orders). Let $\delta_l := 2 \cdot \binom{n}{l}$. Then the probability that for all choices of l variables C_l is less than $\delta_l \cdot \mathbf{E}[C_l]$ is bounded below by $1/2$. Now, we condition on the event that $C_l < \delta_l \cdot \mathbf{E}[C_l]$ for all sets of l variables. From Lemma 4.2.3 we know that $D_l \geq \frac{2^l}{\sqrt{2C_l}+1}$ which is strictly greater than $\frac{2^l}{\sqrt{2\delta_l \cdot \mathbf{E}[C_l]}+1} \geq \frac{2^l}{\sqrt{2\delta_l \cdot \frac{2^{2l}}{2^n \cdot p'}}+1}$. Recall that we want to show that

$D_l > 2^l - 1$. Thus, it is sufficient to show that

$$\begin{aligned}
& \frac{2^l}{\sqrt{2\delta_l \cdot \frac{2^{2l}}{2^n \cdot p'}} + 1} && \geq && 2^l - 1 \\
\Leftrightarrow \quad & 1 && \geq && (1 - \tfrac{1}{2^l}) \cdot (\sqrt{\tfrac{2\delta_l}{2^n \cdot p'}} \cdot 2^l + 1) \\
\Leftrightarrow \quad & 1 && \geq && \sqrt{\tfrac{2\delta_l}{2^n \cdot p'}} \cdot 2^l + 1 - \sqrt{\tfrac{2\delta_l}{2^n \cdot p'}} - \tfrac{1}{2^l} \\
\Leftarrow \quad & \tfrac{1}{2^l} && \geq && \sqrt{\tfrac{2\delta_l}{2^n \cdot p'}} \cdot 2^l \\
\Leftrightarrow \quad & 1 && \geq && \sqrt{\tfrac{2\delta_l}{2^n \cdot p'}} \cdot 2^{2l} \\
\Leftrightarrow \quad & \sqrt{\tfrac{2^n \cdot p'}{2\delta_l}} && \geq && 2^{2l} \\
\Leftrightarrow \quad & (1/2)(n + \log(p') - 1 - \log(\delta_l)) && \geq && 2l \\
\Leftrightarrow \quad & (1/4)(n + \log(p') - 1 - \log(\delta_l)) && \geq && l.
\end{aligned}$$

Notice that step 3 is not an equivalent transformation. Next, we need a known bound for the binomial coefficient (see Lemma A.1.1 in the appendix):

$$\log \binom{n}{l} \leq n \cdot H(l/n)$$

where $H(x) = -x \log(x) - (1-x)\log(1-x)$ is the binary entropy function. Let $l = \varepsilon \cdot n$ for some $\varepsilon < 1/2$. We want to maximize ε with

$$\begin{aligned}
& \tfrac{1}{4}(n + \log(p') - 2 - H(\varepsilon) \cdot n) && \geq && \varepsilon \cdot n \\
\Leftrightarrow \quad & \tfrac{1}{4}(1 + \tfrac{\log(p') - 2}{n}) && \geq && \varepsilon + (1/4)H(\varepsilon)
\end{aligned}$$

Using $\dfrac{1}{1-x} \leq 1 + 2x$ for $0 \leq x \leq 1/2$ and $\log(1+x) \leq x$ for $x \leq 1$, we have

$$\begin{aligned}
\varepsilon + \frac{H(\varepsilon)}{4} &= \varepsilon + \frac{1}{4}(\varepsilon \log(1/\varepsilon) + (1-\varepsilon) \cdot \log(1/(1-\varepsilon))) \\
&\leq \varepsilon + \frac{1}{4}(\varepsilon\sqrt{1/\varepsilon} + \log(1 + 2\varepsilon)) \\
&\leq \varepsilon + \frac{1}{4}(\sqrt{\varepsilon} + 2\varepsilon) \\
&\leq \frac{7}{4}\sqrt{\varepsilon}.
\end{aligned}$$

Thus, if

$$\begin{aligned}
\sqrt{\varepsilon} &\leq \frac{1}{7} + \frac{1}{7} \cdot \left(\frac{\log(p') - 2}{n}\right) \\
\Leftrightarrow \quad \varepsilon &\leq \left(\frac{1}{7} + \frac{1}{7} \cdot \left(\frac{\log(p') - 2}{n}\right)\right)^2,
\end{aligned}$$

then it is $\frac{1}{4}(n + \log(p') - 1 - H(\varepsilon) \cdot n) \geq \varepsilon \cdot n$ which implies that $\frac{2^l}{\sqrt{2\delta_l \cdot \frac{2^{2l}}{2^n \cdot p'}} + 1} \geq 2^l - 1$ with

$l = \varepsilon \cdot n$. Since $D_l > \frac{2^l}{\sqrt{2\delta_l \cdot \frac{2^{2l}}{2^n \cdot p'}+1}}$ this also implies that $D_l > 2^l - 1$. Since $a^2 - 2ab \leq (a-b)^2$, we can choose $\varepsilon \in \Omega\left(1 - \frac{\log(1/p')}{n}\right)$. Thus, there is an r-mixed function f_s with $r = \varepsilon \cdot n = \Omega\left(n - \log(1/p')\right)$. □

Due to Lemma 2.3.7, the last theorem gives us a lower bound of $2^{\Omega(n+\log(p'))}$ for the worst-case size of an FBDD representing a k-wise independent random function with $k \geq 4$.

Corollary 4.2.6. *Let $f_s : S \to B_n$ be a k-wise independent random function over a sample space S with $k \geq 4$ and $\mathbf{Pr}[f_s(x) = 1] = p$ for all $x \in \{0,1\}^n$. Then, there is an $s \in S$ such that the FBDD size of f_s is at least $2^{\Omega(n+\log(p'))}$.*

4.3. Construction of Almost k-wise Independent Random Functions

The gap between the OBDD size of 3-wise independent random functions and 4-wise independent random functions is exponentially large. In order to see what kind of random functions have an OBDD size which is in-between these bounds, we show that a construction of a random OBDD of size $\mathcal{O}((nk)^2/\varepsilon)$ generates (ε, k)-wise independent functions. The idea is to use a top-down construction of a random OBDD with fixed complete width w where the successors of each node are chosen uniformly at random within the set of nodes in the next level. If the width w is large enough, the function values of k different inputs are almost uniformly distributed because the paths of the k inputs in the OBDD are likely to be almost independent.

Algorithm 7 Random OBDD Construction

Input: $n \in \mathbb{N}$, $2 \leq k \leq 2^n$, $\varepsilon > 0$, and variable order π

```
w = nk(k − 1)/(2ε)
For every i ∈ {0, ..., n − 1} let L_i be a set of w nodes labeled by x_{π(i)}.
Let L_n be the set containing the 0- and 1-sink
for i = 0 to n − 1 do
    for Every node v ∈ L_i do
        Choose 0- and 1-successor of v independently and uniformly at random from L_{i+1}.
    end for
end for
Return Arbitrary node from L_0 as root of the OBDD
```

Theorem 4.3.1. *Let $n \in \mathbb{N}$, $2 \leq k \leq 2^n$, and $\varepsilon > 0$. Let $S = \{0,1\}^r$ with $r = \mathcal{O}\left(\frac{(nk)^2}{\varepsilon} \cdot \log \frac{nk}{\varepsilon}\right)$ be the set of random bits used by Algorithm 7. Then the random function $f : S \to B_n$ generated by Algorithm 7 is (ε, k)-wise independent and the OBDD size of f_s is $\mathcal{O}(\frac{n^2k^2}{\varepsilon})$ for every $s \in S$.*

Proof. First, we observe that the number of random bits needed by Algorithm 7 is $\mathcal{O}(w \cdot \log w)$ for each level and, therefore, $\mathcal{O}(nw \log w) = \mathcal{O}\left(\frac{(nk)^2}{\varepsilon} \cdot \log \frac{nk}{\varepsilon}\right)$ in total. Recall that we have to show that for every choice of k different inputs, the joint distribution is almost uniform. Let $a_1, \ldots, a_k \in \{0,1\}^n$ be k different inputs and p be the probability that the function values of f_s on these inputs are $b_1, \ldots, b_k \in \{0,1\}$. For now, we assume that each pair of the inputs $a_1, \ldots, a_k$ differ in at least one bit within the first $n-1$ bits. Let $P_1, \ldots, P_k$ be the paths of $a_1, \ldots, a_k$ to the layer L_{n-1}, i. e., the paths end in a node labeled by x_{n-1}. Let D_i be the event that for every pair of paths P_{i_1}, P_{i_2} with $1 \leq i_1 < i_2 \leq i$ the paths coincide within the first ℓ nodes for some $1 \leq \ell \leq n-1$ and they are disjoint in the last $n-\ell$ nodes. This also means that they are ending in different nodes in layer L_{n-1}. Clearly, it is $\mathbf{Pr}[D_1] = 1$. Let $\ell \in [n-1]$ be maximal such that there is an input from $\{a_1, \ldots, a_i\}$ which coincides with a_{i+1} in the first ℓ bits. Conditioning on the event D_i, the $(\ell+1)$-th edge of the path P_{i+1} is not used by any other path $P_1, \ldots, P_i$ since otherwise ℓ was not maximal. Therefore, it holds $\mathbf{Pr}[D_i \mid D_{i-1}] \geq \left(1 - \frac{i-1}{w}\right)^n$ and with it

$$\mathbf{Pr}[D_k] = \prod_{i=2}^{k} \mathbf{Pr}[D_i \mid D_{i-1}] \geq \prod_{i=2}^{k} \left(1 - \frac{i-1}{w}\right)^n.$$

We have

$$\begin{aligned} \prod_{i=2}^{k} \left(1 - \frac{i-1}{w}\right)^n &= \prod_{i=2}^{k} \left(1 - \frac{1}{w/(i-1)}\right)^{(w/(i-1)) \cdot (i-1)n/w} \\ &\geq e^{-\frac{n}{w}\sum_{i=1}^{k-1} i} = e^{-\frac{nk(k-1)}{2w}} = e^{-\varepsilon} \geq 1 - \varepsilon \end{aligned}$$

for $w = nk(k-1)/(2\varepsilon)$. If all paths end in different nodes, then the function values of the k inputs are independent and uniformly distributed, i. e.,

$$p \geq \mathbf{Pr}[f_s(a_1) = b_1, \ldots, f_s(a_k) = b_k \mid D_k] \cdot \mathbf{Pr}[D_k] = 2^{-k} \cdot \mathbf{Pr}[D_k] \geq 2^{-k} - \varepsilon$$

and

$$\begin{aligned} p &\leq 1 - \mathbf{Pr}[\exists j : f_s(a_j) \neq b_j] \leq 1 - \mathbf{Pr}[\exists j : f_s(a_j) \neq b_j \mid D_k] \cdot \mathbf{Pr}[D_k] \\ &= 1 - (1 - 2^{-k}) \cdot \mathbf{Pr}[D_k] \leq 1 - 1 + 2^{-k} - \varepsilon + \varepsilon \cdot 2^{-k} \leq 2^{-k} + \varepsilon. \end{aligned}$$

Now, let $a_1, \ldots, a_k$ be arbitrary inputs. We group the inputs into $k' \leq k$ sets $A_1, \ldots, A_{k'}$ where each set has at most two elements and if there are two inputs in a set then they are identical in the first $n-1$ bits. Applying the same analysis as before for the paths $P_1, \ldots, P_{k'}$ of the inputs in $A_1, \ldots, A_{k'}$ gives a probability of at least $1-\varepsilon$ that they end in different nodes in layer L_{n-1}. Now, the paths of the inputs are either ending in different nodes in layer L_{n-1} or ending in the same node but the inputs differ in the last input bit. Note that there are

at most two paths ending in the same node. Either way, the function values are independent and uniformly distributed which completes the proof. □

4.4. Randomized OBDD-Based Algorithms

We now turn towards the algorithmic part of this chapter. In the first subsection we present the extension of Sawitzki's structural result [116, 118] to the randomized setting. Then we elaborate the strengths and weaknesses of randomized functions in OBDD-based algorithms. Based on these considerations, we design randomized OBDD-based algorithms for the minimum spanning tree and the maximal matching problem.

Extension of Sawitzki's Structural Result to Randomized Algorithms

Only a small modification is necessary to extend Sawitzki's simulation results from [116] and [118] to show that the set of problems which can be solved by a randomized implicit algorithm is equal to the set of problems solved by a randomized parallel algorithm. We do not give a rigorous proof here because the differences are negligible compared to the complexity of the original proof. Since we are focusing more on the practical part of OBDD-based algorithms, we would not benefit of repeating the whole proof. Instead we sketch the main ideas including our changes to extend them to randomized algorithms.

In [116], Sawitzki proposed a model for implicit algorithms called Symbolic Random Access Machine (SRAM) that bundles registers representing Boolean functions together with a set of functional operations which implicit algorithms typically are based on, e. g., the evaluation of a function on a specific input, binary synthesis, and argument reordering. He showed that an SRAM can be efficiently simulated by a parallel random access machine (PRAM) which is the standard model for parallel computation (see, e. g., [54]). SRAMs are independent of the underlying data structure for the Boolean functions. The simulation result was used to show lower bounds on the number of functional operations by using results from the P-completeness theory which says that so-called P-complete problems cannot be solved by PRAMs in polylog N time and with polynomially many processors unless P = NC. Using the simulation result, this also means that P-complete problems cannot be computed by SRAMs (including OBDD-based algorithms) with polylog N functional operations and $\mathcal{O}(\log N)$ variables. For the case of randomization, we add one symbolic operation to the SRAM model: assign a random function $r : \{0,1\}^l \to \{0,1\}$ with $l = \mathcal{O}(\log N)$ to a symbolic register. Note that the function values of r are completely independent. Constructing such functions with a randomized PRAM is easy: Since the function table of r has only polynomial size, we can use 2^l processors that draw a random bit and write all function values in parallel. The other way around, i. e., simulating a randomized parallel algorithm by a randomized OBDD-based algorithm, is also simple: It is well known that (randomized) PRAMs can be simulated

by circuits. The only difference between randomized PRAMs and deterministic PRAMs is the set of input variables of the circuit. A deterministic circuit has only N input variables whereas the random circuit has additional $\mathcal{O}(N^c)$ random inputs for a constant c. Since we can construct a random function $r : \{0,1\}^l \to \{0,1\}$ with $l = O(\log N)$, we can set the input variables correctly for the simulation in the same way as in [118].

What are Random Functions Good for in OBDD-Based Algorithms?

Let us start with the weaknesses of our model: The first and foremost obstacle is the constraint to use 3-wise independent or (ε, k)-wise independent random functions to have efficient OBDD constructions of the random functions. The problem lies in the analysis of the algorithms because limited independence can make a proof of good upper bounds on the number of functional operations challenging. One ought to think that (ε, k)-wise independence is statistically so close to k-wise independence that we can analyze an algorithm using k-wise independence and then replace the random bits by (ε, k)-wise independent random bits with a small additional error. Unfortunately, Alon, Goldreich, and Mansour [5] showed that (ε, k)-wise independent random variables have a large statistical difference to k-wise independence random variables. In particular, they showed a tight $2^{\Theta(nk)} \cdot \varepsilon$ bound on the statistical difference which means that we cannot replace k-wise independence by (ε, k)-wise independence obliviously without setting $\varepsilon = 2^{-\Theta(nk)}$. Indeed, in applications of (ε, k)-wise independent random variables, e. g., in [104], the ε has to be chosen in the order of $2^{-n \cdot f(k)}$ for some function $f(k)$. This only works whenever the construction has a $\log(1/\varepsilon)$ dependency on the closeness parameter ε but not in our OBDD construction from Section 4.3.

Nevertheless, if we have a randomized OBDD-based algorithm but cannot prove that using 3-wise or (ε, k)-wise independent random functions leads to a small number of functional operations, we can use them as a heuristic approach to have a simple randomized algorithm but with no guarantee on the number of functional operations. Fortunately, there exist some algorithms using pairwise independent random variables, e. g., for the MIS problem [3, 89]. But trying to adapt these algorithms for the implicit setting revealed another weakness of the model: Our random construction misses the possibility to use different probabilities for the inputs (which will be typically binary encoded nodes or edges). E. g., in one algorithm every node v is marked with probability $1/\deg(v)$. We will see how to simulate such a procedure with random function where every function value is 1 with probability $1/2$ and even without computing or knowing the degree of the nodes.

But how does this simulation work? And what are the strengths of the random functions? The answer is sampling subgraphs. In OBDD-based algorithms random functions are meant for deleting edges or nodes with a fixed probability: Let $f : S \to B_{2n}$ be a random function over a sample space S. The inputs of $f_s(x, y)$ are two binary vectors of length n representing nodes in a graph $\chi_E(x, y)$. Assume for the moment that $f_s(x, y) = f_s(y, x)$, then $\chi_E(x, y) \wedge$

Algorithm 8 $HeavyEdges(F(x,y), \chi_E(x,y,d))$: Compute the Set of Heavy Edges of a Graph

Input: Spanning forest $F(x,y)$, weighted graph $\chi_E(x,y,d)$
Output: Set of heavy edges $H(x,y)$ with respect to $F(x,y)$

$\triangleright$ Compute function $P(x,y,s,t)$ that evaluates to 1 iff the edge (x,y)
$\triangleright$ is on the (unique) path in F from s to t
$P(x,y,s,t) = F(x,y) \wedge (s = x) \wedge (t = y)$
repeat
  $P'(x,y,s,t) = P(x,y,s,t)$
  $Reach(s,t) = \exists x,y : P(x,y,s,t)$
  $P(x,y,s,t) = P(x,y,s,t) \vee \exists z : Reach(s,z) \wedge P(x,y,z,t)$
until $P'(x,y,s,t) = P(x,y,s,t)$
$\triangleright$ Compute set of heavy edges
$H(x,y) = \exists d : \chi_E(x,y,d) \wedge \overline{\exists d', x', y' : (d \leq d') \wedge \chi_E(x',y',d') \wedge P(x',y',x,y)}$
return $H(x,y)$

$f_s(x,y)$ is the random subgraph where every edge is deleted with probability $\mathbf{Pr}[f_s(x,y) = 0]$. Another example is to sample nodes with probability p and look at the induced subgraph of these nodes. For this, let $f : S \to B_n$ be a random function with $\mathbf{Pr}[f_s(x) = 1] = p$ and we compute $\chi_E(x,y) \wedge f_s(x) \wedge f_s(y)$. These will be our main tools to design randomized OBDD-based algorithms.

We focus on randomized algorithms with zero error, i. e., we want to design Las Vegas algorithms. A problem with one- or two-sided error could be that we cannot apply probability amplification by repeating the algorithm poly N times since we are dealing with large input graphs. For instance, the parallel mincut algorithm by Luby, Naor, and Naor [90], which uses only pairwise independence and randomly chooses edges with a fixed probability, has a failure probability of $1/\operatorname{poly} N$. While increasing the number of parallel execution is possible, it is not clear how to do this in the implicit setting without sequential repetition.

Nevertheless, we have to keep these limitations in mind and it is not surprising that a restricted computational model is not capable of everything we know from explicit algorithms without increasing memory consumption or running time. But we will see that randomization in OBDD-based algorithms is still a useful mechanism and can lead to simple and fast algorithms outperforming deterministic ones.

Randomized Minimum Spanning Tree Algorithm

Our first randomized OBDD-based algorithm computes a minimum spanning tree and is based on the known randomized MST algorithm by Karger, Klein, and Tarjan [76] with expected linear running time. Let $G = (V,E)$ be a weighted graph and denote the weight of an edge $\{u,v\}$ by $w(u,v)$. The algorithm combines so-called Borůvka steps with random sampling. Here, we describe the Borůvka step used by Bollig [15] in the deterministic OBDD-based minimum spanning tree algorithm: Start with an empty set F_1 of edges. With respect to

F_1, at the beginning each node is a connected component. For every connected component, select an edge with minimum weight that is incident to a node from a different component. Add these edges to F_1 and recompute the connected components with respect to the new set of edges. Iterating the Borůvka step until one component remains results in a minimum spanning tree F_1. A single Borůvka step reduces the number of connected components by at least a factor of two. Edges chosen in a Borůvka step are definitely part of a minimum spanning tree.

The random sampling procedure tries to eliminate edges which cannot be in a minimum spanning tree. For this, we need the notion of F-heavy (F-light) edges with respect to a forest F. An edge $\{u, v\} \in E$ is F-heavy if $w(u, v)$ is greater than every edge weight on the unique path from u to v in F, and F-light otherwise. It is easy to see that an F-heavy edge cannot be part of any minimum spanning tree which means we can compute the set of F-heavy edges for an intermediate forest F and discard these edges to reduce the density of the graph. Now, the algorithm of Karger, Klein, and Tarjan works as follows:

1. Apply two Borůvka steps and let F_1 be the set of selected edges and G_1 be the subgraph that consists of the edges connecting two different components.

2. Let G_2 be a random subgraph of G_1 obtained by choosing each edge with probability 1/2. Apply the algorithm recursively on G_2 which returns a minimum spanning forest F_2 for G_2.

3. Compute the set of F_2-heavy edges and delete them in G_1. Call the remaining graph G_3.

4. Apply the algorithm recursively on G_3 which returns a minimum spanning forest F_3.

5. Return the union of F_1 and F_3.

In order to implement this algorithm as an randomized OBDD-based algorithm, we have to think about how to compute the set of heavy edges and how to randomly sample a subgraph. The latter is the easier problem: We construct two 3-wise independent random functions $f_{r_1}(x), f_{r_2}(y)$ using Algorithm 6 and set

$$Filter(x, y) = (|x|_2 < |y|_2) \wedge (f_{r_1}(x) \oplus f_{r_2}(y)).$$

Because we are dealing with undirected graphs, we want $Filter(x, y) = Filter(y, x)$ for every (x, y). Therefore, we set $Filter(x, y) = Filter(x, y) \vee Filter(y, x)$. Since

$$\mathbf{Pr}\left[f_{r_1}(x) \oplus f_{r_2}(y) = 1\right] = \mathbf{Pr}\left[f_{r_1}(x) \neq f_{r_2}(y)\right] = 1/4 + 1/4 = 1/2,$$

the operation $\chi_E(x, y) = \chi_E(x, y) \wedge Filter(x, y)$ computes a random subgraph as required.

Algorithm 9 $RandomMST(\chi_E(x,y,d), R(x,y))$: Randomized implicit MST algorithm

Input: Weighted graph $\chi_E(x,y,d)$, merged nodes $R(x,y)$
Output: Minimum spanning tree $MST(x,y)$

▷ Edges selected in the Borůvka steps
$F_1(x,y) = 0$
▷ Step 1: Two Borůvka steps (Bollig [15])
for $i = 0$ to 1 **do**
▷ Choose smallest edges (see Appendix B.2 for the definition of Π_1)

$$C(x,y) = \frac{\exists d : \chi_E(x,y,d) \wedge \overline{R(x,y)} \wedge}{\exists y', z, d' : R(x,z) \wedge \chi_E(z,y',d') \wedge \overline{R(z,y')} \wedge \Pi_1((z,y',d'),(x,y,d))}$$

$C(x,y) = C(x,y) \vee C(y,x)$
$F_1(x,y) = F_1(x,y) \vee C(x,y)$
▷ Compute connected components
$R(x,y) = findTransitiveClosure(R(x,y))$
end for
▷ Step 2: Recursive step on random subgraph
▷ Construct 3-wise independent random functions (see Algorithm 6)
$f_{r_1}(x) = RandomFunc(x,n)$
$f_{r_2}(y) = RandomFunc(y,n)$
$Filter(x,y) = (|x|_2 < |y|_2) \wedge (f_{r_1}(x) \oplus f_{r_2}(y))$
$Filter(x,y) = Filter(x,y) \vee Filter(y,x)$
$\chi_{E'}(x,y,d) = \chi_E(x,y,d) \wedge Filter(x,y)$
$F_2(x,y) = RandomMST(\chi_{E'}(x,y,d), R(x,y))$
▷ Compute F_2-heavy edges (see Algorithm 8)
$H(x,y) = HeavyEdges(F_2(x,y), \chi_E(x,y,d))$
$\chi_E(x,y,d) = \chi_E(x,y,d) \wedge \overline{(H(x,y))}$
▷ Step 4: Recursion
$F_3(x,y) = RandomMST(\chi_E(x,y,d), R(x,y))$
▷ Step 5: Return MST
return $F_1(x,y) \vee F_3(x,y)$

For the F-heavy edges, assume that we have a function $P(x,y,s,t)$ which evaluates to 1 if and only if the edge $\{x,y\}$ is on the (unique) path from s to t in the forest F. Then an edge $\{s,t\}$ is F-heavy if there is no edge $\{x,y\}$ with less or equal weight and $P(x,y,s,t) = 1$. Thus, the function

$$H(x,y) = \exists d : \chi_E(x,y,d) \wedge \overline{\exists d', x', y' : (d \leq d') \wedge \chi_E(x',y',d') \wedge P(x',y',x,y)}$$

represents the set of F-heavy edges in the graph. Using an iterative squaring approach, the function $P(x,y,s,t)$ can be easily computed with $\mathcal{O}(\log^2 N)$ functional operations. Algorithms 9 and 8 show the implicit implementation of our ideas. Since we are using 3-wise independent random functions, we cannot apply the same analysis as in [76]. An upper bound on the number of functional operations is currently not known. We experimentally evaluate

the algorithm by a comparison to the deterministic algorithm by Bollig [15] and implicit variants of the well-known algorithms by Kruskal and Prim in Chapter 5.

Randomized Maximal Matching Algorithm

As we mentioned in the discussion about strengths and weaknesses of random functions in OBDD-based algorithms, a drawback of our model is the missing possibility to use different probabilities for different nodes or edges. Thus, for designing a randomized maximal matching algorithm we cannot directly use known algorithm using pairwise independence like in [3] or [89] where nodes are marked with a probability proportional to the node degree. In order to simulate these selections by our construction, we use an iterative random sampling approach: We delete each edge with probability 1/2, store all isolated edges, and repeat this as long as the graph is not empty. Finally, we add the stored isolated edges to the matching, remove all edges incident to matched nodes, and repeat the procedure until there are no edges left. Algorithm 10 shows the whole randomized maximal matching algorithm.

Algorithm 10 Randomized maximal matching algorithm

Input: Graph $G = (V, E)$
Output: Maximal matching $M \subseteq E$

```
M = ∅
while E ≠ ∅ do
    E' = E
    M' := ∅
    while E' ≠ ∅ do
        Delete each edge in E' with probability 1/2 (using 3-wise independent r.v.)
        Add all isolated edges in E' to M'
    end while
    M = M ∪ M'
    E = E \ {(u, v) | u or v is matched by M}
end while
Return M
```

We say that an edge $e \in E$ (before the inner while-loop) survives iff $e \in M'$ after the inner while-loop of Algorithm 10.

Lemma 4.4.1. *For every $e = \{u, v\} \in E$ with $\deg_E(u) > 1$ or $\deg_E(v) > 1$ before the inner while-loop in Algorithm 10, the probability that e survives is at least $\frac{1}{8 \cdot (\deg_E(u) + \deg_E(v) - 2)}$. If $\deg_E(u) = \deg_E(v) = 1$ then e survives with probability 1/2.*

Proof. Let $e = \{u, v\} \in E$ be an edge before the inner while-loop and let R_e be the number of rounds until edge e is deleted. If $\deg_E(u) = \deg_E(v) = 1$ then e survives if and only if $R_e > 1$ which occurs with probability 1/2. The random bits in each iteration are 3-wise independent and the iterations themselves are completely independent. Thus, the variables R_e are also

3-wise independent. Denote the neighborhood of e by $N(e) = \{e' \in E \mid e \cap e' \neq \emptyset\}$, i. e., $N(e)$ contains all edges incident to u or v. Then we have

$$\mathbf{Pr}\left[e \text{ survives}\right] = \mathbf{Pr}\left[R_e \text{ is unique maximum in } \{R_{e'} \mid e' \in N(e)\}\right].$$

It is easy to see that $\mathbf{Pr}\left[R_e = i\right] = \left(\frac{1}{2}\right)^i$ for $i \geq 1$. Let $e' \in N(e)$ with $e' \neq e$ and let $z \geq 1$ be fixed. Since the R_e are 3-wise independent, we have

$$\mathbf{Pr}\left[R_{e'} \geq z \mid R_e = z\right] \quad = \quad \mathbf{Pr}\left[R_{e'} \geq z\right] = \sum_{i=z}^{\infty} \left(\frac{1}{2}\right)^i = \left(\frac{1}{2}\right)^{z-1}.$$

Therefore, the probability that there is an edge $e' \in N(e) \setminus e$ with $R_{e'} \geq z$ is at most $\frac{|N(e)|-1}{2^{z-1}}$. Thus, R_e is unique maximum with probability at least $1 - \frac{|N(e)|-1}{2^{z-1}}$ if $R_e = z$. This probability is greater than 0 for $z \geq \lceil \log(|N(e)|-1) \rceil + 2$. Finally, we have

$$\begin{aligned}
\mathbf{Pr}\left[R_e \text{ is unique maximum}\right] &\geq \sum_{z=\lceil \log(|N(e)|-1)\rceil+2}^{\infty} \left(\frac{1}{2}\right)^z \cdot \left(1 - \frac{|N(e)|-1}{2^{z-1}}\right) \\
&\geq \left(\frac{1}{2}\right)^{\log(|N(e)|-1)+2} \cdot \left(1 - \frac{|N(e)|-1}{2^{\log(|N(e)|-1)+1}}\right) \\
&= \frac{1}{4 \cdot (|N(e)|-1)} \cdot \frac{1}{2} \\
&= \frac{1}{8 \cdot (\deg_E(u) + \deg_E(v) - 2)}.
\end{aligned}$$

□

Algorithm 11 shows the implicit version of Algorithm 10. The random sampling is realized as in our minimum spanning tree algorithm, see Algorithm 9.

The number of deleted edges for a matching edge $\{u, v\}$ that is added to the matching is $\deg(u) + \deg(v) - 2$ if we do not count the matching edge itself. Thus, the expected number of deleted edges is $\Omega(|E|)$ at the end of the outer loop. This gives us the following result.

Theorem 4.4.2. *Let $G = (V, E)$ be a graph with N nodes. Algorithm 11 computes a maximal matching. All functions used in Algorithm 11 depend on at most $3 \log N$ variables and the expected number of operations is $\mathcal{O}(\log^3 N)$.*

Proof. Each iteration of the inner-loop needs $\mathcal{O}(\log N)$ operations. Since we halve the number of edges in expectation in each iteration of this loop, the expected number of iterations is $\mathcal{O}(\log N)$. In expectation, the number of edges $e = \{u, v\}$ with $\deg_E(u) = \deg_E(v) = 1$ is halved in each iteration of the outer loop since the survival probability is $1/2$ of each edge due to Lemma 4.4.1. Thus, we can assume that all edges in E have either $\deg_E(u) > 1$ or $\deg_E(v) > 1$. Using Lemma 4.4.1, we can show that the expected number of edges (excluding

Algorithm 11 Randomized Implicit Maximal Matching Algorithm

Input: Graph $\chi_E(x, y)$
Output: Maximal matching $\chi_M(x, y)$

```
▷ Initial matching
χ_M(x, y) = 0
while χ_E(x, y) ≢ 0 do
    χ_E'(x, y) = χ_E(x, y)
    NewEdges(x, y) = 0
    while χ_E'(x, y) ≢ 0 do
        ▷ Construct 3-wise independent random functions (see Algorithm 6)
        f_r1(x) = RandomFunc(x, n)
        f_r2(y) = RandomFunc(y, n)
        Filter(x, y) = (x > y) ∧ (f_r1(x) ⊕ f_r2(y))
        Filter(x, y) = Filter(x, y) ∨ Filter(y, x)
        ▷ Delete edges with probability 1/2
        χ_E'(x, y) = χ_E'(x, y) ∧ Filter(x, y)
        ▷ Update T(x, y)
        T(x) = ∃z, y : (z ≠ y) ∧ χ_E'(x, y) ∧ χ_E'(x, z)
        ▷ Store isolated edges
        NewEdges(x, y) = NewEdges(x, y) ∨ (χ_E'(x, y) ∧ ¬T(x) ∧ ¬T(y))
    end while
    ▷ Add edges to current matching
    χ_M(x, y) = χ_M(x, y) ∨ NewEdges(x, y)
    Matched(x) = ∃y : χ_M(x, y)
    ▷ Delete edges incident to matched nodes
    χ_E(x, y) = χ_E(x, y) ∧ ¬Matched(x) ∧ ¬Matched(y)
end while
return χ_M(x, y)
```

the matching edges) deleted in each iteration of the outer-loop is at least

$$\sum_{e=\{u,v\}\in E} \frac{|N(e)|-1}{8\cdot(\deg_E(u)+\deg_E(v)-2)} = \sum_{e=\{u,v\}\in E} \cdot\frac{\deg_E(u)+\deg_E(v)-2}{8\cdot(\deg_E(u)+\deg_E(v)-2)} = |E|/8.$$

Thus, the expected number of iterations of the outer-loop is also bounded above by $\mathcal{O}(\log N)$.

Since we add only isolated edges to the matching, M contains a matching at the end of the algorithm. The matching is also maximal because we delete all edges incident to matched nodes and the graph is empty in the end. □

Application to the Maximal Independent Set Problem

In the distributed model which is for instance considered in [99] each node represents a processor in a network and each edge represents a bidirectional communication channel. The time complexity is measured in the number of rounds until each node completes its computation

where one round consists of sending messages then receiving messages from neighbored nodes and some local computation. The bit complexity is measured in the maximum number of bits sent over a single channel in all rounds together.

With a similar idea as for the matching algorithm we are able to design a distributed MIS algorithm: Each node v draws a random bit until this bit is 0. Let R_v be the number of bits drawn by node v. We send R_v to all neighbors and include node v to the independent set if and only if R_v is a local maximum. The expected number of bits for each channel is 1. A similar analysis as before show that we have an maximal independent set after $\mathcal{O}(\log N)$ steps in expectation and the overall expected number of bits per channel is $\mathcal{O}(\log N)$.

5. Experimental Evaluations

In this chapter, we present the results of the experimental evaluations on implicit graph algorithms. We use the developed algorithms from previous chapters and compare them to known explicit or implicit algorithms for the same problem. This chapter is meant to motivate and underline our goal to simplify implicit algorithms by using randomization or, for the special case of interval graphs, by utilizing the special structure of the nodes' labels. Thus, the experiments can be divided into two categories: (1) Show the limitations and problems of known implicit algorithms. (2) Compare our new algorithms to known algorithms (if possible) and try to show the advantages/disadvantages of them. Particularly, the conducted experiments address the following questions:

1. *How are the maximum matching algorithms by Bollig, Gillé, and Pröger [18] performing on a chosen set of bipartite graphs in comparison to the explicit algorithm by Hopcroft and Karp [61]?*
   Section 5.2 is about the performance of the implicit maximum matching algorithm compared to the algorithm by Hopcroft and Karp. We choose three input instances: very structured graphs, partly randomized but sufficiently structured graphs, instances from a real-world application.

2. *Does the improvement with respect to the number of operations of the matching algorithm on unit intervals by using some properties of the node labels also carry over to the actual performance?*
   In Section 5.3, we evaluate the unit interval matching algorithm from Section 3.3. The input graphs are randomly constructed by a random generation algorithm by Saitoh et al. [113]. We focus here only on the matching algorithm since, unfortunately, the coloring algorithm does not perform well on both unit and general interval graphs. The poor performance of the coloring algorithm is very likely due to the fact that the implicit algorithm is simulating the sequential coloring algorithm. Nevertheless, it uses some nice ideas to accomplish this which differ from the parallel implementation ideas.

3. *Can randomization lead to improved implicit algorithms?*
   In Section 5.4, we compare the randomized implicit maximal matching algorithm from Section 4.4 with known deterministic algorithms. In addition, we use the randomized construction of OBDDs from Section 4.3 to generate graphs with a fixed density and

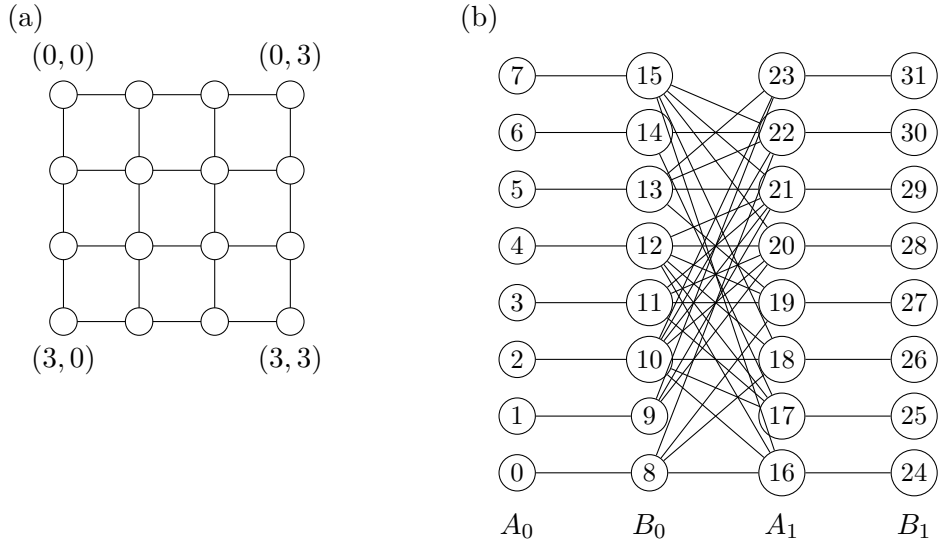


Figure 5.1.: Example of a labeled grid graph with 4×4 nodes and a labeled graph with 32 nodes and block size $d = 8$ from the class `rope`.

investigate whether and how the density influences the running time of both randomized and deterministic maximal matching algorithms. In Section 5.5, we compare the performance of the deterministic implicit minimum spanning tree algorithm by Bollig [15] with the randomized implicit algorithm from Section 4.4.

The input graphs that are used in this chapter are described in Section 5.1. The specific hardware and software setup of the experiments is explained in the beginning of each section. All source files, scripts and random seeds are publicly available[1].

5.1. Input Graphs

Very Structured Graphs: Grid graphs

A graph $G = (V, E)$ on $|V| = N^2$ nodes with integer $N \geq 2$ and $V = \{0, \ldots, N-1\} \times \{0, \ldots, N-1\}$ and edge set

$$E = \{((x, y), (x', y')) \mid \begin{array}{l} (x = x' \text{ and } |y - y'| = 1) \text{ or} \\ (y = y' \text{ and } |x - x'| = 1)\} \end{array}$$

is called undirected *grid* graph (see Fig. 5.1 (a)). We assume that $N = 2^n$ for some $n \in \mathbb{N}$. Therefore, we can implicitly represent a grid graph with $N^2 = 2^{2n}$ nodes for some integer

[1] `http://ls2-www.cs.uni-dortmund.de/~gille/`

$n \geq 1$ by

$$\chi_E((x^r, x^c), (y^r, y^c)) = 1 \Leftrightarrow \begin{array}{l} |x^r|_2 = |y^r|_2 \text{ and } ||x^c|_2 - |y^c|_2| = 1 \text{ or} \\ |x^c|_2 = |y^c|_2 \text{ and } ||x^r|_2 - |y^r|_2| = 1, \end{array}$$

where (x^r, x^c) and (y^r, y^c) are Boolean encodings of nodes.

We have chosen the variable order $\pi = (x_0^r, y_0^r, x_0^c, y_0^c, \ldots, y_{n/2-1}^c)$ for the OBDD representation, i. e., the interleaved variable order with increasing significance. It is not difficult to see that the π-OBDD size for the implicit representation of the grid graph G is bounded by $\mathcal{O}(\log N)$ [131]. Our motivation for choosing this graph class is to analyze our implicit algorithms on very structured graphs. Furthermore, grid graphs have already been used in the investigation of implicit algorithms for maximum flow in 0-1 networks and for topological sorting [115, 131]. The generated grid graphs have a size of $2^n \times 2^n$ beginning with $n = 4$ and n is incremented by 1 after each round.

Both Random and Structured: Rope graphs

The graph class `rope` has been introduced by Cherkassky et al. [27] and each graph from this class is structured but also contains a completely random part, which is quite different from grid graphs. These bipartite graphs are balanced (like grid graphs), i. e., graphs with $V = A \dot\cup B$, $|A| = |B| = N/2$, and N even. The nodes in A and B are grouped into $t = N/(2d)$ blocks of size d. These blocks are denoted by $A_0, \ldots, A_{t-1}$ for A and $B_0, \ldots, B_{t-1}$ for B respectively. Block A_i is connected to block B_i for $i = 0, \ldots, t-1$ and to block B_{i-1} for $i = 1, \ldots, t-1$. The blocks are alternately connected by a perfect matching and a random bipartite graph with average degree $d-1$ beginning and ending with a perfect matching (see Fig. 5.1 (b)). In our tests we have chosen the parameter $d = 8$. The size of the generated graphs begins with 16 and is doubled after each round. The nodes are labeled consecutively according to the order of the connected blocks beginning with A_0, i.e., the nodes of A_i are labeled with $(2d) \cdot i, \ldots, (2d) \cdot i + d - 1$ and the nodes of B_i are labeled with $(2d) \cdot i + d, \ldots, (2d) \cdot i + 2d - 1$ (see Fig. 5.1). For the OBDD representation of the characteristic function of the edge set we have chosen the interleaved variable order with decreasing significance, i. e., $\pi = (x_{n-1}, y_{n-1}, \ldots, x_0, y_0)$.

Instances from Real-World Applications

In order to empirically study maximum flow algorithms, Negruseri et al. [105] have investigated real-world instances which came up from an advertisement application within Google[2]. The close relationship between matching and maximum flow problems motivates our investigation on these real-world instances. In contrast to the other classes these instances are

[2]Graph data files can be found at `http://www.columbia.edu/~cs2035/bpdata/` (Retrieved: July 2015).

Number	$\|A\|$	$\|B\|$	Edges
0	136	18872	222951
1	137	18872	222951
2	137	18888	222951
3	40	7086	33609
4	41	7086	33609
5	41	7093	33609
6	125	7107	33609
7	86	7117	33609
8	86	7127	33609
9	289	16653	33051
10	290	16846	33051
11	290	16904	33051
12	50	13360	50040
13	51	16264	56577
14	51	21016	56577
15	164	288826	2523313
16	165	288826	2523313
17	165	288858	2523313
18	196	89030	1080027
19	197	89044	1080041
20	197	89044	1080041
21	40	28489	43629
22	41	28944	43629
23	41	28944	43629
24	35	11361	22279
25	36	11588	22279
26	36	11695	22279
27	934	8752	42711
28	935	8896	42711
29	935	9028	42711
30	125	55058	844598
31	126	56858	844598
32	126	57926	477356

Table 5.1.: Properties of the real-world instances from [105].

unbalanced, i.e., for all graphs with node set $V = A \dot\cup B$ it is $|A| \gg |B|$. The dataset consists of 33 graphs whose properties are listed in Table 5.1. In our experiments the labeling of the nodes has been adopted from the data files. As for the class `rope`, for the OBDD representation we have chosen the variable order, where the variables are ordered with decreasing significance.

Graph Database: University of Florida Sparse Matrix Collection

The University of Florida Sparse Matrix Collection [39] is a set of sparse matrices from real applications. Table 5.2 lists the largest matrices from this collection used in the experiments ordered by name and gives the size and a short description of the graph. More instances that are used for the minimum spanning tree algorithms can be found in Appendix B.2. The matrices can be downloaded from the website[3] and are represented by a list of edges. For the

[3] `http://www.cise.ufl.edu/research/sparse/matrices/` (Retrieved: July 2015)

Instance	Nodes	Edges
333SP	3712815	22217266
adaptive	6815744	27248640
as-Skitter	1696415	22190596
hollywood-2009	1139905	113891327
roadNet-CA	1971281	5533214
roadNet-PA	1090920	3083796
roadNet-TX	1393383	3843320

Table 5.2.: Summary of the used large matrices from the University of Florida Sparse Matrix Collection [39]

OBDD representation we use the interleaved variable order with decreasing significance and adopt the node labels provided by the files.

Graph Class: Unit Interval Graphs

Unit interval graphs can be represented as balanced nonnegative strings over {'[', ']'} (see, [113]) and such strings are created randomly using the algorithm in [8]. We generated 35 random graphs of size 2^i for $i = 10, \ldots, 23$. The label of the nodes correspond to the position in the sorted sequence of interval starting points as described in Section 3.2.2. The variable order for this graph class is the interleaved variable order with decreasing significance.

Limited Randomness: Pseudorandom Graphs

Our OBDD construction from Section 4.3 can be modified such that we get an input distribution in the following way: If the 1-sink of the OBDD is chosen with probability p as a successor of nodes in the last layer L_{n-1} the expected size of $|f^{-1}(1)|$ is $p \cdot 2^n$. Let $f : S \to B_{2n}$ be a random function generated by this modified construction and set $r(x, y) = f_s(x, y) \wedge (|x|_2 > |y|_2)$. Then we have a random graph by computing the function $\chi_E(x, y) = r(x, y) \vee r(y, x)$ with $\mathbf{Pr}[\chi_E(x, y) = 1] = p$ for $x \neq y$. Thus, the expected density $\frac{|E|}{\binom{N}{2}}$ is p.

Note that the graph is not a random graph where edges are independently chosen with probability p. This distribution is usually notated by $G(N, p)$ where N is the number of nodes and p the edge probability. Actually, our graph distribution is significantly different to $G(N, p)$ in the sense that many properties of $G(N, p)$ graphs are violated in our case, e. g., connectivity, existence of a perfect matching, and chromatic number [7]. Even more, Alon and Nussboim [7] showed that some properties of $G(N, p)$ graphs carry over to k-wise independent graphs for small $k = \operatorname{polylog} N$ but can not be guaranteed in (ε, k)-wise independent graphs

even for large values of k and small values of ε. Here, in an (almost) k-wise independent graph each edge is chosen with a fixed probability p and any subset of k edges is (almost) independent.

5.2. Implicit Maximum Matching Algorithm in Bipartite Graphs

Bollig, Gillé, and Pröger [18] presented two implicit maximum matching algorithms for bipartite graphs which we briefly described in Section 2.3: One algorithm is based on augmenting paths and will be denoted by AP and the other is based on a push-relabel technique which we denote by PR. We recall that the number of functional operations is $\tilde{\mathcal{O}}(N^{1/2+c} \cdot \sqrt{MO(N)} + N^{1-c})$ in the case of AP and $\tilde{\mathcal{O}}(N^{2/3} \cdot MO(N))$ in the case of PR, respectively where $MO(N)$ is the number of operation of the maximal matching subroutine. Both AP and PR need a priority function for breaking symmetries and we tested both priority functions from Section 2.3: $\Pi_1(x, y, z) = 1$ iff $|y|_2 < |z|_2$ and $\Pi_2(x, y, z) = 1$ iff $|y - x|_2 < |z - x|_2$. Initial experiments have shown that the running time of both implicit maximum bipartite matching algorithms using the second priority function Π_2 is significantly faster than the running time of the algorithms with the simple priority function Π_1. A reason can be that the second priority function leads to a better spreading of neighboring nodes and therefore to better results. These initial experiments have also shown that the maximal matching heuristic by Hachtel and Somenzi [57] is faster than the maximal matching algorithm by Bollig and Pröger [20]. Note that the asymptotically sublinear number of functional operations does not hold anymore if we use the algorithm by Hachtel and Somenzi. We only present the results of the implicit algorithms with respect to the second priority function and the maximal matching heuristic by Hachtel and Somenzi. We also compare AP and PR empirically with an explicit implementation of the well-known Hopcroft-Karp algorithm [61], in the following HK for short.

Experimental Setup

All algorithms have been implemented in C++ and we have used the OBDD package CUDD 2.4.2 by Somenzi[4] for the implicit algorithms and LEDA[5] 6.3 graph libraries for HK. The experiments have been performed on a computer with a 3 GHz Intel Core Duo processor and 2 GB main memory running Ubuntu 11.04. The sources have been compiled with the g++ 4.5.2 compiler and optimization flag O3. The running time has been measured by used processor time in seconds, and the space usage of the implicit algorithm is given by the maximum SBDD size which came up during the computation, where a SBDD is a collection of OBDDs which can share nodes. Note that the SBDD size is independent of the used computer

[4]CUDD is available at `http://vlsi.colorado.edu/` (Retrieved: July 2015)
[5]LEDA is available at `http://www.algorithmic-solutions.com/` (Retrieved: July 2015)

system. For our results we have taken the mean value over 50 experiments on graphs with the same number of nodes. Only the graphs belonging to the `rope` class have been generated randomly. Due to the small variance of these values, we only show the mean value in the diagrams.

The OBDD package CUDD provides some heuristics to minimize the size of the OBDDs, which are generated by a functional operation, by changing the variable order. Since initial experiments have shown that this feature slows down our implicit maximum bipartite matching algorithms, it has been disabled for our experiments. The variable order has been fixed throughout the whole execution of the implicit algorithms. In the paragraphs of the graph classes we described in detail which variable order we have chosen.

Implementation of HK

The key idea of HK [61] is to augment the current matching with a maximal set of shortest node-disjoint augmenting paths. Our implementation of HK begins with an empty matching and repeats the following steps until the set of augmenting paths is empty: We find all shortest augmenting paths via breadth first search and delete all edges which are not on such a path. Next, we add shortest augmenting paths one by one via depth first search. After we have found one path, we delete all edges which are incident to one node on the path to ensure that the paths are node-disjoint. Finally, the current matching is augmented by a maximal set of node-disjoint augmenting paths.

Space Usage

It is easy to see that HK needs $\mathcal{O}(|V| + |E|)$ space while the space needed to store an OBDD of size S is $\mathcal{O}(S \log S)$. Therefore, we can roughly compare the space usage of HK and the implicit algorithms if we know the size of the input graph and the maximum OBDD size which is generated during the computation of the implicit algorithm.

Results

The experiments show that both implicit algorithms perform nearly identically and are very fast and space efficient on grid graphs (see Fig. 5.2). They outperform HK with respect to the running time with more than four million nodes. The space requirement of the implicit algorithms is significantly smaller on every grid graph and, in fact, the space usage is sublinear with respect to the number of nodes in the grid graph (see Fig. 5.2). The memory limit of 2 GB is exceeded by HK on graphs with more than eight million nodes while the implicit algorithms use only few megabytes memory even on grid graphs with more than a billion nodes. Grid graphs demonstrate the potential of implicit matching algorithms but are very simple graphs. On graphs from the class `rope` AP outperforms both PR and HK (see Fig.

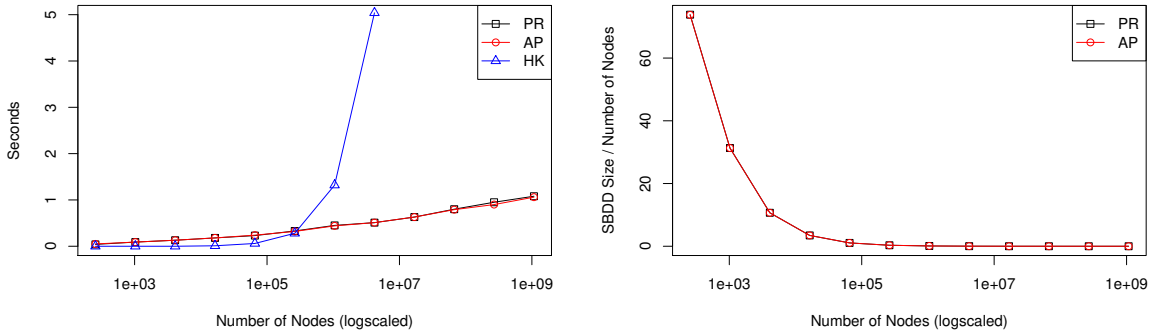


Figure 5.2.: Experimental results of the implicit augmenting-path-based matching algorithm (AP), the implicit push-relabel-based algorithm (PR) and HK on grid graphs.

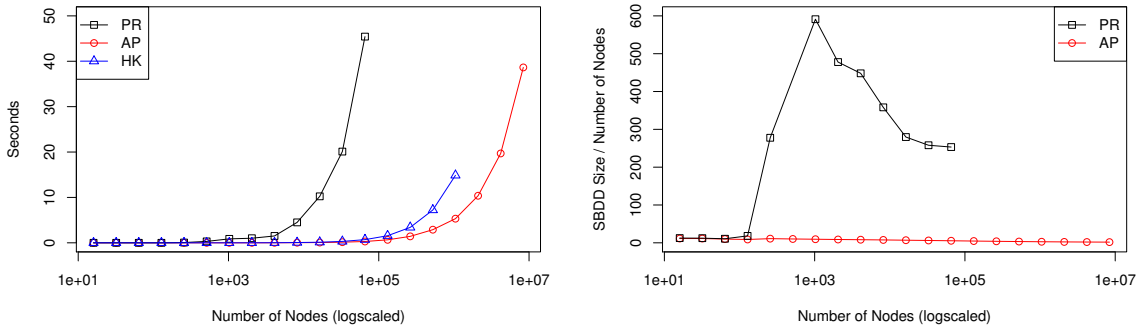


Figure 5.3.: Experimental results of the implicit augmenting-path-based matching algorithm (AP), the implicit push-relabel-based algorithm (PR) and HK on `rope` graphs.

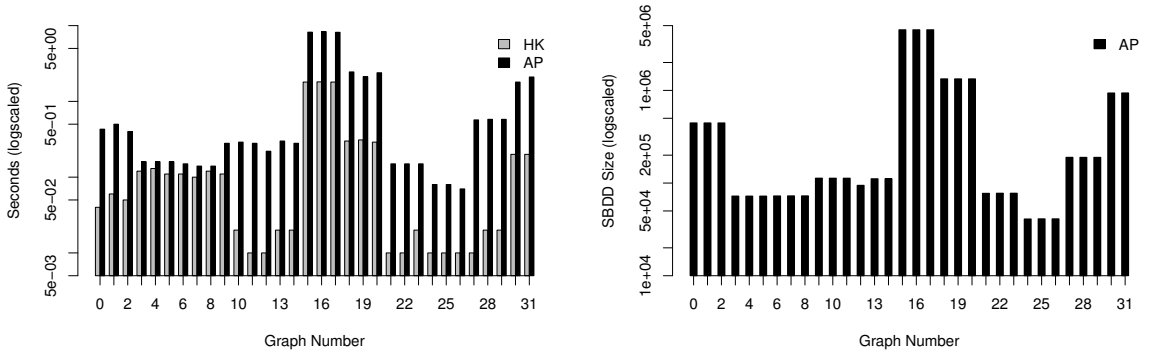


Figure 5.4.: Experimental results of the implicit augmenting-path-based matching algorithm (AP) and HK on real-world instances.

5.3). The space requirement of the implicit algorithms corresponds to the running time of each algorithm. The fast algorithm deals with small OBDDs and in the case of the slower algorithm the size of the OBDDs is large. Fig. 5.3 also shows that the space usage of AP on these instances is linear with respect to the number of nodes while the space usage of PR tends to be superlinear. Our experiments on real-world graphs indicate that PR cannot be used for practical instances. Even on small input graphs we were not able to execute this algorithm because of our memory limit of 2 GB. On these instances HK yields better running times but the running times of AP are competitive (see Fig. 5.4) and we were able to observe that the real space usage is slightly better than in the explicit case.

Summary and Conclusion

Overall, PR does not seem to be applicable on large real-world instances but AP performs very well on more complex but sufficiently structured graphs. Though HK runs faster than AP on real world instances, the gap between both algorithms is not so large as to rule out the possibility of using the latter algorithm for sufficiently structured practical problems.

Our results show that the number of functional operations does not correspond directly to the practical performance of the algorithms but the number of variables seems to have a significant influence on the space usage of the implicit algorithm. Recall that the number of variables is $\max\{3\log N, MV(N)\}$ for AP and $\max\{4\log N, MV(N)\}$ for PR. The maximal matching algorithm by Bollig and Pröger [20], i. e., $MV(N) = 6\log N$, was only working on grid graphs whereas the heuristic by Hachtel and Somenzi [57], i. e., $MV(N) = 3\log N$, performs well on every chosen input instance although the usage rules out a guarantee of a sublinear number of functional operations. These performances are going to be confirmed in Section 5.4 where solely maximal matching algorithms are compared.

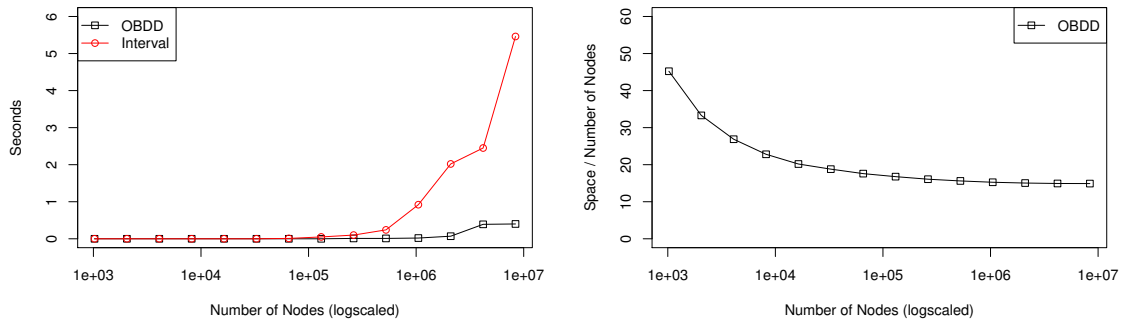


Figure 5.5.: Running time and space usage of the matching algorithms on random unit interval graphs. The second plot shows the ratio of $S \log S$ (space usage of the OBDD-based algorithm) and the number of nodes.

5.3. Implicit Maximum Matching Algorithm in Unit Interval Graphs

First, we start with a brief discussion about the performance of the coloring algorithms on (unit) interval graphs. The implicit algorithm performed poorly even on instances of size around 2000. At a first glance, this might not be surprising due to the complex coloring algorithm but having a closer look we see that, for instance, the implicit matching algorithm which is evaluated in this section is optimized for the implicit setting while the coloring algorithm uses some nice ideas to simulate the sequential algorithm. Hence, these results do not rule out the possibility of an efficient implicit coloring algorithm but suggest that there have to be new ideas to benefit more from the strengths of implicit algorithms.

For the evaluation, we compare the OBDD-based matching algorithm from Section 3.3 to the algorithm which gets the interval representation as an input, sort the intervals according to their left endpoints and compute a maximum matching by scanning this sorted sequence with the same idea used in the implicit algorithm.

Experimental Setup

We implemented the implicit algorithm with the BDD framework CUDD 2.5.0 by F. Somenzi. The algorithms are implemented in C++ and were compiled with Visual Studio 2012 in the default release configuration. The experiments were performed on a computer with a 2.6 GHz Intel Core i5 processor and 4 GB main memory running Windows 7. The runtime is measured by used processor time in seconds and the space usage of the implicit algorithm is given by the maximum SBDD size which came up during the computation. Due to the small variance of these values, we only show the mean in the diagrams.

Results and Conclusion

The implicit matching algorithm outperforms the explicit matching algorithm on unit interval graphs (see Fig. 5.5). Even on graphs with more than 8 million nodes the implicit algorithm computes a maximum matching within 1 second. Storing an SBDD of size S needs $\mathcal{O}(S \log S)$ bits. The memory diagram shows that the asymptotic space usage of the implicit algorithm on these instances is close to $\mathcal{O}(N)$. Recall that the unit interval representation needs $\Theta(N \log N)$ space since $\log N$ bits are needed to represent the starting points. All in all, the implicit algorithm needs less space and can compute a maximum matching on larger instances than the explicit one. An interesting consequence of these results is that the submodules of our maximum matching algorithm, namely computing the connected components, a Hamiltonian path in every connected component and a maximum matching on these paths, are also very fast and space efficient which is surprising, since especially the computation of the transitive closure is often a bottleneck in implicit algorithms.

5.4. Randomized and Deterministic Implicit Maximal Matching Algorithms

In order to evaluate the performance of the randomized maximal matching algorithm from Section 4.4, we can choose between three deterministic maximal matching algorithms: the heuristic by Hachtel and Somenzi (HS) [57], HS combined with a strategy of Karp and Sipser [20], and the algorithm by Bollig and Pröger [20]. All these three algorithms are described in Section 2.3. We focus here only on HS because the combination of HS and Karp-Sipser is intended to result in a larger matching rather than in a faster computation and, as we mentioned before, we were not able to run the maximal matching algorithm by Bollig and Pröger because the memory limitation was exceeded on every instance presented here.

For the comparison, we choose three types of input instances: First, we run the algorithms on the bipartite graphs from the real advertisement application. The motivation was to check whether the randomized algorithm is competitive or even better on instances where HS is running very well. Second, we use non-bipartite graphs from the university of Florida sparse matrix collection. Since HS is designed for bipartite graphs, a preprocessing step computing a bipartition of these graphs are needed to compute a maximal matching (see, e. g., [20]) while our algorithm also works on general graphs. Finally, we use the pseudorandom graphs with a fixed number of $N = 2^{17}$ nodes and variable density to investigate which algorithm is performing better in dense and sparse graphs. For this we generated OBDDs representing graphs (as described in Section 5.1) with width $w = k^2/(2\varepsilon) \cdot \log N$ where ε was fixed to 2^{-k} and $k \in \{4, \ldots, 7\}$. The range for the density parameter p was $\{0.05 \cdot i \mid i \in \{1, \ldots 19\}\}$.

Sparsification

Recall that we mainly use random functions to sample a subgraph from the original graph which can also be seen as some kind of graph sparsification. We will see that sparsification is a good heuristic to get faster algorithms. The reason is probably the fact that the worst-case OBDD size of a function f is bounded by the number of inputs in $f^{-1}(1)$. We use the following implementation of our randomized maximal matching algorithm which we denote by RM: In order to minimize the running time for the computation of the set of nodes with two or more incident edges, we sparsify the graph at the beginning of the outer while loop (see Algorithm 11) by deleting each edge with probability $1/2$ and repeating this D times. Initially, we set $D = \log |E|$ and decrease D by 1 at the end of the outer loop. Asymptotically, the running time does not change since after $\mathcal{O}(\log N)$ iterations, i. e., $D = 0$, it does exactly the same as the original algorithm. Initial experiments showed that this is superior to the original algorithm.

Experimental Setup

All algorithms are implemented in C++ using the BDD framework CUDD 2.5.0 by F. Somenzi and were compiled with Visual Studio 2013 in the default 32-bit release configuration. The experiments were performed on a computer with a 2.5 GHz Intel Core i7 processor and 8 GB main memory running Windows 8.1. The runtime is measured by used processor time in seconds and the space usage of the implicit algorithm is given by the maximum SBDD size which came up during the computation. For our results, we took the mean value over 50 runs on the same graph. In the case of the randomly generated graphs we present the mean value for the running time and space usage in the diagrams and the standard deviation of these values in Appendix B.1. Due to the small variance of the values on the other graphs, we only show the mean in the diagrams/tables.

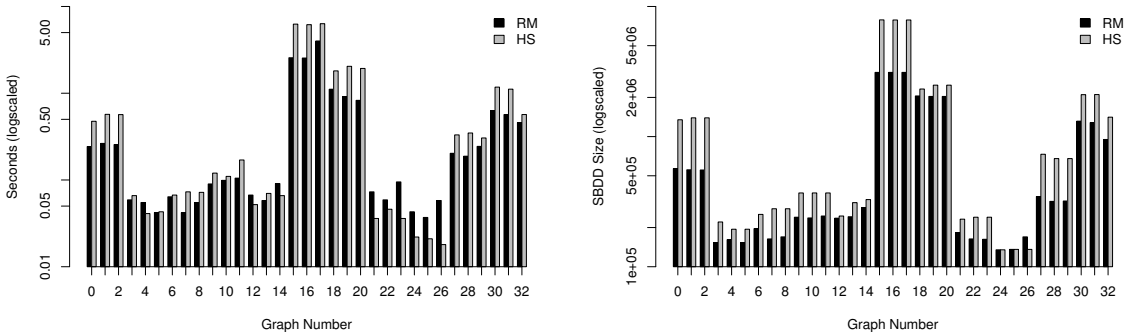


Figure 5.6.: Running times and space usage of HS and RM on the real world instances.

Instance	Nodes	Edges	Time (sec)	Space (SBDD size)
333SP	3712815	22217266	1140.54	66968594
adaptive	6815744	27248640	403.82	22767094
as-Skitter	1696415	22190596	337.53	32020282
hollywood-2009	1139905	113891327	418.36	62253086
roadNet-CA	1971281	5533214	136.18	13177668
roadNet-PA	1090920	3083796	75.26	7633318
roadNet-TX	1393383	3843320	92.62	9125438

Table 5.3.: Running time and space usage of RM on the graphs from [39]

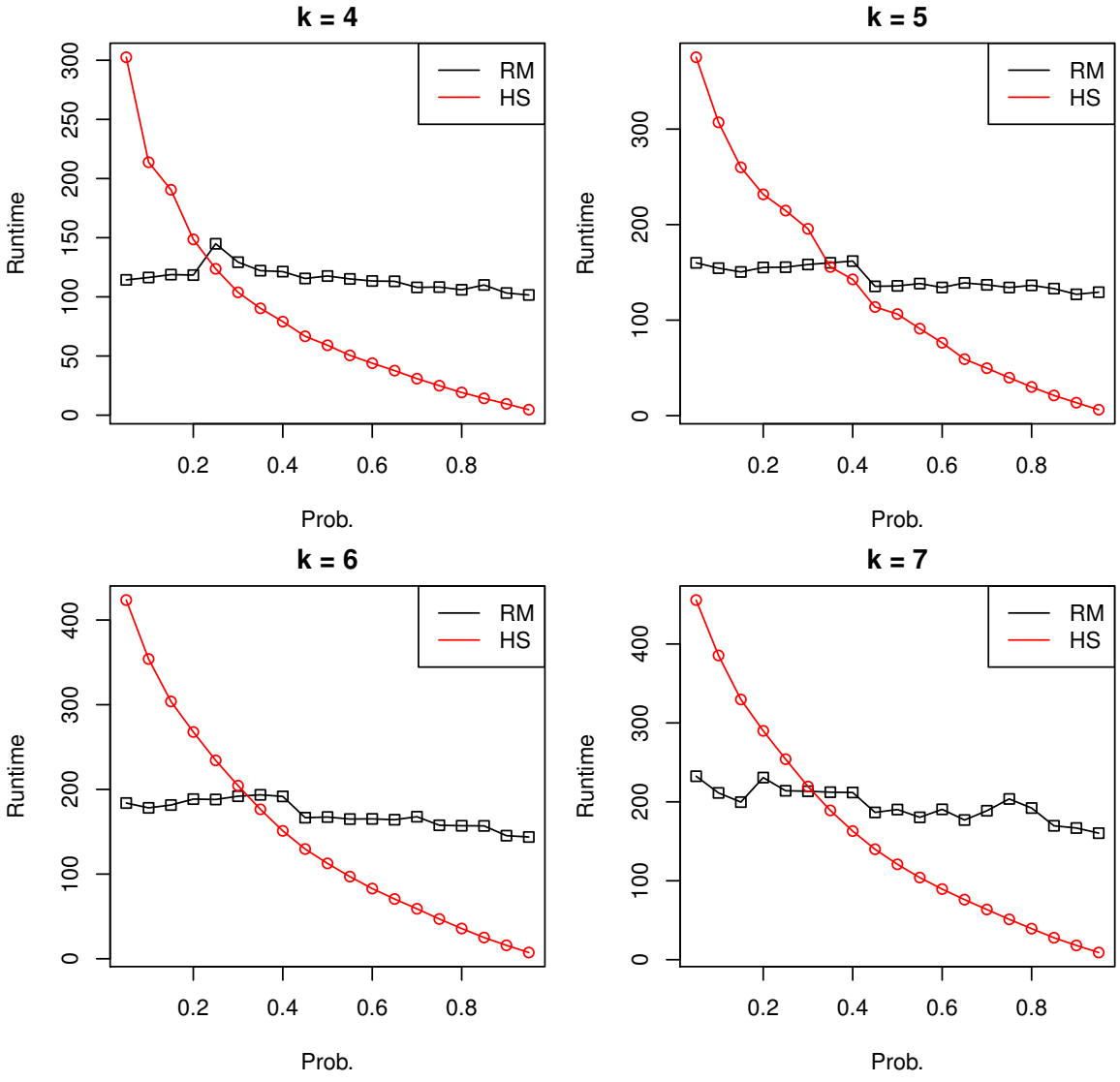


Figure 5.7.: Running times of HS and RM on the random instances.

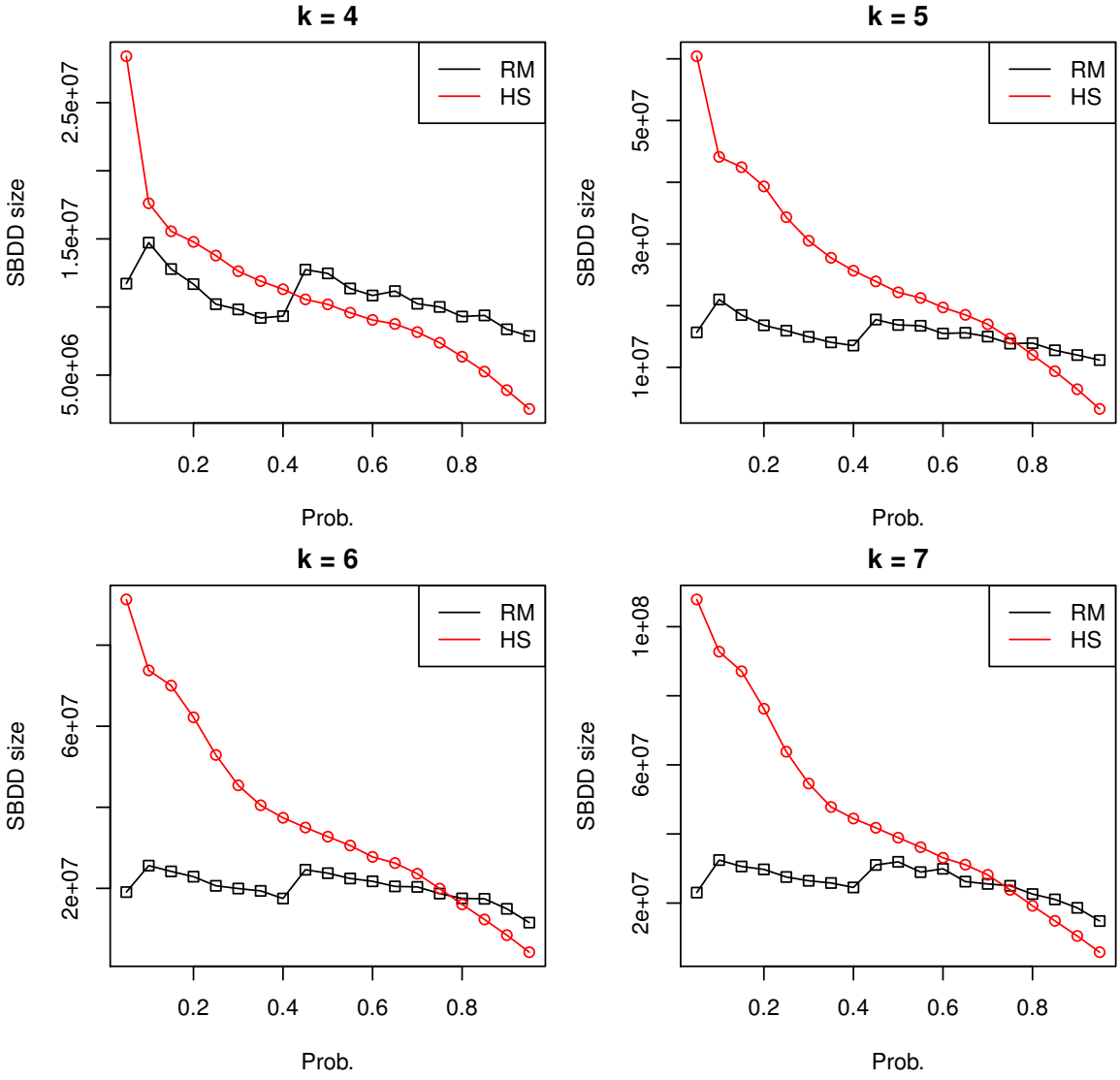


Figure 5.8.: Space usage of HS and RM on the random instances.

Results and Conclusion

On the random instances the running time and space usage of RM was more or less unaffected by the density of the graph while HS was very slow for small values of p and gets faster with increasing density. The performance of HS exhibits a strictly monotonically decreasing behavior with increasing density (Fig. 5.7). For low densities ($p \leq 0.2$) RM was much faster than HS. In terms of space usage (Fig. 5.8) RM is on densities up to 0.7 better than HS.

In Fig. 5.6 we see that on the bipartite real world instances RM is similar to HS if the

running time is negligibly small but on the largest instances (number 15 to 20) RM is much faster. The graphs from [39] were intentionally chosen to show the potential of RM and indeed do so: It was not possible to run HS on these graphs due to memory limitations whereas RM computed a matching in reasonable time and space (see Table 5.3).

Both graphs from [105] and [39] have very small density and the experiments on the random graphs seem to support the hypothesis that RM is a better choice than HS for such graphs. The running time of RM was very robust and did not have much variation for different densities which is quite surprising because of the randomization.

5.5. Randomized and Deterministic Implicit Minimum Spanning Tree Algorithms

We compare the randomized minimum spanning tree algorithm from Section 5.5 with three different deterministic implicit algorithm: First, the algorithm by Bollig [15] using $\mathcal{O}(\log^3 N)$ functional operations. Then two implicit variants of the known explicit algorithms for computing a minimum spanning due to Prim and Kruskal (see, for instance, [35]). These algorithms are simple enough to present them in Appendix B.2. We choose several instances from the university of Florida sparse matrix collection. These instances are unweighted. Thus, for every graph with N nodes we initially chose random weights for every edge from the set $\{1, \ldots, \sqrt{2N}\}$ resulting in 79 weighted graphs of size between 1000 and 2000. For a detailed overview of the instances, we refer to Appendix B.2. The instances are sorted in ascending order of the number of nodes.

Experimental Setup

The implementation and the conduction of the experiments were done by Michael Capelle. All algorithms are in C++ using the BDD framework CUDD 2.5.0 by F. Somenzi and were compiled with the g++ 4.5.2 compiler using Cygwin and with default optimization parameters. The experiments were performed on a computer with a 3 GHz Intel Core Duo processor and 2 GB main memory running Windows Vista. The runtime is measured by used processor time in seconds and the space usage of the implicit algorithm is given by the maximum SBDD size which came up during the computation. For our results, we took the mean value over 50 runs on the same graph. Due to the small variance of the values, we only show the mean in the diagrams.

Results and Conclusion

The randomized MST algorithm is outperforming all other algorithms on almost every instance in terms of running time (see, Fig. 5.9 and Fig. 5.10). This is surprising since the

number of variables used in this algorithm is also $6 \log N$ as in the deterministic algorithms. This large number of variables is due to the priority functions that choose the smallest edge out of a set of candidates in all of the algorithms. It seems that the reduced number of Borůvka steps in comparison to the algorithm by Bollig and the subgraph sampling works well on these instances. In terms of space usage the implicit variants of Prim and Kruskal are using much less space than the other two algorithms. This is often observed when the implicit algorithms are working in a sequential way. But this is also the reason that the lower space usage does not reflect in a better running time. The randomized algorithm uses less space than the algorithm by Bollig but the difference is not as remarkable as in the case of running time.

Unfortunately, all of these instances are very small and the algorithms still have a quite large running time. However, despite the large number of variables and the unknown number of functional operations, the randomized algorithm performs well and it might be promising to further improve or even design a new randomized implicit MST algorithm with a smaller number of variables or a guarantee on the number of functional operations.

Figures

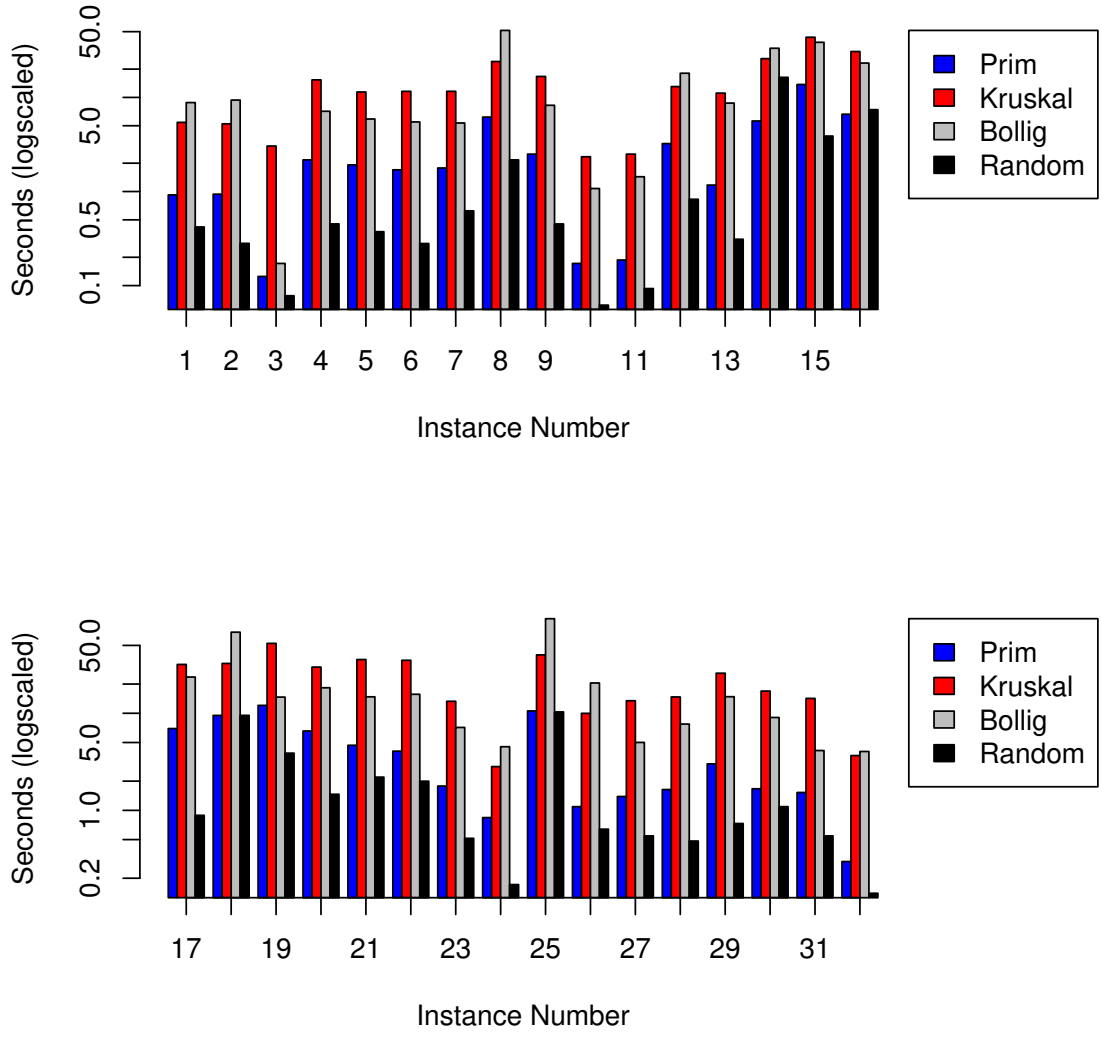


Figure 5.9.: Running times of the MST algorithms on some graphs from [39]

.

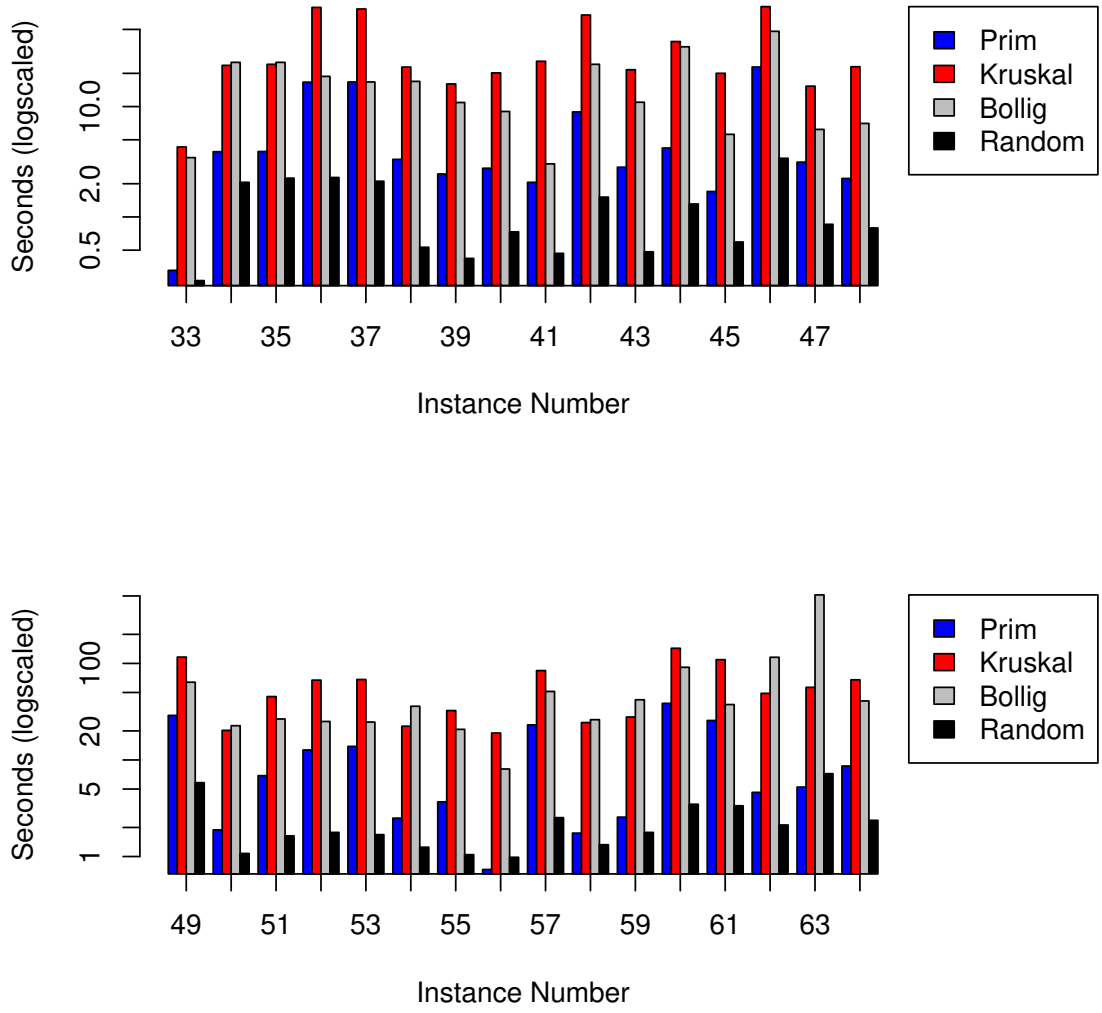


Figure 5.10.: Running times of the MST algorithms on some graphs from [39]

.

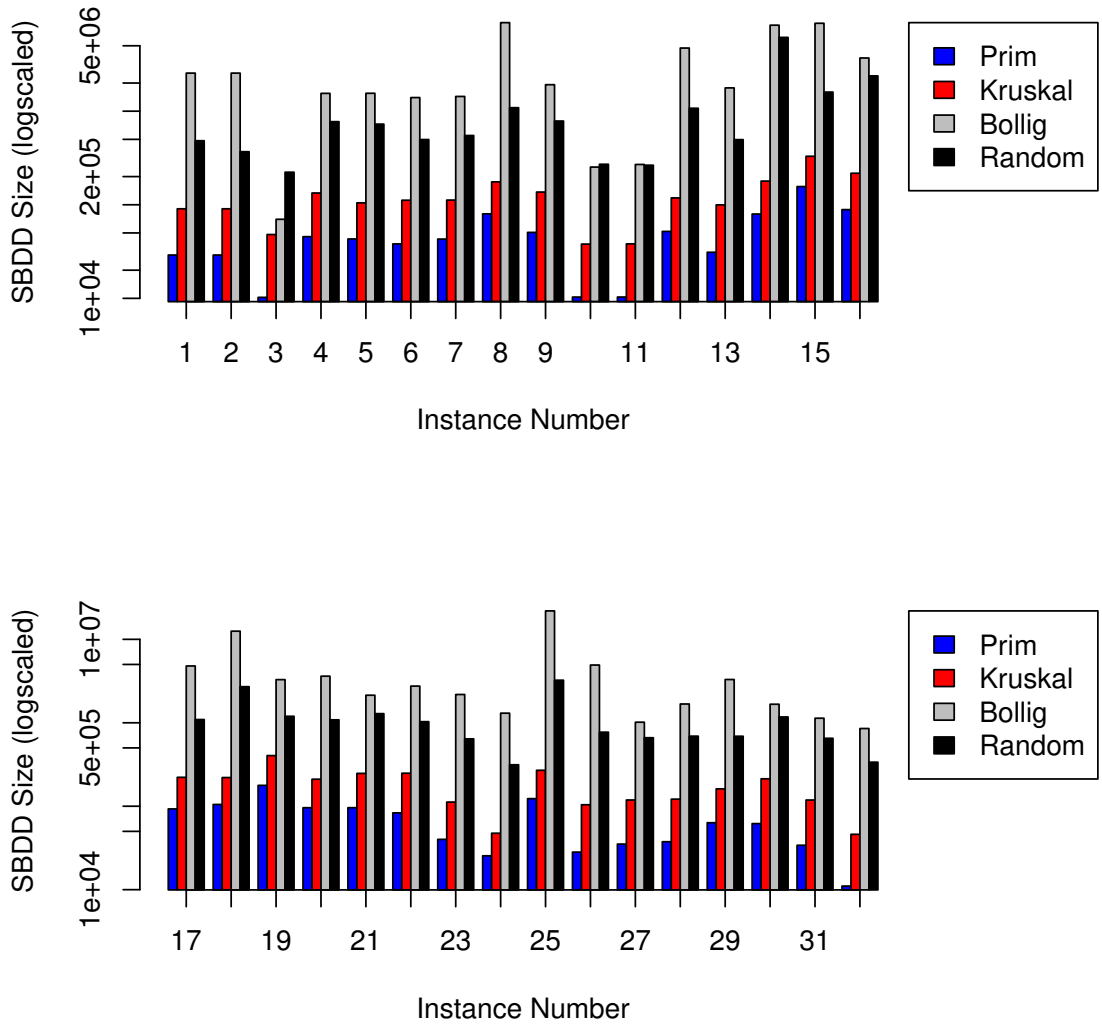


Figure 5.11.: Space usage of the MST algorithms on some graphs from [39]

.

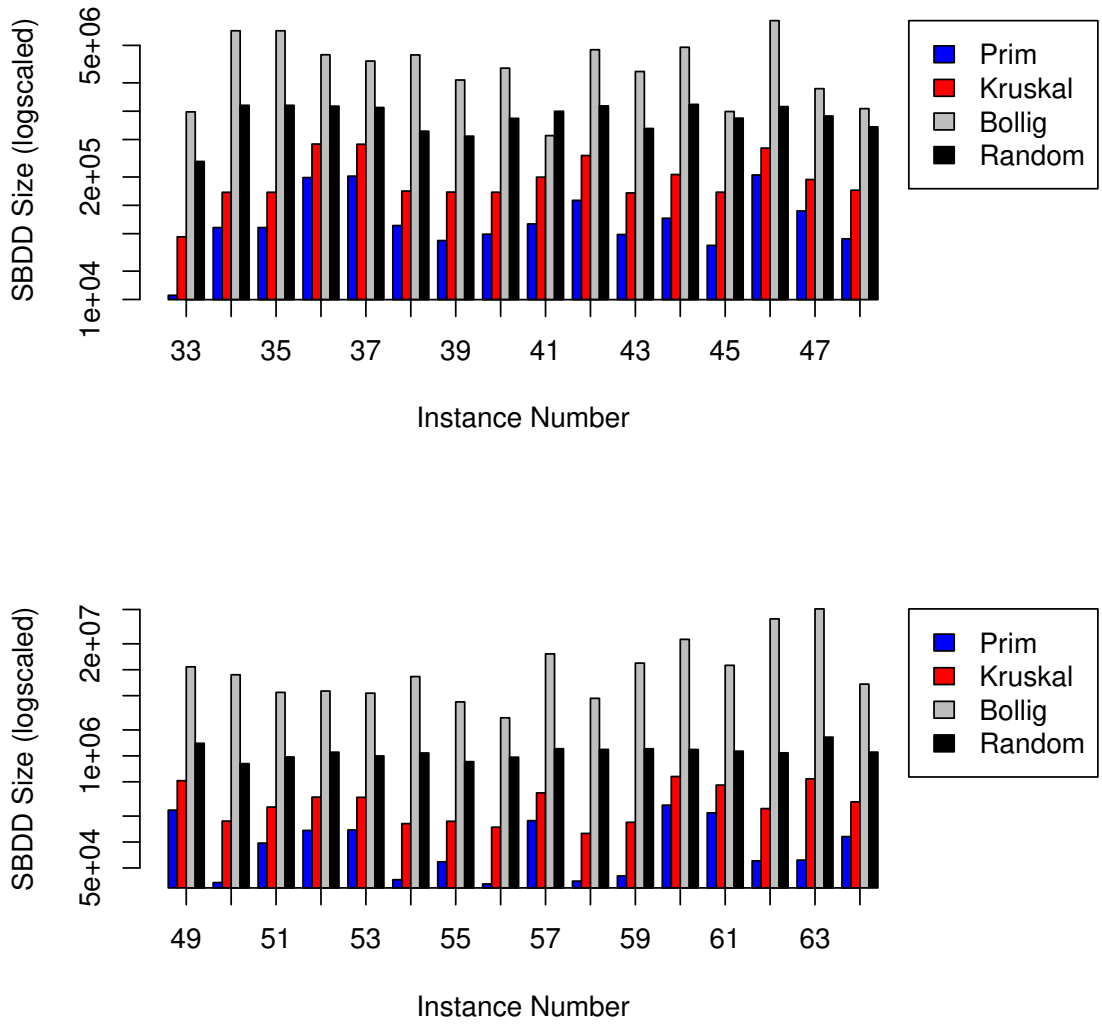


Figure 5.12.: Space usage of the MST algorithms on some graphs from [39]

.

6. Estimation of Matchings in Dynamic Data Streams

This chapter deals with the estimation of weighted and unweighted matchings in dynamic graph streams. It is a very active field of research which we summarize in the first section of this chapter. Then in Section 6.2 we give an algorithm for estimating the value of an optimal weighted matching in a graph by using a black box algorithm that estimates the size of a maximum matching in a graph. In Section 6.3, we extend a known matching size estimation algorithm for a class of sparse graphs (including planar graphs) to dynamic graph streams and improve an estimation algorithm for trees. Section 6.4 concludes the algorithmic part of this chapter by showing applications of the weighted matching estimation algorithms using the algorithms from Section 6.3 and known estimation algorithms. In the last Section 6.5, we prove a space lower bound for every streaming algorithm estimating the matching size up to a $(1+\varepsilon)$-factor for small ε.

6.1. Related Work and Contribution

Matchings in data streams are very well-studied and there is a lot of recent works on this topic. We summarize these results grouped by the two problems that are investigated here: unweighted matching and weighted matching.

Unweighted Matching

Insertion-only Streams: As mentioned by Feigenbaum et al. [45], maintaining a 2-approximation to the maximum matching (MM) in an insertion-only stream can be straightforwardly done by greedily maintaining a maximal matching. Improving on this algorithm turns out to be difficult as Goel, Kapralov, and Khanna [53] showed that even in bipartite graphs no algorithm using $\tilde{\mathcal{O}}(N)$ space can achieve an approximation ratio better than $\frac{3}{2}$. This lower bound was later improved by Kapralov to $\frac{e}{e-1}$ [73]. Nevertheless, Konrad, Magniez, and Mathieu [81] gave an algorithm using $\tilde{\mathcal{O}}(N)$ space with an approximation factor of 1.989 if the edges are assumed to arrive in random order. For distributed matching among k players, Huang et al. [63] gave a lower bound of $\Omega(k \cdot N/\alpha^2)$ for any approximation factor $\alpha > 1$.

To bypass the natural $\Omega(N)$ bound required by any algorithm maintaining an approximate matching, recent research has begun to focus on estimating the size of the maximum matching. Kapralov, Khanna, and Sudan [74] gave a polylogarithmic approximate estimate using polylogarithmic space for random order streams. For certain sparse graphs including planar graphs, Esfandiari et al. [44] described how to obtain a constant factor estimation using $\tilde{\mathcal{O}}(N^{2/3})$ space in a single pass and $\tilde{\mathcal{O}}(\sqrt{N})$ space using two passes or assuming randomly ordered streams. They also gave a $(2+\varepsilon)$-estimation on the size of a maximum matching in trees using $\mathcal{O}(\sqrt{N})$ space. The authors showed a lower bound of $\Omega(\sqrt{N})$ for any approximation better than $\frac{3}{2}$.

Dynamic Streams: Chitnis et al. [29] gave an 1-pass algorithm that computes a maximal matching of size at most k in $\tilde{\mathcal{O}}(k \cdot N)$ space. Recent results by Assadi et al. [9] showed that approximating matchings in dynamic streams is hard by providing a space lower bound of $\Omega(N^{2-3\varepsilon})$ for approximating the maximum matching within a factor of $\tilde{\mathcal{O}}(N^{\varepsilon})$. Simultaneously, Konrad [80] showed a weaker lower bound of $\Omega(N^{3/2-4\varepsilon})$. Both works presented an algorithm with an almost matching upper bound on the space complexity of $\tilde{\mathcal{O}}(N^{2-2\varepsilon})$ [80] and $\tilde{\mathcal{O}}(N^{2-3\varepsilon})$ [9]. Chitnis et al. [28] improved their first result and gave a streaming algorithm using $\tilde{\mathcal{O}}(k^2)$ space that returns an exact maximum matching under the assumption that the size is at most k. It is important to note that all these results actually compute a matching. In terms of estimating the size of the maximum matching, Chitnis et al. [28] extended the estimation algorithms for sparse graphs from [44] to the settings of dynamic streams using $\tilde{\mathcal{O}}(N^{4/5})$ space. This was published almost simultaneously to our similar result which we describe later along with a discussion of the differences.

For multipass streaming algorithms we refer to the survey by McGregor [95]. A bridge between dynamic graphs and the insertion-only streaming model is the sliding window model studied by Crouch, McGregor, and Stubbs [37]. In this model only the last w entries of the stream are considered for a parameter $w > 0$. The authors gave a $(3+\varepsilon)$-approximation algorithm for maximum matching using $\mathcal{O}(N \operatorname{polylog} N)$ space.

Weighted Matching

Insertion-only Streams: For maximum weighted matching (MWM), a series of results have been published starting with a greedy algorithm yielding a 6-approximation [45] which was continuously improved [94, 42, 133, 43] with the current best bound of $4+\varepsilon$ being due to Crouch and Stubbs [36]. All these algorithms return a matching and need $\mathcal{O}(N \operatorname{polylog} N)$ space.

Dynamic Streams: Crouch and Stubs [36] gave a reduction that showed that solving $\mathcal{O}(\frac{1}{\varepsilon}\log N)$ MM instances gives a $2(1+\varepsilon)$-approximation to MWM. The MM instances are given by separating the edges by weight which can easily be done in a stream. Thus, we can use every MM algorithm for dynamic streams to approximate the MWM problem. Chitnis

et al. [28] showed that their results for MM can be extended to MWM with an additional $\mathcal{O}(\frac{1}{\varepsilon}\log N)$ factor resulting from separating the edges by weight.

Other Results

The p-Schatten norm of a matrix $A \in \mathbb{R}^{N\times N}$ is defined as $||A||_{S_p} = \left(\sum_{i=1}^{N}\sigma_i^p\right)^{\frac{1}{p}}$ where σ_i is the i-th singular value of A. Common special cases include the Frobenius norm $||A||_{S_2} = \sqrt{\sum_{i,j}A_{ij}^2}$, and the rank $||A||_{S_0} = |\{\sigma_i|\sigma_i \neq 0\}|$. As we know from Chapter 2, computing the maximum matching size is equivalent to computing the rank of the Tutte matrix. Estimating the maximum matching size therefore is a special case of estimating the rank of a matrix. Li, Nguyen, and Woodruff [85] showed that, with the exception of the Frobenius norm, approximating Schatten norms is far more difficult than approximating the vector frequency counterpart. Any estimation of the rank within any constant factor is shown to require $\Omega(N^2)$ space when using so-called bi-linear sketches and $\Omega(\sqrt{N})$ space for general linear sketches. It should be noted that with the exception of $||A||_{S_2}$, all known algorithms for Schatten norms and graph problems in dynamic streams are based on bi-linear sketches.

	Reference	Graph class	Streaming model	Approx. factor	Space
MM:	greedy	General	Insertion-only	2	$\mathcal{O}(N)$
	[74]	General	Random	$\operatorname{polylog} N$	$\operatorname{polylog} N$
	[44]	Trees	Insertion-only	$2+\varepsilon$	$\tilde{\mathcal{O}}(\sqrt{N})$
	[44]	Bounded arboricity	Insertion-only	$\mathcal{O}(1)$	$\tilde{\mathcal{O}}(N^{2/3})$
	here	Trees	Dynamic	$2+\varepsilon$	$\mathcal{O}(\frac{\log^2 N}{\varepsilon^2})$
	here	Bounded arboricity	Dynamic	$\mathcal{O}(1)$	$\tilde{\mathcal{O}}(N^{4/5})$
	[44]	Forests	Insertion-only	$\frac{3}{2}-\varepsilon$	$\Omega(\sqrt{N})$
	here	General	Insertion-only	$1+\mathcal{O}(\varepsilon)$	$\Omega\left(N^{1-\varepsilon}\right)$
MWM:	[36]	General	Insertion-only	$4+\varepsilon$	$\mathcal{O}(N\log^2 n)$
	here	General	Random	$\operatorname{polylog} N$	$\operatorname{polylog} N$
	here	Bounded arboricity	Dynamic	$\mathcal{O}(1)$	$\tilde{\mathcal{O}}(N^{4/5})$

Table 6.1.: Results for estimating the size (weight) of a maximum (weighted) matching in data streams. In the random streaming model the edges arrive in random order but can also only be inserted.

Techniques and Contribution

Table 6.1 gives an overview of the results presented here in comparison to previously known algorithms and lower bounds. Our first main result is an approximate estimation algorithm for the maximum weight of a matching. We give a generic procedure using any unweighted estimation as black box. In particular, we show that for every λ-approximate estimation algorithm for the unweighted matching problem using S space, there exists an $\mathcal{O}(\lambda^4)$-approximate

estimation algorithm for the weighted matching problem using $\mathcal{O}(S \cdot \log N)$ space. Combining this theorem with the recent result of Kapralov, Khanna, and Sudan [74] gives a polylog(N) space and polylog(N)-approximate estimate for weighted matching in random order streams which is the first sublinear streaming algorithm for the MWM problem.

We can implement this algorithm in dynamic streams. Building on the work by Esfandiari et al. [44], we give a constant estimation on the matching size of bounded arboricity graphs in dynamic streams, which was also independently obtained in [28]. Both our approach and the algorithm of Chitnis et al. [28] yield identical space bounds of $\tilde{\mathcal{O}}(N^{4/5})$, which immediately extends to weighted matching. The difference is that their algorithm uses a sampling routine to recover small matchings while we use the Tutte matrix to estimate the size of small matchings. In some sense their result is stronger. However, our technique is of independent interest: For instance, using the connection between Tutte matrix and matching we can show a space lower bound for algorithms approximating the rank in data streams. In addition, we also present an algorithm maintaining a small matching in two passes using sublinear space. Surprisingly, in terms of space usage the algorithms using the Tutte matrix and the algorithms maintaining a small matching need the same amount in the end. We also improve the estimation algorithm for trees [44] by presenting a simple algorithm for dynamic streams using only $\mathcal{O}(\frac{1}{\varepsilon^2} \log^2 N)$ space.

Our lower bound is proven via reduction from the Boolean Hidden Hypermatching problem (see, Definition 2.6.2) introduced by Verbin and Yu [126]. Recall that in this problem two players Alice and Bob are given a binary N-bit string and a perfect t-hypermatching on N nodes, respectively. Bob also gets a binary string w. The players are promised that the parity of bits corresponding to the nodes of the i-th hypermatching either are equal to w_i for all i or equal to $1-w_i$ for all i and the task is to find out which case holds using only a single round of communication. We construct a graph consisting of N nodes and a t-clique for each hyperedge of Bob's matching and a single edge for each bit of Alice's input that has one node in common with the t-cliques. Then we show that approximating the matching size within a factor better than $1 + O(1/t)$ can also solve the Boolean Hidden Hypermatching instance. Using the lower bound of $\Omega(N^{1-1/t})$ from [126], we have that any 1-pass streaming algorithm approximating the size of the maximum matching up to an $(1 + O(\varepsilon))$-factor requires $\Omega(N^{1-\varepsilon})$ bits of space. This lower bound also implies an $\Omega(N^{1-\varepsilon})$ space bound for $1+O(\varepsilon)$ approximating the rank of a matrix in data streams which also improves the $\Omega(\sqrt{N})$ bound by Li, Nguyen, and Woodruff [85] for general linear sketches.

6.2. Estimation of Weighted Matchings

The greedy algorithm for MWM simply adds an edge to the matching (and removes adjacent edges) if the weight of the matching increases. It is easy to see that this algorithm does

Figure 6.1.: Edges are streamed in ascending order with respect to their weights. The solution of the greedy algorithm is N whereas the optimal solution has weight at least $\sum_{i=1}^{N/2} 2i \approx N^2/4$.

not give any guarantee on the solution (see Fig. 6.1). Previous approximation algorithms for weighted matchings in insertion only streams analyzed in [45, 94, 42, 133, 36] extend this greedy approach by a charging scheme: They partition the set of edges such that they can charge the weight of edges in an optimal solution to edges in the approximation. To achieve this, they either explicitly partition the edges into sets of geometrically increasing weights or this partitioning is done implicitly by adding a new edge if the weight of the matching locally increases by a given threshold. The aforementioned algorithms are either greedy or they are maintaining a matching in each partition. Therefore, they cannot have sublinear space in an insertion-only stream and they need at least $\Omega(N^{2-3\varepsilon})$ in a dynamic stream even when the maintained matching is only an $\mathcal{O}(N^{\varepsilon})$ approximation which follows from the lower bound by Assadi et al. [9]. We overcome this problem by estimating just the weight of a weighted matching. For this, we use a similar charging scheme, but with a twist. As estimation routines do not necessarily give information on distinct edges, single edge weights cannot be charged to an edge with larger weight. However, entire matchings can be charged as the contribution of edges in a specific range of weights, for instance in $[r, 2 \cdot r]$, can only be large if these edges take up a significant part of any maximum matching in the subgraph containing only the edges of weight at least r.

A coarse description of our algorithm is as follows: First, we partition the edges into sets of geometrically increasing weights. For the sake of simplicity, we assume that the edge weights are of the form 2^i with $i \leq c \log N$. Then for every $0 \leq i \leq c \log N$ we estimate the size of the maximum matching in the subgraphs containing only edges with weight at least 2^i. Essentially, we want to estimate the contribution of the edges with weights in $[2^i, 2^{i+1})$ and sum these up over i. To estimate this contribution we look at the difference of the estimators for the maximum matching size in the subgraph for i and $i+1$. This is where the estimation routine for the size of a maximum matching comes into play. Clearly, the difference is not a good estimator for the contribution of the edges with weights in $[2^i, 2^{i+1})$ in general. But whenever these edges have a significant contribution to the maximum weighted matching the difference will also get a good estimator for the contribution: The maximum matching in the subgraph for i has to be much larger than the size of the matching in the subgraph for $i+1$ since the edge weights are geometrically increasing in i. However, another problem is that we want to estimate a difference $a - b$ by estimating a and b separately. This is only sufficiently good if a is by a constant factor larger than b. Fortunately, in our case a has to be larger than b if the contribution of the edges is significant. In the notion which we define more formally

Algorithm 12 Approximation of Weighted Matching from [125]

Input: Graph $(V, E = \bigcup_{i=1}^{t} E_i)$
Output: Matching
for $i = t$ **to** 1 **do**
Find a maximal matching M_i in E_i.
Remove all edges e from E_j with $1 \leq j \leq t$ where $e \in M_i$ or e shares a node with an edge in M_i.
end for
return $\bigcup_{i=1}^{t} M_i$

next we call i a good rank if $a \geq T \cdot b$ and a significant rank if additionally the matching in the subgraph for i is larger than c times the matching in the subgraph for $i+1$. In the end, our estimator is the sum over the contributions of all significant ranks.

Algorithm and Analysis

For the analysis, we use a result on parallel algorithms by Uehara and Chen [125]. We show that the weight outputted by our algorithm is close to the weight of the matching computed by their algorithm, implying an approximation to the maximum weight.

Consider a graph $G = (V, E, w_G)$ with arbitrary edge weights $w_G(e) \in \mathbb{R}^+$. We start by describing the parallel algorithm by Uehara and Chen [125] (see also Algorithm 12). Let $\gamma > 1$ and $k > 0$ be constant. We partition the edge set by t ranks where all edges e in rank $i \in \{1, \ldots, t\}$ have a weight $w_G(e) \in (\gamma^{i-1} \cdot \frac{w_{max}}{kN}, \gamma^i \cdot \frac{w_{max}}{kN}]$ where w_{max} is the maximal weight in G. Let $G' = (V, E, w_{G'})$ be equal to G but each edge e in rank i has weight $r_i := \gamma^i$ for all $i = 1, \ldots, t$. Starting with $i = t$, we compute an unweighted maximal matching M_i considering only edges in rank i (in G') and remove all edges incident to a matched node. Continue with $i - 1$. The weight of the matching $M := \bigcup M_i$ is $w_{G'}(M) = \sum_{i=1}^{t} r_i \cdot |M_i|$ and satisfies $w_G(M^*) \geq w_{G'}(M) \geq \frac{1}{2\gamma} \cdot w_G(M^*)$ where M^* is an optimal weighted matching in G.

In order to adapt this idea to our setting, we need to work out the key properties of the partitioning and how we can implement it in a stream. The first problem is that we cannot know w_{max} in a stream a priori and in a dynamic stream even maintaining w_{max} is difficult. But we need to know the partition an inserted edge belongs to which is not possible if we do not have w_{max}. Recalling the partitioning of Uehara and Chen, we disregard all edges with weight smaller than $\frac{w_{max}}{kN}$ which is possible because the contribution of these edges is at most $\frac{N}{2} \cdot \frac{w_{max}}{kN} = \frac{w_{max}}{2k} \leq \frac{OPT}{2k}$ where OPT is the weight of an optimal weighted matching. Thus, they only consider edges with larger weights and it is also possible to partition the set of edges in a logarithmic number of sets. But how do we choose the partitioning when we do not know the value of w_{max}? Now, we use the properties that edge weights within a single partition set are similar and that $\frac{1}{\gamma} \leq \frac{w(e)}{w(e')} \leq \gamma$ for two edges $e \in E_i$ and $e' \in E_{i-1}$ with $i \in \{2, \ldots, t\}$. These properties are sufficient to get a good approximation on the optimal

weighted matching which we show in the next lemma. The proof is essentially the same as in [125].

Lemma 6.2.1. *Let $G = (V, E, w)$ be a weighted graph and $\varepsilon > 0$ be an approximation parameter. If a partitioning $E_1, \ldots, E_t$ of E and a weight function $w' : E \to \mathbb{R}$ satisfy*

$$\frac{1}{1+\varepsilon} \leq \frac{w'(e)}{w(e)} \leq 1 \text{ for all } e \in E \quad \text{and} \quad \frac{w(e_1)}{w(e_2)} \leq 1+\varepsilon \quad \text{and} \quad w(e) < w(e')$$

for all choices of edges $e_1, e_2 \in E_i$ and $e \in E_i, e' \in E_j$ with $i < j$ and $i, j \in \{1, \ldots, t\}$ then Algorithm 12 returns a matching $M = \bigcup_{i=1}^{t} M_i$ with

$$\frac{1}{2(1+\varepsilon)^2} \cdot w(M^*) \leq w'(M) \leq w(M^*)$$

where M^* is an optimal weighted matching in G.

Proof. The first property $\frac{1}{1+\varepsilon} \leq \frac{w'(e)}{w(e)} \leq 1$ for all $e \in E$ implies that $\frac{w(S)}{1+\varepsilon} \leq w'(S) \leq w(S)$ for every set of edges $S \subseteq E$. Thus, it suffices to show that $\frac{1}{2(1+\varepsilon)} \cdot w(M^*) \leq w(M) \leq w(M^*)$. Since M^* is an optimal weighted matching, it is clear that $w(M) \leq w(M^*)$. For the lower bound, we distribute the weight of the edges from the optimal solution to edges in M. Let $e \in M^*$ and $i \in \{1, \ldots, t\}$ such that $e \in E_i$. We consider the following cases:

1. $e \in M_i$: We charge the weight $w(e)$ to the edge itself.
2. $e \notin M_i$ but at least one node incident to e is matched by an edge in M_i: Let $e' \in M_i$ be an edge sharing a node with e. Distribute the weight $w(e)$ to e'.
3. $e \notin M_i$ and there is no edge in M_i sharing a node with e: By Algorithm 12, there has to be an edge $e' \in M_j$ with $j > i$ which shares a node with e. We distribute the weight $w(e)$ to e'.

Since M^* is a matching, there can only be at most two edges from M^* distributing their weights to the same edge in M. We know that $\frac{w(e)}{w(e')} \leq 1+\varepsilon$ for all choices of two edges $e, e' \in E_i$ with $i \in \{1, \ldots, t\}$ which means that in case 2 we have $w(e) \leq (1+\varepsilon) \cdot w(e')$. In case 3 it holds $w(e) < w(e')$. Thus, the weight distributed to an edge e' in M is at most $2(1+\varepsilon)w(e')$. This implies that $w(M^*) = \sum_{e \in M^*} w(e) \leq \sum_{e' \in M} 2(1+\varepsilon) \cdot w(e') = 2(1+\varepsilon) \cdot w(M)$ which concludes the proof. □

Using Lemma 6.2.1, we can partition the edge set in a stream in an almost oblivious manner: Let $(e_0, w(e_0))$ be the first inserted edge. Then an edge e belongs to E_i iff $2^{i-1} \cdot w(e_0) < w(e) \leq 2^i \cdot w(e_0)$ for some $i \in \mathbb{Z}$. We can assume that the weights are greater than 0. Then the number of sets is $\mathcal{O}\left(\log \frac{w_{max}}{w_{min}}\right)$. For the sake of simplicity, we will assume that the edge weights are in the interval $[1, W]$ for some $W \in \mathbb{N}$. Thus the number of sets can be bounded by

Algorithm 13 Weighted Matching Approximation

Input: Graph $G = (V, \bigcup_{i=0}^{t} E_i)$ with weights r_i for edges in E_i, Parameters $T, c > 0$
Output: Estimator of the weighted matching

```
for i = t to 0 do
    Ŝ_i = R̂_i = 0
end for
weight = 0
last = t
R̂_t = Ŝ_t = Unweighted Matching Estimation(V, E_t)
for i = t − 1 to 0 do
    Ŝ_i = Unweighted Matching Estimation(V, ⋃_{j=i}^{t} E_j)
    if Ŝ_i > Ŝ_last · T then                          ▷ Add current index i to I_good
        if Ŝ_i − Ŝ_last ≥ c · R̂_last then             ▷ Add current index i to I_sign
            R̂_i = Ŝ_i − Ŝ_last
            last = i
        end if
    else
        Ŝ_i = 0
    end if
end for
for i = t to 0 do
    weight = weight + r_i · R̂_i
end for
return (2/5) · weight
```

$\mathcal{O}(\log W)$. Note that we are still not able to discard the edges with weights smaller than $\frac{w_{max}}{kN}$ since we do not know w_{max} beforehand. However, typically we can assume that $W = \operatorname{poly} N$ because otherwise the representation size of a single weight is too large.

We now introduce a bit of notation we will use in the algorithm and throughout the proof. As before, we partition the edge set $E = \bigcup_{i=0}^{t} E_i$ into $t + 1 = O(\log W)$ ranks where the set E_i contains all edges e with weight $w(e) \in [2^i, 2^{i+1})$. W.l.o.g. we assume $E_t \neq \emptyset$ (otherwise let t be the largest rank with $E_t \neq \emptyset$). Let $G' = (V, E, w')$ be equal to G but each edge $e \in E_i$ has weight $w'(e) = r_i := 2^i$ for all $i = 0, \ldots, t$. Let $M = \bigcup_{i=0}^{t} M_i$ be the matching computed by Algorithm 12 and S be a $(t+1)$-dimensional vector with $S_i = \sum_{j=i}^{t} |M_i|$.

Algorithm 13 now proceeds as follows: For every $i \in \{0, \ldots t\}$ the size of a maximum matching in $(V, \bigcup_{j=i}^{t} E_j)$ and S_i differ by only a constant factor. Conceptually, we set our estimator $\widehat{S_i}$ of S_i to be the approximation of the size of the maximum matching of $(V, \bigcup_{j=i}^{t} E_i)$ and $\widehat{R_i} = \widehat{S_i} - \widehat{S_{i+1}}$ is the estimator of the contribution of the edges in E_i to the weight of an optimal weighted matching. As we know, the estimator $\widehat{R_i}$ is crude and generally not a good approximation to $|M_i|$. What helps us is that if the edges M_i have a significant contribution to $w(M)$, then $|M_i| \gg \sum_{j=i+1}^{t} |M_j| = S_{i+1}$. In order to detect whether the matching M_i has a significant contribution to the objective value, we introduce two parameters T and

c. The first matching M_t is always significant (and the simplest to approximate by setting $\widehat{R_t} = \widehat{S_t}$). For all subsequent matchings where $i < t$, let M_j be the most recent matching which we deemed to be significant. We require $\widehat{S_i} \geq T \cdot \widehat{S_j}$ and $\widehat{R_i} \geq c \cdot \widehat{R_j}$. If both criteria are satisfied, we use the estimator $\widehat{R_i} = \widehat{S_i} - \widehat{S_j}$ and set i to be the now most recent, significant matching, otherwise we set $\widehat{R_i} = 0$. The final estimator of the weight is $\frac{2}{5}\sum_{i=0}^{t} r_i \cdot \widehat{R_i}$. The next definition gives a more detailed description of the two sets of ranks which are important for the analysis.

Definition 6.2.2 (Good and Significant Ranks)**.** Let $\widehat{S}$ and $\widehat{R}$ be the vectors at the end of Algorithm 13. An index i is called to be a *good rank* if $\widehat{S_i} \neq 0$ and i is a *significant rank* if $\widehat{R_i} \neq 0$. We denote the set of good ranks by I_{good} and the set of significant ranks by I_{sign}, i. e.,

$$I_{good} := \left(i \in \{0, \dots t\} \,\middle|\, \widehat{S_i} \neq 0\right) \text{ and}$$

$$I_{sign} := \left(i \in \{0, \dots t\} \,\middle|\, \widehat{R_i} \neq 0\right).$$

We define I_{good} and I_{sign} to be in descending order and we will refer to the ℓ-th element of I_{good} and I_{sign} by $I_{good}(\ell)$ and $I_{sign}(\ell)$, respectively. That means

$$I_{good}(1) > I_{good}(2) > \ldots > I_{good}(|I_{good}|) \text{ and}$$

$$I_{sign}(1) > I_{sign}(2) > \ldots > I_{sign}(|I_{sign}|).$$

We slightly abuse the notation and set $I_{sign}(|I_{sign}| + 1) = 0$. Let $D_1 := |M_t|$ and for $\ell \in \{2, \dots, |I_{sign}|\}$ we define the sum of the matching sizes between two significant ranks $I_{sign}(\ell)$ and $I_{sign}(\ell - 1)$ where the smaller significant rank is included by

$$D_\ell := \sum_{i=I_{sign}(\ell)}^{I_{sign}(\ell-1)-1} |M_i|.$$

In the following, we subscript indices by s for significant ranks and by g for good ranks for the sake of readability. Looking at Algorithm 13 we can prove some simple properties of I_{good} and I_{sign}.

Lemma 6.2.3. *Let I_{good} and I_{sign} be defined as in Definition 6.2.2. The parameter T and c are from Algorithm 13. Then*

1. *$I_{good}(1) = I_{sign}(1) = t$ and $I_{sign} \subseteq I_{good}$.*
2. *For every good rank $i_g \in I_{good}$ there is an $\ell \in \{0, \dots, |I_{sign}|\}$ such that $I_{sign}(\ell) > i_g \geq I_{sign}(\ell + 1)$ and $\widehat{S_{i_g}} > T \cdot \widehat{S_{I_{sign}(\ell)}}$.*
3. *For any $i_s \in I_{sign}$ and $i'_s \in I_{sign}$ with $i'_s < i_s$ it is $\widehat{R_{i'_s}} \geq c \cdot \widehat{R_{i_s}}$.*

Proof.

1. It is clear that $I_{sign} \subseteq I_{good}$. Since we assumed that $E_t \neq \emptyset$, there is a nonempty matching in E_t which means that $\widehat{S_t} = \widehat{R_t} > 0$.

2. Let ℓ be the position of *last* in I_{sign} where *last* is the value of the variable in Algorithm 13 during the iteration $i = i_g$. Then $I_{sign}(\ell) > i_g \geq I_{sign}(\ell+1)$ (recall that we defined $I_{sign}(|I_{sign}|+1) = 0$). Since i_g is good, it is $\widehat{S}_{i_g} > T \cdot \widehat{S_{last}} = \widehat{S_{I_{sign}(\ell)}}$.

3. For every $i_s \in I_{sign}$ we have $\widehat{R_{i_s}} \geq c \cdot \widehat{R_{last}}$ where *last* is the value of the variable in Algorithm 13 in iteration $i = i_s$. By definition it is $last \in I_{sign}$ and $last > i_s$. Therefore, it holds $\widehat{R_{I_{sign}(\ell+1)}} > c \cdot \widehat{R_{I_{sign}(\ell)}}$ for every $\ell \in \{0, \ldots, |I_{sign}| - 1\}$ which implies the statement.

□

Now, we have the necessary notations and properties of good and significant ranks to prove our main theorem.

Theorem 6.2.4. *Let $G = (V, E, w)$ be a weighted graph where the weights are from $[1, W]$. Let $\mathcal{A}$ be a randomized algorithm that returns an λ-estimator $\widehat{M}$ for the size of a maximum matching M of a graph with $1/\lambda \cdot |M| \leq \widehat{M} \leq |M|$ with probability at least $1-\delta$ and that needs space S. If we partition the edge set into sets $E_0, \ldots, E_t$ with $t = \lfloor \log W \rfloor$ where E_i consists of all edges with weight in $[2^i, 2^{i+1})$, set $r_i = 2^i$, and use $\mathcal{A}$ as the unweighted matching estimator in Algorithm 13, then with $T = 8\lambda^2 - 2\lambda$ and $c = \frac{2}{5} \cdot T + 5\lambda$ the algorithm returns an $\mathcal{O}(\lambda^4)$-estimator $\widehat{OPT}$ for the weight of the maximum weighted matching with probability at least $1 - (t+1) \cdot \delta$ using $\mathcal{O}(S \cdot t)$ space. This means there is a constant d such that*

$$\frac{1}{d\lambda^4} \cdot w(M^*) \leq \widehat{OPT} \leq w(M^*)$$

where M^* is an optimal weighted matching.

Proof. The probability that all $t+1$ estimations returned by algorithm A are within the approximation bounds is at least $1-(t+1)\cdot\delta$. Conditioning on this event, we now prove that we get an $\mathcal{O}(\lambda^4)$-estimator. The estimator returned by Algorithm 13 without the 2/5 factor can be written as $\sum_{\ell=1}^{|I_{sign}|} r_{I_{sign}(\ell)} \cdot \widehat{R_{I_{sign}(\ell)}}$. Let $M = \bigcup_{i=0}^{t} M_i$ be the result of Algorithm 12 on the input $(V, \bigcup E_i)$. Since $1/2 \leq 2^i/w(e) \leq 1$ and $w(e)/w(e') \leq 2$ for $e, e' \in E_i$ and $i \in \{0, \ldots t\}$ by the definition of E_i, we can use Lemma 6.2.1 with $\varepsilon = 1$ to get

$$\frac{1}{8} \cdot w(M^*) \leq \sum_{i=0}^{t} r_i |M_i| \leq w(M^*). \tag{6.1}$$

Thus, it is sufficient to show that $\sum_{\ell=1}^{|I_{sign}|} r_{I_{sign}(\ell)} \cdot \widehat{R_{I_{sign}(\ell)}}$ is a good estimator for $\sum_{i=0}^{t} r_i |M_i|$. We divide this into two subproblems:

1. **(Estimation)** $\widehat{R_{I_{sign}(\ell)}}$ is a good estimator for D_ℓ.
2. **(Charging)** We show that $\sum_{i=0}^{t} r_i|M_i|$ can be estimated by $\sum_{\ell=1}^{|I_{sign}|} r_{I_{sign}(\ell)} D_\ell$.

Recall that $D_\ell = \sum_{i=I_{sign}(\ell)}^{I_{sign}(\ell-1)-1} |M_i|$.

(1) Estimation of D_ℓ

Recall that we defined $S_i = \sum_{j=i}^{t} |M_i|$. Since $\bigcup_{j=i}^{t} M_j$ is a maximal matching in $\bigcup_{j=i}^{t} E_j$, $\widehat{S_i}$ is a good estimator for S_i.

Lemma 6.2.5. *For all $i \in \{0, \dots, t\}$ we have $\frac{1}{\lambda} \cdot S_i \leq \widehat{S_i} \leq 2 \cdot S_i$.*

Proof. Let F_j be the set of unmatched nodes after the iteration j of Algorithm 12. Let M^* be a maximum matching in $(V, \bigcup_{j=i}^{t} E_j)$. M_j is a maximal matching of $(V, E_j(F_j))$ and therefore $\bigcup_{j=i}^{t} M_j$ is a maximal matching of $(V, \bigcup_{j=i}^{t} E_j)$. This allows us to apply the bounds of the λ-approximate estimation algorithm:

$$\frac{1}{\lambda} \cdot S_i = \frac{1}{\lambda} \cdot \sum_{j=i}^{t} |M_j| \leq \frac{1}{\lambda} \cdot |M^*| \leq \widehat{S_i} \leq |M^*| \leq 2 \cdot \sum_{j=i}^{t} |M_j| = 2 \cdot S_i.$$

□

Next, we show that for an index $i_g \in I_{good}$ the difference $\widehat{S_{i_g}} - \widehat{S_{I_{sign}(\ell)}}$ to the last significant rank is a good estimator for $\sum_{i=i_g}^{I_{sign}(\ell)-1} |M_i|$.

Lemma 6.2.6. *For all $i_g \in I_{good}$ with $I_{sign}(\ell+1) \leq i_g < I_{sign}(\ell)$ for some $\ell \in \{1, \dots, |I_{sign}|\}$ and $T = 8\lambda^2 - 2\lambda$ it is*

$$\frac{1}{2\lambda} \cdot \sum_{i=i_g}^{I_{sign}(\ell)-1} |M_i| < \widehat{S_{i_g}} - \widehat{S_{I_{sign}(\ell)}} < \frac{5}{2} \cdot \sum_{i=i_g}^{I_{sign}(\ell)-1} |M_i|$$

and $\frac{1}{\lambda}|M_t| \leq \widehat{S_t} = \widehat{R_t} \leq 2|M_t|$.

Proof. For all $i_g \in I_{good}$ with $I_{sign}(\ell+1) \leq i_g < I_{sign}(\ell)$ we have

$$\begin{aligned}
\sum_{i=i_g}^{I_{sign}(\ell)-1} |M_i| &= S_{i_g} - S_{I_{sign}(\ell)} \underset{\text{Lem. 6.2.5}}{\geq} \frac{1}{2} \cdot \widehat{S_{i_g}} - \lambda \cdot \widehat{S_{I_{sign}(\ell)}} \\
&\underset{\text{Lem. 6.2.3 (2)}}{>} \frac{T}{2} \cdot \widehat{S_{I_{sign}(\ell)}} - \lambda \cdot \widehat{S_{I_{sign}(\ell)}} \underset{\text{Lem. 6.2.5}}{\geq} \left(\frac{T}{2} - \lambda\right) \cdot \frac{1}{\lambda} \cdot S_{I_{sign}(\ell)} \\
&= \frac{T - 2\lambda}{2\lambda} \cdot S_{I_{sign}(\ell)}. \qquad (6.2)
\end{aligned}$$

Setting $T = 8\lambda^2 - 2\lambda$, we then obtain the following upper and lower bounds

$$\begin{aligned}
\widehat{S_{i_g}} - \widehat{S_{I_{sign}(\ell)}} &\underset{\text{Lem. 6.2.5}}{\geq} \frac{1}{\lambda} \cdot S_{i_g} - 2 \cdot S_{I_{sign}(\ell)} = \frac{1}{\lambda} \sum_{i=i_g}^{I_{sign}(\ell)-1} |M_i| - \left(2 - \frac{1}{\lambda}\right) \cdot S_{I_{sign}(\ell)} \\
&\underset{\text{Eq. (6.2)}}{>} \frac{1}{\lambda} \sum_{i=i_g}^{I_{sign}(\ell)-1} |M_i| - \left(2 - \frac{1}{\lambda}\right) \cdot \frac{2\lambda}{T - 2\lambda} \cdot \sum_{i=i_g}^{I_{sign}(\ell)-1} |M_i| \\
&= \frac{1}{\lambda} \sum_{i=i_g}^{I_{sign}(\ell)-1} |M_i| - \left(\frac{2\lambda - 1}{\lambda}\right) \cdot \frac{2\lambda}{8\lambda^2 - 4\lambda} \cdot \sum_{i=i_g}^{I_{sign}(\ell)-1} |M_i| \\
&= \left(\frac{1}{\lambda} - \frac{2}{4\lambda}\right) \sum_{i=i_g}^{I_{sign}(\ell)-1} |M_i| \\
&= \frac{1}{2\lambda} \cdot \sum_{i=i_g}^{I_{sign}(\ell)-1} |M_i|
\end{aligned}$$

and

$$\begin{aligned}
\widehat{S_{i_g}} - \widehat{S_{I_{sign}(\ell)}} &\underset{\text{Lem. 6.2.5}}{\leq} 2 \cdot S_{i_g} - \frac{1}{\lambda} \cdot S_{I_{sign}(\ell)} = 2 \cdot \sum_{i=i_g}^{I_{sign}(\ell)-1} |M_i| + \left(2 - \frac{1}{\lambda}\right) \cdot S_{I_{sign}(\ell)} \\
&\underset{\text{Eq. (6.2)}}{<} 2 \sum_{i=i_g}^{I_{sign}(\ell)-1} |M_i| + \left(2 - \frac{1}{\lambda}\right) \cdot \frac{2\lambda}{8\lambda^2 - 4\lambda} \cdot \sum_{i=i_g}^{I_{sign}(\ell)-1} |M_i| \\
&= \left(2 + \frac{2}{4\lambda}\right) \sum_{i=i_g}^{I_{sign}(\ell-1)-1} |M_i| \\
&\leq \frac{5}{2} \cdot \sum_{i=i_g}^{I_{sign}(\ell-1)-1} |M_i|,
\end{aligned}$$

where we used $\lambda \geq 1$ in the last inequality. Since $|M_t| = S_t$ and $\widehat{R_t} = \widehat{S_t}$, the last statement follows directly from Lemma 6.2.5. □

From Lemma 6.2.3 (1) we know that $I_{sign} \subseteq I_{good}$ which together with the last Lemma 6.2.6 implies that $\widehat{R_{I_{sign}(\ell)}}$ is a good estimator for D_ℓ.

Corollary 6.2.7. *For $\ell \in \{1, \dots, |I_{sign}|\}$ it is*

$$\frac{1}{2\lambda} \cdot D_\ell \leq \widehat{R_{I_{sign}(\ell)}} \leq \frac{5}{2} \cdot D_\ell.$$

Furthermore, if $c > 5\lambda$ then the values of the D_ℓ are exponentially increasing:

$$D_1 \leq \frac{5\lambda}{c} D_2 \leq \ldots \leq \left(\frac{5\lambda}{c}\right)^{|I_{sign}|-2} D_{|I_{sign}|-1}.$$

Proof. Recall that for $\ell \in \{2, \dots, |I_{sign}|\}$ we defined $D_\ell = \sum_{i=I_{sign}(\ell)}^{I_{sign}(\ell-1)-1} |M_i|$. For $\ell = 1$ the value of $\widehat{R_{I_{sign}(1)}} = \widehat{S_t}$ is a good estimator for the size of the matching M_t (which is equal to D_1 by Definition 6.2.2) due to Lemma 6.2.5. Since for $\ell \in \{2, \dots, |I_{sign}|\}$ it is $\widehat{R_{I_{sign}(\ell)}} = \widehat{S_{I_{sign}(\ell)}} - \widehat{S_{I_{sign}(\ell-1)}}$ and $I_{sign} \subseteq I_{good}$, the first statement is a direct implication of Lemma 6.2.6 by setting $i_g = I_{sign}(\ell)$.

For three adjoining significant ranks $I_{sign}(\ell+1), I_{sign}(\ell), I_{sign}(\ell-1)$ with $\ell \in \{2, \dots, |I_{sign}| - 1\}$, we have

$$\begin{aligned}
\frac{1}{2\lambda} \cdot D_\ell = \frac{1}{2\lambda} \sum_{i=I_{sign}(\ell)}^{I_{sign}(\ell-1)-1} |M_i| &\underset{\text{Lem. 6.2.6}}{<} \widehat{S_{I_{sign}(\ell)}} - \widehat{S_{I_{sign}(\ell-1)}} = \widehat{R_{I_{sign}(\ell)}} \\
&\underset{\text{Lem. 6.2.3 (3)}}{<} \frac{1}{c} \cdot \widehat{R_{I_{sign}(\ell+1)}} = \frac{1}{c} \cdot \left(\widehat{S_{I_{sign}(\ell+1)}} - \widehat{S_{I_{sign}(\ell)}}\right) \\
&\underset{\text{Lem. 6.2.6}}{<} \frac{5}{2c} \sum_{i=I_{sign}(\ell+1)}^{I_{sign}(\ell)-1} |M_i| = \frac{5}{2c} \cdot D_{\ell+1}.
\end{aligned}$$

Since $D_1 = |M_t|$ and $\widehat{R_{I_{sign}(1)}} = \widehat{R_t} = \widehat{S_t}$, we also have

$$\frac{1}{2\lambda} \cdot D_1 \leq \frac{1}{\lambda} \cdot D_1 \underset{\text{Lem. 6.2.5}}{\leq} \widehat{R_t} \underset{\text{Lem. 6.2.3 (3)}}{\leq} \frac{1}{c} \cdot \widehat{R_{I_{sign}(2)}} \underset{\text{Lem. 6.2.6}}{\leq} \frac{5}{2c} \sum_{i=I_{sign}(2)}^{I_{sign}(1)-1} |M_i| = \frac{5}{2c} \cdot D_2.$$

Thus, for $c > 5\lambda$ the values of the D_ℓ are exponentially increasing:

$$D_1 \leq \frac{5\lambda}{c} D_2 \leq \ldots \leq \left(\frac{5\lambda}{c}\right)^{|I_{sign}|-2} D_{|I_{sign}|-1}.$$

□

(2) The Charging Argument

We show that the sum of the matching sizes between two significant ranks $I_{sign}(\ell+1)$ and $I_{sign}(\ell)$ is bounded by $\mathcal{O}(\lambda \cdot T \cdot D_\ell) = \mathcal{O}\left(\lambda \cdot T \cdot \sum_{i=I_{sign}(\ell)}^{I_{sign}(\ell-1)+1} |M_i|\right)$.

Lemma 6.2.8. *For $c = \frac{2}{5} \cdot T + 5\lambda$ and $\ell \in \{1, \dots, |I_{sign}| - 1\}$ it is*

$$\sum_{i=I_{sign}(\ell+1)+1}^{I_{sign}(\ell)-1} |M_i| \leq (2\lambda \cdot T + 25\lambda^2) \cdot D_\ell$$

and if $0 \notin I_{sign}$

$$\sum_{i=0}^{I_{sign}(|I_{sign}|)-1} |M_i| \leq (2\lambda \cdot T + 25\lambda^2) \cdot D_{|I_{sign}|}.$$

Proof. For the proof of the first inequality, let $i_g \in I_{good}$ be minimal such that $I_{sign}(\ell+1) < i_g < I_{sign}(\ell)$ for $\ell \in \{1, \dots, |I_{sign}| - 1\}$. If such a good rank does not exist, set $i_g = -1$. We distinguish between two cases. Note that $c = \frac{2}{5} \cdot T + 5\lambda > 5\lambda$ what we need to apply Corollary 6.2.7.

Case 1: $i_g = I_{sign}(\ell+1) + 1$**.** For the sake of simplicity, we abuse the notation and set $\widehat{S_{I_{sign}(0)}} = 0$ such that $\widehat{R_{I_{sign}(\ell)}} = \widehat{S_{I_{sign}(\ell)}} - \widehat{S_{I_{sign}(\ell-1)}}$ also holds for $\ell = 1$. Using Lemma 6.2.6 we have

$$\begin{aligned}
\frac{1}{2\lambda} \cdot \sum_{i=I_{sign}(\ell+1)+1}^{I_{sign}(\ell)-1} |M_i| \quad &= \quad \frac{1}{2\lambda} \cdot \sum_{i=i_g}^{I_{sign}(\ell)-1} |M_i| \underset{\text{Lem. 6.2.6}}{\leq} \widehat{S_{i_g}} - \widehat{S_{I_{sign}(\ell)}} \\
&\underset{i_g \notin I_{sign}}{\leq} \quad c \cdot \widehat{R_{I_{sign}(\ell)}} = c \cdot \left(\widehat{S_{I_{sign}(\ell)}} - \widehat{S_{I_{sign}(\ell-1)}} \right) \\
&\underset{\text{Lem. 6.2.6}}{\leq} \quad \frac{5}{2} \cdot c \cdot \sum_{i=I_{sign}(\ell)}^{I_{sign}(\ell-1)-1} |M_i| = \frac{5}{2} \cdot c \cdot D_\ell \\
\Leftrightarrow \sum_{i=I_{sign}(\ell+1)+1}^{I_{sign}(\ell)-1} |M_i| \quad &< \quad 5\lambda \cdot c \cdot D_\ell
\end{aligned} \tag{6.3}$$

Case 2: $i_g \neq I_{sign}(\ell+1) + 1$**.** In this case $\widehat{S_{I_{sign}(\ell+1)+1}} \leq T \cdot \widehat{S_{I_{sign}(\ell)}}$ by the definition of good ranks. Thus

$$\begin{aligned}
\frac{1}{\lambda} \cdot \sum_{i=I_{sign}(\ell+1)+1}^{I_{sign}(\ell)-1} |M_i| \quad &\leq \quad \frac{1}{\lambda} \cdot S_{I_{sign}(\ell+1)+1} \underset{\text{Lem. 6.2.5}}{\leq} \widehat{S_{I_{sign}(\ell+1)+1}} \\
&\leq \quad T \cdot \widehat{S_{I_{sign}(\ell)}} \underset{\text{Lem. 6.2.5}}{\leq} 2 \cdot T \cdot S_{I_{sign}(\ell)} = 2 \cdot T \cdot \sum_{i=1}^{\ell} D_i \\
&\underset{\text{Cor. 6.2.7}}{\leq} \quad 2 \cdot T \cdot D_\ell \cdot \sum_{i=1}^{\ell} \left(\frac{5\lambda}{c} \right)^{i-1} \leq 2 \cdot T \cdot D_\ell \cdot \frac{1}{1 - \frac{5\lambda}{c}} \\
\Leftrightarrow \sum_{i=I_{sign}(\ell+1)+1}^{I_{sign}(\ell)-1} |M_i| \quad &\leq \quad \frac{2\lambda \cdot T}{1 - \frac{5\lambda}{c}} \cdot D_\ell.
\end{aligned} \tag{6.4}$$

Combining the inequalities 6.3 and 6.4, we have $\sum_{i=I_{sign}(\ell+1)+1}^{I_{sign}(\ell)-1} |M_i| \leq \max\left\{5\lambda \cdot c, \frac{2\lambda \cdot T}{1-\frac{5\lambda}{c}}\right\} \cdot D_\ell$ which simplifies to

$$\sum_{i=I_{sign}(\ell+1)+1}^{I_{sign}(\ell)-1} |M_i| \leq (2\lambda \cdot T + 25\lambda^2) \cdot D_\ell \qquad \text{for } \ell \in \{1, \ldots, |I_{sign}| - 1\} \tag{6.5}$$

because $5\lambda \cdot c = \frac{2\lambda \cdot T}{1-\frac{5\lambda}{c}}$ if and only if $c = \frac{2}{5} \cdot T + 5\lambda$. If $0 \notin I_{sign}$ we have to do the same arguments to bound $\sum_{i=0}^{I_{sign}(|I_{sign}|)-1} |M_i|$ by $(2\lambda \cdot T + 25\lambda^2) \cdot D_{|I_{sign}|}$. Let $i_g \in I_{good}$ be minimal such that $0 \leq i_g < I_{sign}(|I_{sign}|)$. Again, we distinguish between two cases.

Case 1: $i_g = 0$**.** Using Lemma 6.2.6 we have

$$\begin{aligned}
\frac{1}{2\lambda} \cdot \sum_{i=0}^{I_{sign}(|I_{sign}|)-1} |M_i| &\underset{\text{Lem. 6.2.6}}{<} \widehat{S_0} - \widehat{S_{I_{sign}(|I_{sign}|)}} \\
&\underset{0 \notin I_{sign}}{<} c \cdot \widehat{R_{I_{sign}(|I_{sign}|)}} = c \cdot \left(\widehat{S_{I_{sign}(|I_{sign}|)}} - \widehat{S_{I_{sign}(|I_{sign}|-1)}}\right) \\
&\underset{\text{Lem. 6.2.6}}{<} \frac{5}{2} \cdot c \cdot \sum_{i=I_{sign}(|I_{sign}|)}^{I_{sign}(|I_{sign}|-1)-1} |M_i| = \frac{5}{2} \cdot c \cdot D_{|I_{sign}|} \\
\Leftrightarrow \sum_{i=0}^{I_{sign}(|I_{sign}|)-1} |M_i| &< 5\lambda \cdot c \cdot D_{|I_{sign}|}
\end{aligned}$$

Case 2: $i_g \neq 0$**.** In this case $\widehat{S_0} \leq T \cdot \widehat{S_{I_{sign}(|I_{sign}|)}}$ by the definition if good ranks. Thus

$$\begin{aligned}
\frac{1}{\lambda} \cdot \sum_{i=0}^{I_{sign}(|I_{sign}|)-1} |M_i| &\leq \frac{1}{\lambda} \cdot S_0 \underset{\text{Lem. 6.2.5}}{\leq} \widehat{S_0} \leq T \cdot \widehat{S_{I_{sign}(|I_{sign}|)}} \\
&\underset{\text{Lem. 6.2.5}}{\leq} 2 \cdot T \cdot S_{I_{sign}(|I_{sign}|)} = 2 \cdot T \cdot \sum_{i=1}^{|I_{sign}|} D_i \\
&\underset{\text{Cor. 6.2.7}}{\leq} 2 \cdot T \cdot D_{|I_{sign}|} \cdot \sum_{i=1}^{|I_{sign}|} \left(\frac{5\lambda}{c}\right)^{i-1} \leq 2 \cdot T \cdot D_{|I_{sign}|} \cdot \frac{1}{1-\frac{5\lambda}{c}} \\
\Leftrightarrow \sum_{i=0}^{I_{sign}(|I_{sign}|)-1} |M_i| &\leq \frac{2\lambda \cdot T}{1-\frac{5\lambda}{c}} \cdot D_{|I_{sign}|}.
\end{aligned}$$

With the same $c = \frac{2}{5} \cdot T + 5\lambda$ as before we have

$$\sum_{i=0}^{I_{sign}(|I_{sign}|)-1} |M_i| \leq (2\lambda \cdot T + 25\lambda^2) \cdot D_{|I_{sign}|}.$$

□

Now, we can show that $w(M)$ is bounded in terms of $\sum_{\ell=1}^{|I_{sign}|} r_{I_{sign}(\ell)} \cdot D_\ell$. A lower bound to $w(M) = \sum_{i=0}^{t} r_i \cdot |M_i|$ is given by

$$\begin{aligned}\sum_{i=0}^{t} r_i \cdot |M_i| &\geq r_t \cdot M_t + \sum_{\ell=2}^{|I_{sign}|} r_{I_{sign}(\ell)} \cdot \sum_{i=I_{sign}(\ell)}^{I_{sign}(\ell-1)-1} |M_i| + r_0 \cdot \sum_{i=0}^{I_{sign}(|I_{sign}|)-1} |M_i| \\ &\geq \sum_{\ell=1}^{|I_{sign}|} r_{I_{sign}(\ell)} \cdot D_\ell \end{aligned} \tag{6.6}$$

and for an upper bound

$$\begin{aligned}\sum_{i=0}^{t} r_i \cdot |M_i| &= \sum_{\ell=1}^{|I_{sign}|} r_{I_{sign}(\ell)} \cdot M_{I_{sign}(\ell)} + \sum_{\ell=1}^{|I_{sign}|-1} \sum_{i=I_{sign}(\ell+1)+1}^{I_{sign}(\ell)-1} r_i \cdot |M_i| \\ &\quad + \sum_{i=0}^{I_{sign}(|I_{sign}|)-1} r_i \cdot |M_i| \\ &\leq \sum_{\ell=1}^{|I_{sign}|} r_{I_{sign}(\ell)} \cdot D_\ell + \sum_{\ell=1}^{|I_{sign}|-1} \sum_{i=I_{sign}(\ell+1)+1}^{I_{sign}(\ell)-1} r_i \cdot |M_i| + \sum_{i=0}^{I_{sign}(|I_{sign}|)-1} r_i \cdot |M_i| \\ &\leq \sum_{\ell=1}^{|I_{sign}|} r_{I_{sign}(\ell)} \cdot D_\ell + \sum_{\ell=1}^{|I_{sign}|-1} r_{I_{sign}(\ell)} \cdot \sum_{i=I_{sign}(\ell+1)+1}^{I_{sign}(\ell)-1} |M_i| \\ &\quad + r_{I_{sign}(|I_{sign}|)} \cdot \sum_{i=0}^{I_{sign}(|I_{sign}|)-1} |M_i| \\ &\overset{\text{Lemma. 6.2.8}}{\leq} \sum_{\ell=1}^{|I_{sign}|} r_{I_{sign}(\ell)} \cdot D_\ell + \sum_{\ell=1}^{|I_{sign}|} r_{I_{sign}(\ell)} \cdot (2\lambda \cdot T + 25\lambda^2) \cdot D_\ell \\ &= (1 + 2\lambda \cdot T + 25\lambda^2) \cdot \sum_{\ell=1}^{|I_{sign}|} r_{I_{sign}(\ell)} \cdot D_\ell. \end{aligned} \tag{6.7}$$

Putting Everything Together

Using Corollary 6.2.7 we have $\frac{1}{2\lambda} \cdot D_\ell \leq \widehat{R_{I_{sign}(\ell)}} \leq \frac{5}{2} \cdot D_\ell$ for all $\ell \in \{1, \ldots, |I_{sign}|\}$ which with (6.6) and (6.7) gives

$$\begin{aligned}
\frac{1}{2\lambda \cdot (1 + 2\lambda \cdot T + 25\lambda^2)} \cdot w(M) &\underset{(6.7)}{\leq} \frac{1}{2\lambda} \sum_{\ell=1}^{|I_{sign}|} r_{I_{sign}(\ell)} \cdot D_\ell \\
&\underset{\text{Cor. } 6.2.7}{\leq} \sum_{\ell=1}^{|I_{sign}|} r_{I_{sign}(\ell)} \cdot \widehat{R_{I_{sign}(\ell)}} \\
&\underset{\text{Cor. } 6.2.7}{\leq} \frac{5}{2} \sum_{\ell=1}^{|I_{sign}|} r_{I_{sign}(\ell)} \cdot D_\ell \\
&\underset{(6.6)}{\leq} \frac{5}{2} \cdot w(M).
\end{aligned}$$

Recall that we set $T = 8\lambda^2 - 2\lambda$. Now, folding in the factor of $\frac{1}{8}$ from (6.1) and rescaling the estimator by $2/5$ gives an $\mathcal{O}(\lambda^4)$-estimation on the weight of an optimal weighted matching. □

6.3. Estimation of the Size of Unweighted Matchings

In this section, we give two estimation algorithms for the size of a maximum matching. First, we see that it is easy to estimate the matching size in trees. Second, we extend the result from [44] where the matching size of so-called bounded arboricity graphs is estimated for insertion-only streams to dynamic graph streams.

6.3.1. Estimating the Matching Size of Trees in Dynamic Streams

Let $T = (V, E)$ be a tree with at least 3 nodes and let h_T be the number of internal nodes, i. e., nodes with degree greater than 1. Esfandiari et al. [44] showed that the size of a maximum matching is between $h_T/2$ and h_T. Therefore, it suffices to estimate the number of internal nodes of a tree to approximate the maximum matching within $2 + \varepsilon$ factor. In order to estimate the matching size, we maintain an ℓ_0-estimator for the degree vector $d \in \mathbb{R}^N$ such that $d_v = deg(v) - 1$ holds at the end of the stream and with it $\ell_0(d) = h_T$. In other words, we initialize the vector by adding -1 to each entry and update the two corresponding entries when we get an edge deletion or insertion. Since the number of edges in a tree is $N - 1$, the preprocessing time can be amortized during the stream. Using Theorem 2.5.9, we can maintain the ℓ_0-estimator for d in $\mathcal{O}(\varepsilon^{-2} \log^2 N)$ space.

Theorem 6.3.1. *Let $T = (V, E)$ be a tree with at least 3 nodes and let $\varepsilon \in (0, 1)$. Then there is an algorithm that estimates the size of a maximum matching in T within a $(2 + \varepsilon)$-factor in the dynamic streaming model using 1-pass over the data and $\mathcal{O}(\varepsilon^{-2} \log^2 N)$ space.*

As in [44] this algorithm can be extended to forests with no isolated node.

6.3.2. Dynamic Streaming Algorithms for Estimating the Matching Size in Graphs with Bounded Arboricity

Our algorithm is based on the results from [44]. Since we need parametrized versions of their results, we summarize and rephrase the ideas and proofs in this section. Let $G = (V, E)$ be a graph. The arboricity $a(G)$ of G is a kind of density measure: The number of edges in every induced subgraph with s nodes in G is bounded by $s \cdot a(G)$. Formally, the arboricity $a(G)$ of G is defined by $a(G) = \max_{U \subseteq V} \left\lceil \frac{|E(U)|}{|U|-1} \right\rceil$. If μ_G is an upper bound on the average degree of every induced subgraph of G then $\mu_G \leq 2 \cdot a(G)$.

Definition 6.3.2 ([44])**.** A node $v \in V$ is *light* if $deg(v) \leq C$ with $C = \lceil \mu_G \rceil + 3$. Otherwise, v is *heavy*. An edge is *shallow* if and only if both of its endpoints are light. We denote by h_G the number of heavy nodes in G and by s_G the number of shallow edges in G, respectively.

Using the results from Czygrinow, Hanchowiak, and Szymanska [38] (and $C = 20a(G)/\varepsilon^2$) it is possible to get an $\mathcal{O}(a(G))$ approximation for the size of a maximum matching by just estimating h_G and s_G. Esfandiari et al. [44] improved the approximation factor to roughly $5 \cdot a(g)$.

Lemma 6.3.3 ([44])**.** *Let $G = (V, E)$ be a graph with maximum matching M^*. Then we have $\frac{\max\{h_G, s_G\}}{\eta} \leq |M^*| \leq h_G + s_G$ where $\eta = 1.25C + 0.75$ where C is at most $\lceil 2a(G) + 3 \rceil$.*

Estimating h_G and s_G is possible by random sampling: For heavy nodes, we randomly draw a large enough set of nodes and count the heavy nodes by maintaining their degree. Rescaling the counter gives a sufficiently good estimate, if h_G is large enough. For s_G we randomly draw nodes and maintain the induced subgraph. For each node contained in the subgraph it is straightforward to maintain the degree and thereby to decide whether or not a given edge from the subgraph is shallow. Then we can rescale the counted number of shallow edges which gives us an estimation on s_G if s_G is large enough. Dealing with small values of s_G and h_G, Esfandiari et al. additionally maintain a small maximal matching of size at most n^α with $\alpha < 1$. If the maintained matching exceeds this value then we know that either s_G or h_G is greater than $n^\alpha/2$ by Lemma 6.3.3 and the estimation of the parameters h_G and s_G will be sufficiently accurate. The main tool to extend this algorithm to dynamic graph streams is to estimate the size of a small matching by means of the Tutte matrix. But first, we restate the following three lemmas from [44] for arbitrary parameters and extend them to dynamic streams.

Lemma 6.3.4. *Let T be an integer and $\varepsilon \in (0,1)$. Then there exists a 1-pass algorithm for dynamic streams that outputs a value $\widehat{h}$ which is a $(1 \pm \varepsilon)$ estimation of h_G if $h_G \geq T$ and which is smaller than $3T$ otherwise. The algorithm needs $\mathcal{O}\left(\frac{\log^2 N}{\varepsilon^2} \cdot \frac{N}{T}\right)$ space and succeeds with high probability.*

Proof. The probability of sampling a heavy node is $\frac{h_G}{N}$. Hence, sampling a set of nodes S gives us $|S| \cdot \frac{h_G}{N}$ heavy nodes on expectation. Set $|S| = \frac{3 \log N}{\varepsilon^2} \frac{N}{T}$. For each node $v \in S$ we maintain its degree using $\mathcal{O}(\log N)$ space. We define the indicator variable X_v with $v \in S$ which is 1 if v is heavy and 0 otherwise. Then our estimator for h_G is $\hat{h} = \frac{N}{|S|} \sum X_v$ which is equal to h_G in expectation. First, assume $h_G \geq T$. Then using the Chernoff bound (Theorem A.2.3), we have

$$\begin{aligned}
\mathbf{Pr}\left[\hat{h} \geq (1+\varepsilon) \cdot \mathbf{E}\left[\hat{h}\right]\right] &= \mathbf{Pr}\left[\sum_{v \in S} X_v \geq (1+\varepsilon) \cdot \mathbf{E}\left[\sum_{v \in S} X_v\right]\right] \\
&\leq \exp\left(-\frac{3 \log N}{\varepsilon^2} \frac{N}{T} \cdot \frac{h_G}{N} \cdot \frac{\varepsilon^2}{3}\right) \leq \frac{1}{N}.
\end{aligned}$$

The same bound also holds for $\mathbf{Pr}\left[\hat{h} \leq (1-\varepsilon) \cdot \mathbf{E}\left[\hat{h}\right]\right]$. If $h_G < T$, then using the Chernoff bound (Corollary A.2.5) gives us

$$\begin{aligned}
&\mathbf{Pr}\left[\frac{N}{|S|} \cdot \left(\sum_{v \in S} X_v\right) \geq 3T\right] \\
= \ &\mathbf{Pr}\left[\sum_{v \in S} X_v \geq \frac{3T \cdot |S| \cdot h_G}{N \cdot h_G}\right] \\
= \ &\mathbf{Pr}\left[\sum_{v \in S} X_v > \left(1 + \frac{3T}{h_G} - 1\right) \cdot \mathbf{E}\left[\sum_{v \in S} X_v\right]\right] \\
\leq \ &\exp\left(-\frac{3 \log N}{\varepsilon^2} \frac{N}{T} \cdot \frac{h_G}{N} \cdot \frac{\frac{3T}{h_G} - 1}{3}\right) \\
\leq \ &\exp\left(-\frac{3 \log N}{\varepsilon^2} \frac{N}{T} \cdot \frac{h_G}{N} \cdot \frac{2T}{3 \cdot h_G}\right) \leq \frac{1}{N},
\end{aligned}$$

where the last inequality follows from $\varepsilon \leq \sqrt{2}$. □

Lemma 6.3.5. *Let T be an integer and $\varepsilon \in (0,1)$. Then there exists a 2-pass algorithm for dynamic streams that outputs a value $\widehat{s}$ which is a $(1 \pm \varepsilon)$ estimation of s_G if $s_G \geq T$ and which is smaller than $3T$ if $s_G < T$. The algorithm uses $\mathcal{O}\left(\frac{a(G) \cdot N \log^4 N}{\varepsilon^2 T}\right)$ space and succeeds with high probability.*

Proof. In the first pass, we sample $\frac{3 \log N}{\varepsilon^2} \frac{a(G) \cdot N}{T}$ edges uniformly at random using ℓ_0-samplers, each of which cost at most $\mathcal{O}(\log^3 N)$ space (Lemma 2.5.3). For each node of a sampled edge, we maintain its degree in the second pass to decide whether a given edge is shallow

or not. Hereafter, we reapply the analysis of Lemma 6.3.4: Let $S = (e_1, \ldots, e_{|S|})$ be the sequence of sampled edges in the first pass and let X_i be the indicator variable which is 1 if and only if e_i is shallow. The probability of sampling a shallow edge is $\frac{s_G}{|E|}$ which implies that $\mathbf{E}\left[\sum X_i\right] = |S| \cdot \frac{s_G}{|E|} \geq |S| \cdot \frac{s_G}{a(G)\cdot N}$. Now, let $\widehat{s} = \frac{|E|}{|S|}\sum X_i$ be our estimator. We know that $\mathbf{E}\left[\widehat{s}\right] = s_G$. If $s_G \geq T$ then by Chernoff we have

$$\begin{aligned}\mathbf{Pr}\left[\hat{s} \geq (1+\varepsilon)\cdot\mathbf{E}\left[\hat{s}\right]\right] &= \mathbf{Pr}\left[\sum X_i \geq (1+\varepsilon)\cdot\mathbf{E}\left[\sum X_i\right]\right] \\ &\leq \exp\left(-\frac{3\log N}{\varepsilon^2}\frac{a(G)\cdot N}{T}\cdot\frac{s_G}{a(G)\cdot N}\cdot\frac{\varepsilon^2}{3}\right) \leq \frac{1}{N}.\end{aligned}$$

The same bound also holds for $\mathbf{Pr}\left[\hat{s} \leq (1-\varepsilon)\cdot\mathbf{E}\left[\hat{s}\right]\right]$. If $s_G < T$, then using the Chernoff bound gives us

$$\begin{aligned}&\mathbf{Pr}\left[\frac{|E|}{|S|}\cdot\left(\sum X_i\right) \geq 3T\right] \\ =\ &\mathbf{Pr}\left[\sum X_i \geq \frac{3T\cdot|S|\cdot s_G}{|E|\cdot s_G}\right]\end{aligned}$$

$$\begin{aligned}&= \mathbf{Pr}\left[\sum X_i > \left(1+\frac{3T}{s_G}-1\right)\cdot\mathbf{E}\left[\sum X_i\right]\right] \\ &\leq \exp\left(-\frac{3\log N}{\varepsilon^2}\frac{a(G)\cdot N}{T}\cdot\frac{s_G}{a(G)\cdot N}\cdot\frac{\frac{3T}{s_G}-1}{3}\right) \\ &\leq \exp\left(-\frac{3\log N}{\varepsilon^2}\frac{a(G)\cdot N}{T}\cdot\frac{s_G}{a(G)\cdot N}\cdot\frac{2T}{3\cdot s_G}\right) \leq \frac{1}{N},\end{aligned}$$

where the last inequality follows from $\varepsilon \leq \sqrt{2}$. □

Lemma 6.3.6. *Let $\varepsilon \in (0,1)$ and $T > (16C/\varepsilon)^2$ be an integer. Then there exists a 1-pass algorithm for dynamic streams that outputs a value $\widehat{s}$ which is a $(1\pm\varepsilon)$ estimation of s_G if $s_G \geq T$ and which is smaller than $3T$ if $s_G < T$. The algorithm uses $\tilde{\mathcal{O}}\left(\frac{a(G)\cdot N}{\varepsilon\sqrt{T}}\right)$ space and succeeds with constant probability.*

Proof. Let S be a set of $\frac{4N}{\varepsilon\sqrt{T}}$ randomly chosen nodes. We maintain the entire subgraph induced by S and the degree of each node in S. Note that the number of edges in this subgraph at the end of the stream is at most $a(G)\cdot|S|$. Since we have edge deletions this number may be exceeded at some point during the stream. Thus, we cannot explicitly store the subgraph but we can recover all entries using an $(a(G)\cdot|S|)$-sparse recovery sketch using $\tilde{\mathcal{O}}(a(G)\cdot|S|)$ space. Let $e_1, \ldots, e_{s_G}$ be the shallow edges in G. Define $X_i = 1$ if $e_i \in E(S)$ and 0 otherwise. X_i is Bernoulli distributed where the probability of both nodes being included in the subgraph follows from the hypergeometric distribution (see Appendix A.2) with population N, 2 successes in the population, sample size $|S|$ and 2 successes in the

sample:

$$p = \frac{\binom{2}{2}\binom{N-2}{|S|-2}}{\binom{N}{|S|}} = \frac{|S| \cdot (|S|-1)}{N \cdot (N-1)} \geq \frac{|S|^2}{2N^2} = \frac{8}{\varepsilon^2 T}.$$

Hence X_i is Bernoulli distributed, we have $\mathbf{Var}[X_i] = p \cdot (1-p) \leq p$. We know that $\mathbf{Var}[\sum X_i] = \sum \mathbf{Var}[X_i] + \sum_{i \neq j} \mathbf{Cov}[X_i, X_j]$. For the covariance between two variables X_i and X_j we have two cases: If e_i and e_j do not share a node, then X_i and X_j cannot be positively correlated, i. e., $\mathbf{Cov}[X_i, X_j] > 0$. To be more precise, we observe that by definition $\mathbf{Cov}[X_i, X_j]$ is equal to $\mathbf{E}[X_i X_j] - \mathbf{E}[X_i] \cdot \mathbf{E}[X_j]$ which is equal to $\mathbf{Pr}[X_i = X_j = 1] - p^2$. The probability $\mathbf{Pr}[X_i = X_j = 1]$ is equal to the probability of drawing exactly 4 fixed nodes from V with a sample of size $|S|$ which is

$$\frac{\binom{4}{4}\binom{N-4}{|S|-4}}{\binom{N}{|S|}} = \frac{|S| \cdot (|S|-1) \cdot (|S|-2) \cdot (|S|-3)}{N \cdot (N-1) \cdot (N-2) \cdot (N-3)}.$$

Due to Lemma A.1.2, we have $\frac{|S|-c}{N-c} \leq \frac{|S|-c+1}{N-c+1} \leq \cdots \leq \frac{|S|}{N}$.Therefore, $\mathbf{Pr}[X_i = X_j = 1]$ is at most p^2 which means that the covariance is at most 0. If e_i and e_j share a node, we have

$$\begin{aligned} \mathbf{Cov}[X_i, X_j] &\leq \mathbf{Pr}[X_i = X_j = 1] \\ &= \frac{\binom{3}{3}\binom{N-3}{|S|-3}}{\binom{N}{|S|}} = \frac{|S| \cdot (|S|-1) \cdot (|S|-2)}{N \cdot (N-1) \cdot (N-2)} \underset{\text{Lem. A.1.2}}{\leq} p^{3/2}. \end{aligned}$$

By definition each node incident to a shallow edge has at most C neighbors and therefore, we have at most $2C$ edges that share a node with a given shallow edge. In total, we can bound the variance of X

$$\begin{aligned} \mathbf{Var}[X] &= \sum \mathbf{Var}[X_i] + \sum_{i \neq j} \mathbf{Cov}[X_i, X_j] \\ &\leq p \cdot s_G + \sum_{\substack{e_i \neq e_j, \\ e_i, e_j \text{ share a node}}} \mathbf{Cov}[X_i, X_j] \leq p \cdot s_G + 2C \cdot s_G \cdot p^{3/2} \\ &\leq p \cdot s_G + 2C \cdot p \cdot s_G \frac{8}{\varepsilon\sqrt{T}} \leq 2p \cdot s_G \end{aligned}$$

where the last two inequalities follow from $\sqrt{p} \leq \sqrt{\frac{|S|^2}{N(N-1)}} \leq \frac{|S|}{N/2} = \frac{8}{\varepsilon\sqrt{T}}$ and $T \geq (16C/\varepsilon)^2$.

Using Chebyshev's inequality we have for $s_G \geq T$

$$\begin{aligned}\mathbf{Pr}\left[\left|\frac{1}{p}\cdot X - \frac{1}{p}\mathbf{E}[X]\right| > \epsilon\cdot\frac{1}{p}\mathbf{E}[X]\right] &= \mathbf{Pr}\left[|X-\mathbf{E}[X]| > \epsilon\cdot\mathbf{E}[X]\right]\\ &\leq \frac{\mathbf{Var}[X]}{\epsilon^2\mathbf{E}[X]^2} \leq \frac{2p\cdot s_G}{\epsilon^2 p^2\cdot s_G^2} \leq \frac{2}{\epsilon^2 Tp}\\ &\leq \frac{2\varepsilon^2 T}{8\epsilon^2 T} = \frac{1}{4}.\end{aligned}$$

If $s_G < T$, we have $\mathbf{E}[X] = p\cdot s_G < pT$. Thus, it is

$$\begin{aligned}\mathbf{Pr}\left[\frac{1}{p}\cdot X \geq 3T\right] &= \mathbf{Pr}\left[X-\mathbf{E}[X] \geq 3Tp - \mathbf{E}[X]\right]\\ &\leq \mathbf{Pr}\left[|X-\mathbf{E}[X]\,| \geq 2Tp\right]\\ &\leq \frac{\mathbf{Var}[X]}{4T^2p^2} \leq \frac{2p\cdot s_G}{4T^2p^2} \leq \frac{2}{4Tp} \leq \frac{2\varepsilon^2 T}{32T} = \frac{\varepsilon^2}{16} \leq \frac{1}{16}.\end{aligned}$$

□

Algorithm 14 Unweighted Matching Approximation

Input: $G=(V,E)$ with $a(G)\leq\alpha$ and $\varepsilon\in(0,1)$
Output: Estimator on the size of a maximum matching

Set $T = n^{2/5}$ for a single pass and $T = n^{1/3}$ for two passes and $\eta = 2.5\lceil 2\cdot\alpha+3\rceil + 5.75$.
Let $\hat{h}$ and $\hat{s}$ be the estimators from Lemma 6.3.4 and Lemma 6.3.6
for $i = 0,\ldots,\log 3T/(1-\varepsilon)$ **do**
  Solve rank decision with parameter $k=2^i$ on the Tutte-Matrix $T(G)$ for a random assignment to the indeterminates
end for
if $\operatorname{rank}(T(G)) < 3T/(1-\varepsilon)$ **then**
  Output 2^{i+1} for the maximal $i\in\{0,\ldots,2^{\log 3T/(1-\varepsilon)}\}$ with $\operatorname{rank}(T(G))\geq 2^i$
else
  Output $\dfrac{\max\{\hat{h},\hat{s}\}}{(1+\varepsilon)\eta}$.
end if

Algorithm 14 shows the idea of the estimation of the unweighted maximum matching size in bounded arboricity graphs using the previous results and the relation between the rank of the Tutte matrix and the matching size.

Theorem 6.3.7. *Let G be a graph with $a(G) \leq \alpha$ and $N \geq (16\alpha/\varepsilon)^5$. Let $\varepsilon \in (0,1)$. Then there exists an algorithm estimating the size of the maximum matching in G within a $\frac{2(1+\varepsilon)(5\cdot a(G)+\mathcal{O}(1))}{(1-\varepsilon)}$-factor in the dynamic streaming model*

- *using a single pass over the data and $\tilde{\mathcal{O}}(\frac{\alpha \cdot N^{4/5}}{\varepsilon^2})$ space that succeeds with constant probability or*
- *using 2 passes over the data and $\tilde{O}(\alpha \cdot N^{2/3})$ space that succeeds with high probability.*

Proof. We condition on the event that the estimators succeed which gives the desired success probability of the entire algorithms. For the sake of simplicity we assume that $3T/(1-\varepsilon)$ is a power of two. We know that we can decide the rank decision problem with parameter k in a dynamic stream with one pass using $\mathcal{O}(k^2 \log^2 N)$ space by Theorem 2.5.8. Thus, invoking this algorithm for $k = 2^0, 2^1, \ldots, 2^{\log(3T/(1-\varepsilon))}$ results in a space requirement of $\mathcal{O}(T^2 \cdot \log T \cdot \log^2 N) = \mathcal{O}(T^2 \log^3 N)$ for our choices of T. For the first part of the theorem, we estimate s_G and h_G in 1-pass by $\hat{h}$ and $\hat{s}$ using $\tilde{O}\left(\frac{N}{\varepsilon^2 T}\right)$ and $\tilde{O}\left(\frac{\alpha \cdot N}{\varepsilon\sqrt{T}}\right)$ space, see Lemma 6.3.4 and Lemma 6.3.6. Setting $T = N^{2/5}$ gives us the desired space bound of $\tilde{\mathcal{O}}\left(\frac{\alpha \cdot N^{4/5}}{\varepsilon^2}\right)$ (note that $T > (16\alpha/\varepsilon)^2$ which is required for Lemma 6.3.6). For the second part of the theorem, we can improve the space requirements for the estimator $\hat{h}$ and $\hat{s}$ to $\tilde{O}\left(\frac{a(G)N}{T}\right)$ by using Lemma 6.3.4 and Lemma 6.3.5. Now, setting $T = N^{1/3}$ gives the desired space bound.

Let OPT be the size of a maximum matching. First, we check whether $OPT \geq 2 \cdot 3T/(1-\varepsilon)$ by invoking the rank decision algorithm with parameter $k = 3T/(1-\varepsilon)$. Since the rank of the matrix is equal to $2OPT$, this decides whether $OPT \geq 2 \cdot 3T/(1-\varepsilon)$. If this is not true, we can give a 2-approximation on OPT by testing whether the rank of the Tutte matrix is in $[2^i, 2^{i+1})$ for $i = 0, \ldots, \log(3T/(1-\varepsilon)) - 1$. If $OPT \geq 2 \cdot 3T/(1-\varepsilon)$ Lemma 6.3.3 implies that $\max\{h_G, s_G\} \geq 3T/(1-\varepsilon)$ since $h_G + s_G \geq OPT$. Assuming that we can approximate $\max\{h_G, s_G\}$ then again by Lemma 6.3.3 we can estimate OPT since

$$\frac{\max\{h_G, s_G\}}{\eta} \leq OPT \leq h_G + s_G \leq 2\max\{h_G, s_G\}.$$

W.l.o.g. let $\widehat{h} = \arg\max\{\hat{h}, \hat{s}\}$. Now we have two cases:

1. If $h_G = \arg\max\{h_G, s_G\} \geq T$ then by Lemma 6.3.4 $\widehat{h}$ is a $(1 \pm \varepsilon)$ estimation on h_G.
2. If $s_G = \arg\max\{h_G, s_G\} \geq 3T/(1-\varepsilon)$ we know by Lemma 6.3.6 that $\hat{s} \geq 3T$ which implies that $\widehat{h} \geq \hat{s} \geq 3T$. Thus by Lemma 6.3.4 $\widehat{h}$ is a $(1 \pm \varepsilon)$ estimation on h_G. This gives us
$$(1-\varepsilon)s_G \leq \hat{s} \leq \widehat{h} \leq (1+\varepsilon)h_G \leq (1+\varepsilon)s_G.$$

Therefore, $\max\{\hat{h}, \hat{s}\}$ is a good estimator for $\max\{h_G, s_G\}$. For the estimator $\frac{\max\{\hat{h},\hat{s}\}}{(1+\varepsilon)\eta}$ we

have

$$\frac{(1-\varepsilon)}{2(1+\varepsilon)\eta} \cdot OPT \leq \frac{(1-\varepsilon)\max\{h_G, s_G\}}{(1+\varepsilon)\eta} \leq \frac{\max\{\hat{h}, \hat{s}\}}{(1+\varepsilon)\eta} \leq \frac{(1+\varepsilon)\max\{h_G, s_G\}}{(1+\varepsilon)\eta} \leq OPT.$$

□

We want to mention that it is also possible to maintain a small matching in 2-passes using only sublinear space with respect to the number of edges in a dynamic stream. This algorithm can also be used for the 2-pass algorithm above also using $\tilde{O}(a(G) \cdot N^{2/3})$ space.

Lemma 6.3.8. *Let G be a graph with M edges and let $\alpha_1 \in (0,1]$ and $\alpha_2 \in (0,2)$ with $\alpha_1 \leq \alpha_2$. There is a dynamic streaming algorithm that maintains a matching of size N^{α_1} in space $\tilde{O}(\frac{M}{N^{\alpha_2}} \cdot N^{\alpha_1} + N^{\alpha_2})$ using two passes with high probability.*

Proof. We denote by F the unmatched nodes w.r.t. some matching (which is clear from the context). Let M be some matching and assume that $|E(F)|$ is large. If we repeatedly sample an edge uniformly at random, then we will see an edge from $E(F)$ after a small number of steps with high probability. If we continue this process long enough and update our matching M every time we see an free edge, we have either found a large matching or the remaining number of free edges is small.

Let $T = 2 \cdot \frac{M}{N^{\alpha_2}} \cdot N^{\alpha_1}$. In the first pass we sample a sequence $(e_1, \cdots, e_T)$ of T edges uniformly at random. After each sampled edge we update a matching M_1 (starting with an empty matching) by adding an edge to M_1 iff both endpoints are currently unmatched. If after the first pass M_1 is smaller than N^{α_1}, we build an N^{α_2}-recovery sketch for the edges in $E(F)$ where F is the set of unmatched nodes.

Let F_i be the set of unmatched nodes before we sample edge e_i, i. e., $F_1 = V$, and let F_{T+1} be the set of unmatched nodes after the first pass. Assume that $|E(F_{T+1})| \geq N^{\alpha_2}$. Then we know that $|E(F_i)| \geq N^{\alpha_2}$ for every $1 \leq i \leq T$. Let $X_i = 1$ iff $e_i \in E(F_i)$ and let $X = \sum_{i=1}^{T} X_i$, i. e., $|M_1| = \min\{N^{\alpha_1}, X\}$. It is

$$\mathbf{E}\left[X \,\middle|\, |E(F_{T+1})| \geq N^{\alpha_2}\right] \geq T \cdot \frac{N^{\alpha_2}}{M} = 2 \cdot N^{\alpha_1}.$$

Since adding an edge to M_1 at step i, i. e., $X_i = 1$, decreases the probability that $X_j = 1$ for $j > i$ while $X_i = 0$ does not influence the probabilities, these random variables are negatively correlated (as defined in Definition 2.4). Thus, we can use Chernoff bounds (Theorem 2.4.5) to prove that

$$\begin{aligned}\mathbf{Pr}\left[X < N^{\alpha_1} \,\middle|\, |E(F_{T+1})| \geq N^{\alpha_2}\right] &\leq \mathbf{Pr}\left[X < (1/2)E[X] \,\middle|\, |E(F_{T+1})| \geq N^{\alpha_2}\right] \\ &\leq e^{-N^{\alpha_1}/4}\end{aligned}$$

which gives us

$$\begin{aligned} & \mathbf{Pr}\left[X < N^{\alpha_1} \wedge |E(F_{T+1})| \geq N^{\alpha_2}\right] \\ = \; & \mathbf{Pr}\left[X < N^{\alpha_1} \,\middle|\, |E(F_{T+1})| \geq N^{\alpha_2}\right] \cdot \mathbf{Pr}\left[|E(F_{T+1})| \geq N^{\alpha_2}\right] \\ \leq \; & \mathbf{Pr}\left[X < N^{\alpha_1} \,\middle|\, |E(F_{T+1})| \geq N^{\alpha_2}\right] \leq e^{-N^{\alpha_1}/4}. \end{aligned}$$

Therefore, we have either $|M_1| = N^{\alpha_1}$ or $|E(F_{T+1})| < N^{\alpha_2}$ after the first pass with high probability.

Sampling T edges can be done by T ℓ_0-sampler in space $\tilde{O}(T)$. In the second pass we have the matching and an N^{α_2}-recovery sketch where both can be maintain in $\tilde{O}(N^{\alpha_2})$ space. □

Graphs with arboricity $a(G)$ can only have at most $a(G) \cdot N$ edges. Therefore, by using Lemma 6.3.8 with $M = a(G) \cdot N, \alpha_1 = 1/3$ and $\alpha_2 = 2/3$ we can maintain a matching of size at most $N^{1/3}$ in space $\tilde{O}(a(G) \cdot N^{2/3})$.

Corollary 6.3.9. *There is a 2-pass dynamic streaming algorithm using $\tilde{O}(\alpha \cdot N^{2/3})$ space that maintains a matching of size at most $N^{1/3}$ in graphs with bounded arboricity $a(G) \leq \alpha$.*

6.4. Applications of the Weighted Matching Estimation Algorithm

Since every edge insertion and deletion supplies the edge weight, it is straightforward to determine the rank for each edge upon every update. Using the following results for unweighted matching, we can straightforwardly obtain estimates with similar approximation guarantee and space bounds for weighted matching by using Theorem 6.2.4.

Random Order Streams

For an arbitrary graph whose edges are streamed in random order, Kapralov, Khanna and Sudan [74] gave an algorithm with polylog N approximation guarantee and failure probability $\delta = 1/\operatorname{polylog} N$ using polylog N space. Since this probability is over the randomness of the input stream we cannot easily amplify it. Though if $\log W + 1 \leq 1/(c \cdot \delta)$ the extension to weighted matching in random order streams still succeeds with constant probability $1 - 1/c$.

Adversarial Streams

For graphs of bounded arboricity, Esfandiari et al. [44] gave an algorithm with constant approximation guarantee using $\tilde{O}(N^{2/3})$ space.

Dynamic Streams

Using the results from Section 6.3, we can estimate the value of an optimal weighted matching in trees using $\mathcal{O}(\varepsilon^{-2} \log^3 N)$ space and in graphs with bounded arboricity α in $\tilde{\mathcal{O}}(\alpha N^{4/5}/\varepsilon^2)$ space.

6.5. Space Lower Bound of Streaming Algorithms Estimating the Size of Matchings

In this section, we give a space lower bound for streaming algorithms estimating the size of unweighted matchings with small approximation factors. Esfandiari et al. [44] showed a lower bound of $\Omega(\sqrt{N})$ bits of space for any estimation better than 3/2. Their reduction (see below) uses the Boolean Hidden Matching Problem introduced by Bar-Yossef, Jayram, and Kerenedis [11], and further studied by Gavinsky et al. [49]. We will use the Boolean Hidden Hypermatching Problem which we have defined in Section 2.6. By Definition 2.6.2, in the Boolean Hidden Hypermatching problem $BHH_{t,N}$, Alice gets a Boolean vector $x \in \{0,1\}^N$ with $N = 2kt$ for some $k \in \mathbb{N}$ and Bob gets a perfect t-hypermatching M on the N coordinates of x, i. e., each edge has exactly t coordinates, and a string $w \in \{0,1\}^{N/t}$. It is promised that either $Mx \oplus w = 1^{N/t}$ or $Mx \oplus w = 0^{N/t}$. The problem is to return 1 in the first case and 0 otherwise.

For our reduction we require the vector w to be the all zero vector, i. e., $Mx = 1^{N/t}$ or $Mx = 0^{N/t}$. We first show that this does not reduce the communication complexity.

Definition 6.5.1. The problem $BHH^0_{t,N}$ is the same as the $BHH_{t,N}$ problem with w fixed to be $0^{N/t}$ and $x \in \{0,1\}^N$ has exactly $N/2$ bits equal to 1.

Lemma 6.5.2. *The communication complexity of $BHH^0_{t,4N}$ is lower bounded by the communication complexity of $BHH_{t,N}$.*

Proof. First, let assume that t is odd. Let $x \in \{0,1\}^N$ with $N = 2kt$ for some $k \in \mathbb{N}$ and M be a perfect t-hypermatching on the N coordinates of x and $w \in \{0,1\}^{N/t}$. We define $x' = [x, x, \overline{x}, \overline{x}]$ to be the concatenation of two identical copies of x and two identical copies of the vector resulting from the bitwise negation of x. W.l.o.g. let $\{x_1, \ldots, x_t\} \in M$ be the l-th hyperedge of M. Then we add the following four hyperedges to M':

- $\{x_1, \overline{x_2}, \ldots, \overline{x_t}\}, \{\overline{x_1}, x_2, \overline{x_3}, \ldots, \overline{x_t}\}$, $\{\overline{x_1}, \overline{x_2}, x_3, \ldots, x_t\}$, and $\{x_1, \ldots, x_t\}$ if $w_l = 0$,
- $\{\overline{x_1}, x_2, \ldots, x_t\}, \{x_1, \overline{x_2}, x_3, \ldots, x_t\}$, $\{x_1, x_2, \overline{x_3}, \ldots, \overline{x_t}\}$, and $\{\overline{x_1}, \ldots, \overline{x_t}\}$ if $w_l = 1$.

Note that we use every bit of the vector x exactly four times: Two times x_i and two times $\overline{x_i}$ for every $i \in [n]$. Therefore, it is possible to construct these edges with the bits of x' such that M' is a perfect matching on the coordinates of x'. We omit this here for the sake of readability. The important observation here is that we flip an even number of bits in the case $w_l = 0$ and an odd number of bits if $w_l = 1$ (since t is odd). Since every bit flip results in a change of the parity of the set of bits, the parity does not change if we flip an even number of bits and the parity flips if we negate an odd number of bits. Therefore, if w_l is the correct (respectively wrong) parity of $\{x_1, \ldots, x_t\}$ then the parity of the added sets is 0 (respectively

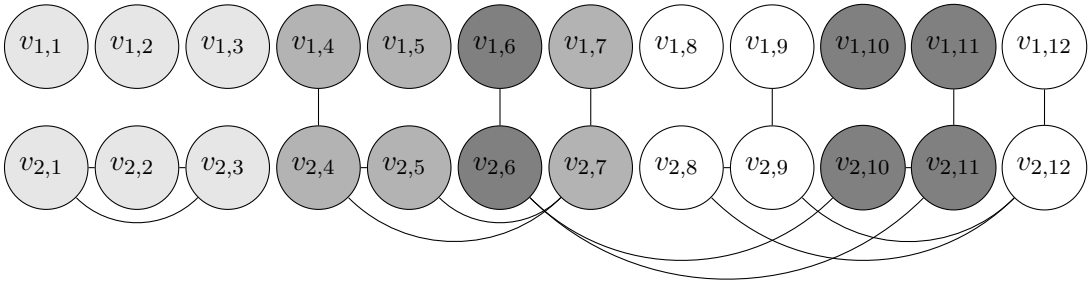


Figure 6.2.: Worst case instance for $t = 3$. Bob's hypermatching (illustrated by the shadings) corresponds to disjoint 3-cliques among the lower nodes and Alice' input vector $x \in \{0,1\}^{12}$ corresponds to the edges between upper and lower nodes, i. e., $x = (0,0,0,1,0,1,1,0,1,0,1,1)$.

1), i. e., $M'x' = 0^{2N}$ if $Mx \oplus w = 0^{N/2}$ and $M'x' = 1^{2N}$ if $Mx \oplus w = 1^{N/2}$. The number of ones in $x' \in \{0,1\}^{4N}$ is exactly $2N$. If t is even, we can just change the cases for the added edges such that we flip an even number of bits in the case $w_l = 0$ and an odd number of bits if $w_l = 1$. Overall, this shows that a lower bound for $BHH_{t,N}$ implies a lower bound for $BHH^0_{t,4N}$. □

Let us now sketch the reduction by Esfandiari et al. [44] from $BHH^0_{2,N}$ to approximate maximum matching to get the idea how to extend it to the general bound. Let x, M be the input for Alice and Bob. They construct a graph consisting of $2N$ nodes denoted by $v_{1,i}$ and $v_{2,i}$, for $i \in \{1,\dots,N\}$. For each bit x_i of $x \in \{0,1\}^N$, Alice adds an edge $\{v_{1,i}, v_{2,i}\}$ iff $x_i = 1$ and sends a message to Bob. Bob adds an edge between $v_{2,i}$ and $v_{2,j}$ for each edge $\{x_i, x_j\} \in M$ and approximates the size of the matching. If all parities are 1 then the size of the maximum matching is $N/2$. If the parities are all 0 then the size is $3N/4$. Every streaming algorithm that approximates better than $3/2$ can distinguish between these two cases. The first observation is that the size of the matching is lower bounded by the number of ones in x. The second observation is that the added edges by Bob increase the matching iff the parities of all pairs are 0 and only the edges between the two 0 input bits of Alice increase the matching. Since it is promised that all parities are equal and the number of ones is exactly $N/2$, we can calculate the number of $(0,0)$ pairs. For our lower bound we show that this calculation is still possible if Bob adds a t-clique between the corresponding nodes of the hyperedge.

Theorem 6.5.3. *Any randomized streaming algorithm that approximates the size of a maximum matching of a graph with N nodes within a factor better than $1 + \frac{1}{3t/2-1}$ for some integer $t \geq 2$ needs $\Omega(N^{1-1/t})$ space.*

Proof. Let x, M be the input to the $BHH^0_{t,N}$ problem, i. e., M is a perfect t-hypermatching on the coordinates of x, x has exactly $N/2$ ones and it is promised that either $Mx = 0^{N/t}$

or $Mx = 1^{N/t}$. We construct the graph for the reduction as described above: For each bit x_i we have two nodes $v_{1,i}, v_{2,i}$ and Alice adds the edge $\{v_{1,i}, v_{2,i}\}$ iff $x_i = 1$. For each edge $\{x_{i_1}, \ldots, x_{i_t}\} \in M$ Bob adds a t-clique consisting of the nodes $v_{2,i_1}, \ldots, v_{2,i_t}$. For now, let us assume t to be odd. We know that the matching is at least $N/2$ because x has exactly $N/2$ ones. Since Bob adds a clique for every edge it is always possible to match all (or all but one) nodes of the clique whose corresponding bit is 0. In the case of $Mx = 0^{N/t}$ the parity of every edge is 0, i. e., the number of nodes whose corresponding bit is 1 is even. Let $M_{2i} \subseteq M$ be the hyperedges containing exactly $2i$ one bits and define $l_{2i} := |M_{2i}|$. Then we know $N/2 = \sum_{i=0}^{\lfloor t/2 \rfloor} 2i \cdot l_{2i}$ and $|M| = N/t = \sum_{i=0}^{\lfloor t/2 \rfloor} l_{2i}$. For every edge in M_{2i} the size of the maximum matching within the corresponding subgraph is exactly $2i + \lfloor (t-2i)/2 \rfloor = 2i + \lfloor t/2 \rfloor - i$ for every $i = 0, \ldots, \lfloor t/2 \rfloor$ (see Fig. 6.2). Thus, we have a matching of size

$$\sum_{i=0}^{\lfloor t/2 \rfloor} (2i + (\lfloor t/2 \rfloor - i)) l_{2i} = \frac{N}{2} + \frac{t-1}{2} \cdot \frac{N}{t} - \frac{N}{4} = \frac{3N}{4} - \frac{N}{2t}.$$

If we have $Mx = 1^{N/t}$ then let $M_{2i+1} \subseteq M$ be the hyperedges containing exactly $2i+1$ one bits and define $l_{2i+1} := |M_{2i+1}|$. Again, we know $N/2 = \sum_{i=0}^{\lfloor t/2 \rfloor} (2i+1) \cdot l_{2i+1}$ and $|M| = N/t = \sum_{i=0}^{\lfloor t/2 \rfloor} l_{2i+1}$. For every edge in M_{2i+1} the size of the maximum matching within the corresponding subgraph is exactly $2i + 1 + (t - 2i - 1)/2 = 2i + 1 + \lfloor t/2 \rfloor - i$ for every $i = 0, \ldots, \lfloor t/2 \rfloor$. Thus, the maximum matching has a size of

$$\sum_{i=0}^{\lfloor t/2 \rfloor} (2i + 1 + (\lfloor t/2 \rfloor - i)) l_{2i+1} = \frac{N}{2} + \frac{t-1}{2} \cdot \frac{N}{t} - \frac{1}{2} \sum_{i=0}^{\lfloor t/2 \rfloor} (2i+1) \cdot l_{2i+1} + \frac{N}{2t} = \frac{3N}{4}.$$

For t even, the size of the matching is

$$\sum_{i=0}^{t/2} (2i + (t-2i)/2) l_{2i} = \frac{N}{2} + \frac{t}{2} \cdot \frac{N}{t} - \frac{N}{4} = \frac{3N}{4}$$

if $Mx = 0^{N/t}$. Otherwise, we have

$$\begin{aligned} \sum_{i=0}^{t/2} \left(2i + 1 + \left\lfloor \frac{t-2i-1}{2} \right\rfloor\right) l_{2i+1} &= \frac{N}{2} + \sum_{i=0}^{t/2} (t/2 - i - 1) l_{2i+1} \\ &= \frac{N}{2} - (t/2-1) \cdot \frac{N}{t} - \frac{N}{4} + \frac{N}{2t} = \frac{3N}{4} - \frac{N}{2t}. \end{aligned}$$

As a consequence, every streaming algorithm that computes an λ-approximation on the size of a maximum matching with

$$\lambda < \frac{(3/4)N}{((3/4) - 1/(2t))N} = 1/(1 - 4/6t) = 1 + \frac{1}{3t/2 - 1}$$

can distinguish between $Mx = 0^{N/t}$ and $Mx = 1^{N/t}$ and, thus, needs $\Omega(N^{1-1/t})$ space. □

Finally, constructing the Tutte-matrix with randomly chosen entries gives us

Corollary 6.5.4. *Any randomized algorithm that approximates the rank of a streamed matrix $A \in \mathbb{R}^{N \times N}$ within a factor better than $1 + \frac{1}{3t/2-1}$ for some integer $t \geq 2$ requires $\Omega(N^{1-1/t})$ space.*

Bibliography

[1] Ahn, K. J., and Guha, S. Linear programming in the semi-streaming model with application to the maximum matching problem. *Information and Computation 222* (2013), 59–79.

[2] Ahn, K. J., Guha, S., and McGregor, A. Analyzing graph structure via linear measurements. In *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)* (2012), pp. 459–467.

[3] Alon, N., Babai, L., and Itai, A. A fast and simple randomized parallel algorithm for the maximal independent set problem. *Journal of Algorithms 7*, 4 (1986), 567–583.

[4] Alon, N., Goldreich, O., Håstad, J., and Peralta, R. Simple construction of almost k-wise independent random variables. *Random Structures and Algorithms 3*, 3 (1992), 289–304.

[5] Alon, N., Goldreich, O., and Mansour, Y. Almost k-wise independence versus k-wise independence. *Information Processing Letters 88*, 3 (2003), 107–110.

[6] Alon, N., Matias, Y., and Szegedy, M. The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences 58*, 1 (1999), 137–147.

[7] Alon, N., and Nussboim, A. k-wise independent random graphs. In *Proceedings of the 49th Annual IEEE Symposium on Foundations of Computer Science (FOCS)* (2008), pp. 813–822.

[8] Arnold, D. B., and Sleep, M. R. Uniform random generation of balanced parenthesis strings. *ACM Transactions on Programming Languages and Systems 2*, 1 (1980), 122–128.

[9] Assadi, S., Khanna, S., Li, Y., and Yaroslavtsev, G. Tight bounds for linear sketches of approximate matchings. *ArXiv e-prints* (2015), 1505.01467.

[10] Awerbuch, B., Goldberg, A. V., Luby, M., and Plotkin, S. A. Network decomposition and locality in distributed computation. In *Proceedings of the 30th Annual IEEE Symposium on Foundations of Computer Science (FOCS)* (1989), pp. 364–369.

[11] Bar-Yossef, Z., Jayram, T. S., and Kerenidis, I. Exponential separation of quantum and classical one-way communication complexity. *SIAM Journal on Computing 38*, 1 (2008), 366–384.

[12] Barkay, N., Porat, E., and Shalem, B. Efficient sampling of non-strict turnstile data streams. *Theoretical Computer Science 590* (2015), 106–117. Fundamentals of Computation Theory.

[13] Bazzaro, F., and Gavoille, C. Localized and compact data-structure for comparability graphs. *Discrete Mathematics 309*, 11 (2009), 3465–3484.

[14] Bloem, R., Gabow, H. N., and Somenzi, F. An algorithm for strongly connected component analysis in $n \log n$ symbolic steps. *Formal Methods in System Design 28*, 1 (2006), 37–56.

[15] Bollig, B. On symbolic OBDD-based algorithms for the minimum spanning tree problem. *Theoretical Computer Science 447* (2012), 2–12.

[16] Bollig, B. On the complexity of some ordering problems. In *Proceedings of the 39th International Symposium on Mathematical Foundations of Computer Science (MFCS)* (2014), pp. 118–129.

[17] Bollig, B. On the width of ordered binary decision diagrams. In *Proceedings of the 8th International Conference on Combinatorial Optimization and Applications (COCOA)* (2014), pp. 444–458.

[18] Bollig, B., Gillé, M., and Pröger, T. Implicit computation of maximum bipartite matchings by sublinear functional operations. *Theoretical Computer Science 560* (2014), 131–146.

[19] Bollig, B., Löbbing, M., and Wegener, I. On the effect of local changes in the variable ordering of ordered decision diagrams. *Information Processing Letters 59*, 5 (1996), 233–239.

[20] Bollig, B., and Pröger, T. On efficient implicit OBDD-based algorithms for maximal matchings. *Information and Computation 239* (2014), 29–43.

[21] Bollig, B., and Wegener, I. Improving the variable ordering of OBDDs is NP-complete. *IEEE Transactions on Computers 45*, 9 (1996), 993–1002.

[22] Bollig, B., and Wegener, I. Asymptotically optimal bounds for OBDDs and the solution of some basic OBDD problems. *Journal of Computer and System Sciences 61*, 3 (2000), 558–579.

[23] Brandstädt, A., Le, V. B., and Spinrad, J. P. *Graph Classes: A Survey*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1999.

[24] Breitbart, Y., III, H. B. H., and Rosenkrantz, D. J. On the size of binary decision diagrams representing boolean functions. *Theoretical Computer Science 145*, 1&2 (1995), 45–69.

[25] Bryant, R. E. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers 35*, 8 (1986), 677–691.

[26] Carter, L., and Wegman, M. N. Universal classes of hash functions. *Journal of Computer and System Sciences 18*, 2 (1979), 143–154.

[27] Cherkassky, B. V., Goldberg, A. V., Martin, P., Setubal, J. C., and Stolfi, J. Augment or push: a computational study of bipartite matching and unit-capacity flow algorithms. *ACM Journal of Experimental Algorithmics 3* (1998), 8.

[28] Chitnis, R., Cormode, G., Esfandiari, H., Hajiaghayi, M., McGregor, A., Monemizadeh, M., and Vorotnikova, S. Kernelization via sampling with applications to dynamic graph streams. *ArXiv e-prints* (2015), 1505.01731.

[29] Chitnis, R. H., Cormode, G., Hajiaghayi, M. T., and Monemizadeh, M. Parameterized streaming: Maximal matching and vertex cover. In *Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)* (2015), pp. 1234–1251.

[30] Chor, B., Goldreich, O., Hastad, J., Freidmann, J., Rudich, S., and Smolensky, R. The bit extraction problem or t-resilient functions. In *Proceedings of the 26th Annual IEEE Symposium on Foundations of Computer Science (FOCS)* (1985), pp. 396–407.

[31] Christiani, T., and Pagh, R. Generating k-independent variables in constant time. In *Proceedings of the 55th Annual IEEE Symposium on Foundations of Computer Science (FOCS)* (2014), pp. 196–205.

[32] Chung, Y., Park, K., and CHO, Y. Parallel maximum matching algorithms in interval graphs. *International Journal of Foundations of Computer Science 10*, 01 (1999), 47–60.

[33] Clarkson, K. L., and Woodruff, D. P. Numerical linear algebra in the streaming model. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing (STOC)* (2009), pp. 205–214.

[34] Coppersmith, D., and Winograd, S. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation 9*, 3 (1990), 251–280.

[35] Cormen, T. H. *Introduction to algorithms.* MIT press, 2009.

[36] Crouch, M., and Stubbs, D. Improved streaming algorithms for weighted matching, via unweighted matching. In *Proceedings of the 18th Workshop on Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)* (2014), pp. 96–104.

[37] Crouch, M. S., McGregor, A., and Stubbs, D. Dynamic graphs in the sliding-window model. In *Proceedings of the 21st Annual European Symposium (ESA)* (2013), pp. 337–348.

[38] Czygrinow, A., Hanckowiak, M., and Szymanska, E. Fast distributed approximation algorithm for the maximum matching problem in bounded arboricity graphs. In *Proceedings of the 20th International Symposium on Symbolic and Algebraic Computation (ISSAC)* (2009), pp. 668–678.

[39] Davis, T. A., and Hu, Y. The University of Florida Sparse Matrix Collection. *ACM Transactions on Mathematical Software 38*, 1 (2011), 1:1–1:25.

[40] Dietzfelbinger, M. Universal hashing and k-wise independent random variables via integer arithmetic without primes. In *Proceedings of the 13th Annual Symposium on Theoretical Aspects of Computer Science (STACS)* (1996), pp. 569–580.

[41] Edmonds, J. Paths, trees, and flowers. *Canadian Journal of Mathematics 17* (1965), 449–467.

[42] Epstein, L., Levin, A., Mestre, J., and Segev, D. Improved approximation guarantees for weighted matching in the semi-streaming model. *SIAM Journal on Discrete Mathematics 25*, 3 (2011), 1251–1265.

[43] Epstein, L., Levin, A., Segev, D., and Weimann, O. Improved bounds for online preemptive matching. In *Proceedings of the 30th Annual Symposium on Theoretical Aspects of Computer Science (STACS)* (2013), pp. 389–399.

[44] Esfandiari, H., Hajiaghayi, M. T., Liaghat, V., Monemizadeh, M., and Onak, K. Streaming algorithms for estimating the matching size in planar graphs and beyond. In *Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)* (2015), pp. 1217–1233.

[45] Feigenbaum, J., Kannan, S., McGregor, A., Suri, S., and Zhang, J. On graph problems in a semi-streaming model. *Theoretical Computer Science 348*, 2-3 (2005), 207–216.

[46] Feigenbaum, J., Kannan, S., Vardi, M. Y., and Viswanathan, M. The complexity of problems on graphs represented as OBDDs. *Chicago Journal of Theoretical Computer Science* (1999).

[47] Flum, J., and Grohe, M. *Parameterized Complexity Theory.* Springer-Verlag New York, Inc., 2006.

[48] Gall, F. L. Powers of tensors and fast matrix multiplication. In *Proceedings of the 25th International Symposium on Symbolic and Algebraic Computation (ISSAC)* (2014), pp. 296–303.

[49] Gavinsky, D., Kempe, J., Kerenidis, I., Raz, R., and de Wolf, R. Exponential separation for one-way quantum communication complexity, with applications to cryptography. *SIAM Journal on Computing 38*, 5 (2008), 1695–1708.

[50] Gavoille, C., and Paul, C. Optimal distance labeling for interval graphs and related graph families. *SIAM Journal on Discrete Mathematics 22*, 3 (2008), 1239–1258.

[51] Gentilini, R., Piazza, C., and Policriti, A. Symbolic graphs: Linear solutions to connectivity related problems. *Algorithmica 50*, 1 (2008), 120–158.

[52] Gillé, M. OBDD-based representation of interval graphs. In *Proceedings of the 39th Workshop on Graph-Theoretic Concepts in Computer Science (WG)* (2013), pp. 286–297.

[53] Goel, A., Kapralov, M., and Khanna, S. On the communication and streaming complexity of maximum bipartite matching. In *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)* (2012), pp. 468–485.

[54] Greenlaw, R., Hoover, H. J., and Ruzzo, W. L. *Limits to Parallel Computation: P-completeness Theory.* Oxford University Press, Inc., 1995.

[55] Gu, T., Chang, L., and Xu, Z. A novel symbolic algorithm for maximum weighted matching in bipartite graphs. *International Journal of Communications, Network and System Sciences 4*, 2 (2011), 111–121.

[56] Gu, T., Chang, L., and Xu, Z. The symbolic OBDD algorithm for finding optimal semi-matching in bipartite graphs. *Communications and Network 3*, 2 (2011), 65–72.

[57] Hachtel, G. D., and Somenzi, F. A symbolic algorithms for maximum flow in 0-1 networks. *Formal Methods in System Design 10*, 2/3 (1997), 207–219.

[58] Harary, F., and Palmer, E. M. *Graphical enumeration.* Addison-Wesley, 1973.

[59] Henzinger, M. R., Raghavan, P., and Rajagopalan, S. Computing on data streams. In *External Memory Algorithms: DIMACS Workshop External Memory and Visualization* (1999), vol. 50, American Mathematical Society, pp. 107–118.

[60] Hojati, R., Touati, H., Kurshan, R. P., and Brayton, R. K. Efficient ω-regular language containment. In *Proceedings of the 4th International Workshop on Computer Aided Verification (CAV)* (1992), pp. 396–409.

[61] Hopcroft, J. E., and Karp, R. M. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing 2*, 4 (1973), 225–231.

[62] Hosaka, K., Takenaga, Y., Kaneda, T., and Yajima, S. Size of ordered binary decision diagrams representing threshold functions. *Theoretical Computer Science 180*, 1-2 (1997), 47–60.

[63] Huang, Z., Radunović, B., Vojnović, M., and Zhang, Q. Communication complexity of approximate matching in distributed graphs. In *Proceedings of the 32nd International Symposium on Theoretical Aspects of Computer Science (STACS)* (2015).

[64] Indyk, P. Sublinear time algorithms for metric space problems. In *Proceedings of the 31st Annual ACM Symposium on Theory of Computing (STOC)* (1999), pp. 428–434.

[65] Indyk, P. Stable distributions, pseudorandom generators, embeddings, and data stream computation. *Journal of the ACM 53*, 3 (2006), 307–323.

[66] Israeli, A., and Itai, A. A fast and simple randomized parallel algorithm for maximal matching. *Information Processing Letters 22*, 2 (1986), 77–80.

[67] Joffe, A. On a set of almost deterministic k-independent random variables. *The Annals of Probability 2*, 1 (1974), 161–162.

[68] Johnson, W. B., and Lindenstrauss, J. Extensions of Lipschitz mappings into a Hilbert space. *Contemporary Mathematics 26*, 189-206 (1984), 1–1.

[69] Jowhari, H., Saglam, M., and Tardos, G. Tight bounds for l_p samplers, finding duplicates in streams, and related problems. In *Proceedings of the 29th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)* (2011), pp. 49–58.

[70] Jukna, S. Entropy of contact circuits and lower bounds on their complexity. *Theoretical Computer Science 57* (1988), 113–129.

[71] Kabanets, V. Almost k-wise independence and hard boolean functions. *Theoretical Computer Science 297*, 1-3 (2003), 281–295.

[72] Kane, D. M., Nelson, J., and Woodruff, D. P. An optimal algorithm for the distinct elements problem. In *Proceedings of the 29th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)* (2010), pp. 41–52.

[73] Kapralov, M. Better bounds for matchings in the streaming model. In *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)* (2013), pp. 1679–1697.

[74] Kapralov, M., Khanna, S., and Sudan, M. Approximating matching size from random streams. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)* (2014), pp. 734–751.

[75] Kapralov, M., Khanna, S., and Sudan, M. Streaming lower bounds for approximating MAX-CUT. In *Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)* (2015), pp. 1263–1282.

[76] Karger, D. R., Klein, P. N., and Tarjan, R. E. A randomized linear-time algorithm to find minimum spanning trees. *Journal of the ACM 42*, 2 (1995), 321–328.

[77] Karp, R. M., and Sipser, M. Maximum matching in sparse random graphs. In *Proceedings of the 22nd Annual IEEE Symposium on Foundations of Computer Science (FOCS)* (1981), pp. 364–375.

[78] Kelsen, P. An optimal parallel algorithm for maximal matching. *Information Processing Letters 52*, 4 (1994), 223–228.

[79] Knuth, D. E., Graham, R. L., and Patashnik, O. *Concrete Mathematics*. Addison-Wesley Longman Publishing Co., Inc., 1994.

[80] Konrad, C. Maximum matching in turnstile streams. *ArXiv e-prints* (2015), 1505.01460.

[81] Konrad, C., Magniez, F., and Mathieu, C. Maximum matching in semi-streaming with few passes. In *Proceedings of the 16th Workshop on Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)* (2012), pp. 231–242.

[82] Kremer, I., Nisan, N., and Ron, D. On randomized one-round communication complexity. *Computational Complexity 8*, 1 (1999), 21–49.

[83] Kushilevitz, E., and Nisan, N. *Communication Complexity*. Cambridge University Press, 2006.

[84] Lai, T.-H., and Wei, S.-S. Bipartite permutation graphs with application to the minimum buffer size problem. *Discrete Applied Mathematics 74*, 1 (1997), 33–55.

[85] Li, Y., Nguyen, H. L., and Woodruff, D. P. On sketching matrix norms and the top singular vector. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)* (2014), pp. 1562–1581.

[86] Linial, N. Locality in distributed graph algorithms. *SIAM Journal on Computing 21*, 1 (1992), 193–201.

[87] Lovász, L. On determinants, matchings, and random algorithms. In *Proceedings of the 2nd Conference on Fundamentals of Computation Theory (FCT)* (1979), pp. 565–574.

[88] Lovász, L., and Plummer, M. *Matching theory*, vol. 367. American Mathematical Soc., 2009.

[89] Luby, M. A simple parallel algorithm for the maximal independent set problem. *SIAM Journal on Computing 15*, 4 (1986), 1036–1053.

[90] Luby, M., Naor, J., and Naor, M. On removing randomness from a parallel algorithm for minimum cuts (extended abstract). Tech. rep., 1993.

[91] Mahadev, N. V. R., and Peled, U. N. *Threshold graphs and related topics*. Annals of Discrete Mathematics. Elsevier, 1995.

[92] Masek, W. A fast algorithm for the string editing problem and decision graph complexity. Master's thesis, MIT, 1976.

[93] McGregor, A. Open problems in sublinear algorithms: Problem 64. `http://sublinear.info/64`. (Retrieved: July 2015).

[94] McGregor, A. Finding graph matchings in data streams. In *Proceedings of the 9th Workshop on Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)* (2005), pp. 170–181.

[95] McGregor, A. Graph stream algorithms: a survey. *SIGMOD Record 43*, 1 (2014), 9–20.

[96] Meer, K., and Rautenbach, D. On the OBDD size for graphs of bounded tree- and clique-width. *Discrete Mathematics 309*, 4 (2009), 843–851.

[97] Meinel, C., and Theobald, T. On the influence of the state encoding on OBDD-representations of finite state machines. *Informatique théorique et applications 33*, 1 (1999), 21–32.

[98] Mertzios, G. B. A matrix characterization of interval and proper interval graphs. *Applied Mathematics Letters 21*, 4 (2008), 332–337.

[99] Métivier, Y., Robson, J. M., Saheb-Djahromi, N., and Zemmari, A. An optimal bit complexity randomized distributed MIS algorithm. *Distributed Computing 23*, 5-6 (2011), 331–340.

[100] Micali, S., and Vazirani, V. V. An $\mathcal{O}(\sqrt{|V|}|E|)$ algorithm for finding maximum matching in general graphs. In *Proceedings of the 21st Annual IEEE Symposium on Foundations of Computer Science (FOCS)* (1980), pp. 17–27.

[101] Mitzenmacher, M., and Upfal, E. *Probability and computing: Randomized algorithms and probabilistic analysis*. Cambridge University Press, 2005.

[102] Mucha, M., and Sankowski, P. Maximum matchings via Gaussian elimination. In *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS)* (2004), pp. 248–255.

[103] Muthukrishnan, S. Data streams: Algorithms and applications. *Foundations and Trends in Theoretical Computer Science 1*, 2 (2005).

[104] Naor, J., and Naor, M. Small-bias probability spaces: Efficient constructions and applications. *SIAM Journal on Computing 22*, 4 (1993), 838–856.

[105] Negruseri, Pasoi, C. S., Stanley, M. B., Stein, B., C, and Strat, C. G. Solving maximum flow problems on real world bipartite graphs. *ACM Journal of Experimental Algorithmics 16* (2011).

[106] Nisan, N. Pseudorandom generators for space-bounded computation. *Combinatorica 12*, 4 (1992), 449–461.

[107] Nunkesser, R., and Woelfel, P. Representation of graphs by OBDDs. *Discrete Applied Mathematics 157*, 2 (2009), 247–261.

[108] Olariu, S. An optimal greedy heuristic to color interval graphs. *Information Processing Letters 37*, 1 (1991), 21–25.

[109] Otter, R. The number of trees. *Annals of Mathematics 49*, 3 (1948), 583–599.

[110] Panconesi, A., and Srinivasan, A. Randomized distributed edge coloring via an extension of the chernoff–hoeffding bounds. *SIAM Journal on Computing 26*, 2 (1997), 350–368.

[111] Peled, U. N., and Sun, F. Enumeration of difference graphs. *Discrete Applied Mathematics 60*, 1-3 (1995), 311–318.

[112] Saitoh, T., Otachi, Y., Yamanaka, K., and Uehara, R. Random generation and enumeration of bipartite permutation graphs. *Journal of Discrete Algorithms 10*, 0 (2012), 84–97.

[113] Saitoh, T., Yamanaka, K., Kiyomi, M., and Uehara, R. Random generation and enumeration of proper interval graphs. *IEICE Transactions on Information and Systems 93-D*, 7 (2010), 1816–1823.

[114] Savický, P. Improved boolean formulas for the Ramsey graphs. *Random Structures and Algorithms 6*, 4 (1995), 407–416.

[115] Sawitzki, D. Implicit flow maximization by iterative squaring. In *Proceedings of the 30th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM)* (2004), pp. 301–313.

[116] Sawitzki, D. The complexity of problems on implicitly represented inputs. In *Proceedings of the 32nd International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM)* (2006), pp. 471–482.

[117] Sawitzki, D. Exponential lower bounds on the space complexity of OBDD-based graph algorithms. In *Proceedings of the 7th Latin American Symposium (LATIN)* (2006), pp. 781–792.

[118] Sawitzki, D. Implicit simulation of FNC algorithms. *Electronic Colloquium on Computational Complexity (ECCC) 14*, 028 (2007).

[119] Siegel, A. On universal classes of extremely random constant-time hash functions. *SIAM Journal on Computing 33*, 3 (2004), 505–543.

[120] Sieling, D., and Wegener, I. NC-algorithms for operations on binary decision diagrams. *Parallel Processing Letters 3* (1993), 3–12.

[121] Spinrad, J., Brandstädt, A., and Stewart, L. Bipartite permutation graphs. *Discrete Applied Mathematics 18*, 3 (1987), 279–292.

[122] Stothers, A. *On the Complexity of Matrix Multiplication.* PhD thesis, University of Edinburgh, 2010.

[123] Takaoka, A., Tayu, S., and Ueno, S. OBDD representation of intersection graphs. *IEICE Transactions on Information and Systems 98*, 4 (2015), 824–834.

[124] Tutte, W. T. The factorization of linear graphs. *Journal of the London Mathematical Society 22* (1947), 107–111.

[125] Uehara, R., and Chen, Z.-Z. Parallel approximation algorithms for maximum weighted matching in general graphs. *Information Processing Letters 76*, 1-2 (2000), 13–17.

[126] Verbin, E., and Yu, W. The streaming complexity of cycle counting, sorting by reversals, and other problems. In *Proceedings of the 22th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)* (2011), pp. 11–25.

[127] Wegener, I. The size of reduced OBDDs and optimal read-once branching programs for almost all boolean functions. *IEEE Transactions on Computers 43*, 11 (1994), 1262–1269.

[128] Wegener, I. *Branching programs and binary decision diagrams*. SIAM Monographs on Discrete Mathematics and Applications, 2000.

[129] Wegman, M. N., and Carter, J. New hash functions and their use in authentication and set equality. *Journal of Computer and System Sciences 22*, 3 (1981), 265–279.

[130] Williams, V. V. Multiplying matrices faster than Coppersmith-Winograd. In *Proceedings of the 44th Annual ACM Symposium on Theory of Computing (STOC)* (2012), pp. 887–898.

[131] Woelfel, P. Symbolic topological sorting with OBDDs. *Journal of Discrete Algorithms 4* (2006), 51–71.

[132] Yao, A. C.-C. Some complexity questions related to distributive computing (preliminary report). In *Proceedings of the 11th Annual ACM Symposium on Theory of Computing (STOC)* (1979), STOC '79, pp. 209–213.

[133] Zelke, M. Weighted matching in the semi-streaming model. *Algorithmica 62*, 1-2 (2012), 1–20.

[134] Zomaya, A. Y. H., Ed. *Parallel and distributed computing handbook*. McGraw-Hill, Inc., 1996.

A. Some Mathematical Facts

A.1. Useful Inequalities

It is well known that the binomial coefficient $\binom{n}{k}$ can be bounded above by $2^{n \cdot H(k/n)}$ where $H(x) = -x\log(x) - (1-x)\log(1-x)$ is the binary entropy function. A proof of an even stronger bound can be found in Lemma 16.19. from [47]: For $n \geq 1$ and $0 < \varepsilon \leq 1/2$ it is

$$\sum_{i=0}^{\lfloor \varepsilon \cdot n \rfloor} \binom{n}{i} \leq 2^{H(\varepsilon) \cdot n}.$$

It directly implies the bound on the logarithm of a binomial coefficient.

Lemma A.1.1. *For $n \geq 1$ and $0 < \varepsilon \leq 1/2$ it is*

$$\log \binom{n}{\lfloor \varepsilon \cdot n \rfloor} \leq n \cdot H(\varepsilon).$$

Next, we show two useful inequalities for calculations with fractions.

Lemma A.1.2. *Let $a, b, c \in \mathbb{N}$. If $a \geq b$ then*

$$\frac{a+c}{b+c} \leq \frac{a}{b}$$

and if $a \leq b$ then

$$\frac{a+c}{b+c} \geq \frac{a}{b}.$$

Proof. In the first case of $a \geq b$ we have

$$a \geq b \Leftrightarrow ac + ab \geq bc + ab \Leftrightarrow \frac{a}{b} \geq \frac{a+c}{b+c}.$$

The same calculation also proves the second statement. □

A.2. Probability Theory

A random variable follows the hypergeometric distribution if

$$\mathbf{Pr}\left[X = k\right] = \frac{\binom{K}{k} \cdot \binom{N-K}{n-k}}{\binom{N}{n}}$$

where $N \in \mathbb{N}$, $n, K \in \{0, \ldots, N\}$. The hypergeometric distribution describes the probability of k successes in n draws without replacement from a population of size N that contains K successes.

The following three theorems present useful inequalities for bounding probabilities or expectations of random variables.

Theorem A.2.1 (Markov's Inequality)**.** *Let X be a nonnegative random variable and $a > 0$, then*

$$\mathbf{Pr}[X \geq a] \leq \frac{\mathbf{E}[X]}{a}.$$

Theorem A.2.2 (Reverse Markov Inequality [64])**.** *Let X be a random variable with $0 \leq X \leq b$ and $\mathbf{E}[X] \geq \frac{b}{a}$ for some $b \in \mathbb{N}$ and $a > 1$. Then for any $\alpha > 0$*

$$\mathbf{Pr}[X \geq (1-\alpha) \cdot \mathbf{E}[X]] \geq \frac{\alpha}{a}.$$

Proof. The proof is very similar to the proof of Markov's inequality:

$$\begin{aligned} \mathbf{E}[X] &= \sum_{i=0}^{b} i \cdot \mathbf{Pr}[X = i] = \sum_{i<(1-\alpha)\cdot\mathbf{E}[X]} i \cdot \mathbf{Pr}[X = i] + \sum_{i\geq(1-\alpha)\cdot\mathbf{E}[X]} i \cdot \mathbf{Pr}[X = i] \\ &\leq (1-\alpha)\mathbf{E}[X] + b \cdot \mathbf{Pr}[X \geq (1-\alpha)\mathbf{E}[X]] \end{aligned}$$

Solving this inequality for $\mathbf{Pr}[X \geq (1-\alpha)\mathbf{E}[X]]$ gives a lower bound of $\alpha \cdot \mathbf{E}[X]/b \geq \alpha/a$. □

Theorem A.2.3 (Chernoff Bound)**.** *Let $X_1, \ldots, X_n$ be independent binary random variables. Define $X = \sum X_i$ and let $\mu = \mathbf{E}[X]$. Then for any $\delta \in (0, 1)$*

$$\mathbf{Pr}[X \geq (1+\delta)\mu] \leq \exp\left(-\delta^2\mu/3\right)$$

and

$$\mathbf{Pr}[X \leq (1-\delta)\mu] \leq \exp\left(-\delta^2\mu/2\right).$$

For any $\delta > 0$ it is

$$\mathbf{Pr}[X \geq (1+\delta)\mu] \leq \left(\frac{\exp(\delta)}{(1+\delta)^{1+\delta}}\right)^{\mu}.$$

Theorem A.2.4 (Jensen's Inequality)**.** *If f is a convex function, then for any random variable X it holds*

$$\mathbf{E}[f(X)] \geq f(\mathbf{E}[X]).$$

Proofs of the Chernoff bound and Jensen's inequality can be found in [101]. As a corollary of the Chernoff bounds we have

Corollary A.2.5. *For any $\delta \geq 1$ it is*

$$\mathbf{Pr}\left[X \geq (1+\delta)\mu\right] \leq \exp(-\delta\mu/3).$$

Proof. Using Theorem A.2.3 we have

$$\mathbf{Pr}\left[X \geq (1+\delta)\mu\right] \leq \left(\frac{\exp(\delta)}{(1+\delta)^{1+\delta}}\right)^{\mu}.$$

Thus, it remains to prove that $\left(\frac{\exp(\delta)}{(1+\delta)^{1+\delta}}\right)^{\mu} \leq \exp(-\delta\mu/3)$ which is equivalent to

$$f(\delta) := \delta - (1+\delta)\ln(1+\delta) + \delta/3 \leq 0.$$

It is $f(1) < 0$ and $f'(\delta) = 4/3 - \frac{1+\delta}{1+\delta} - \ln(1+\delta) = 1/3 - \ln(1+\delta)$. Since $f'(\delta) < 0$ for $\delta \geq 1$, we have $f(\delta) < 0$ for all $\delta \geq 1$. □

B. Experimental Evaluation

B.1. Randomized and Deterministic Implicit Maximal Matching Algorithms

Number	Running Time (RM)	Running Time (HS)	SBDD Size (RM)	SBDD Size (HS)
0	**0.243**	0.475	**567210**	1346996
1	**0.264**	0.571	**555968**	1394008
2	**0.256**	0.567	**553924**	1394008
3	**0.059**	0.066	**153300**	220752
4	0.055	**0.041**	**161476**	194180
5	**0.042**	0.043	**153300**	194180
6	**0.064**	0.067	**196224**	252434
7	**0.042**	0.073	**163520**	279006
8	**0.055**	0.072	**169652**	279006
9	**0.09**	0.12	**240170**	368942
10	**0.099**	0.11	**237104**	368942
11	**0.105**	0.17	**245280**	368942
12	0.067	**0.052**	**236082**	245280
13	**0.058**	0.07	**242214**	310688
14	0.091	**0.066**	**284116**	328062
15	**2.565**	6.259	**3115056**	7887796

Table B.1.: Running times and space usage of RM and HS on real-world instances from [105].

Number	Running Time (RM)	Running Time (HS)	SBDD Size (RM)	SBDD Size (HS)
16	**2.545**	6.167	**3115056**	7874510
17	**4.002**	6.329	**3115056**	7874510
18	**1.112**	1.81	**2053198**	2320962
19	**0.913**	2.043	**2035824**	2485504
20	**0.828**	1.931	**2035824**	2485504
21	0.073	**0.036**	**182938**	231994
22	0.059	**0.046**	**163520**	240170
23	0.095	**0.036**	**162498**	240170
24	0.043	**0.022**	134904	134904
25	0.037	**0.021**	135926	135926
26	0.058	**0.018**	169652	**135926**
27	**0.203**	0.331	**346458**	731752
28	**0.188**	0.348	**317842**	677586
29	**0.244**	0.305	**319886**	677586
30	**0.632**	1.176	**1314292**	2100210
31	**0.568**	1.114	**1280566**	2104298
32	**0.458**	0.568	**950460**	1410360

Table B.2.: Running times and space usage of RM and HS on real-world instances from [105].

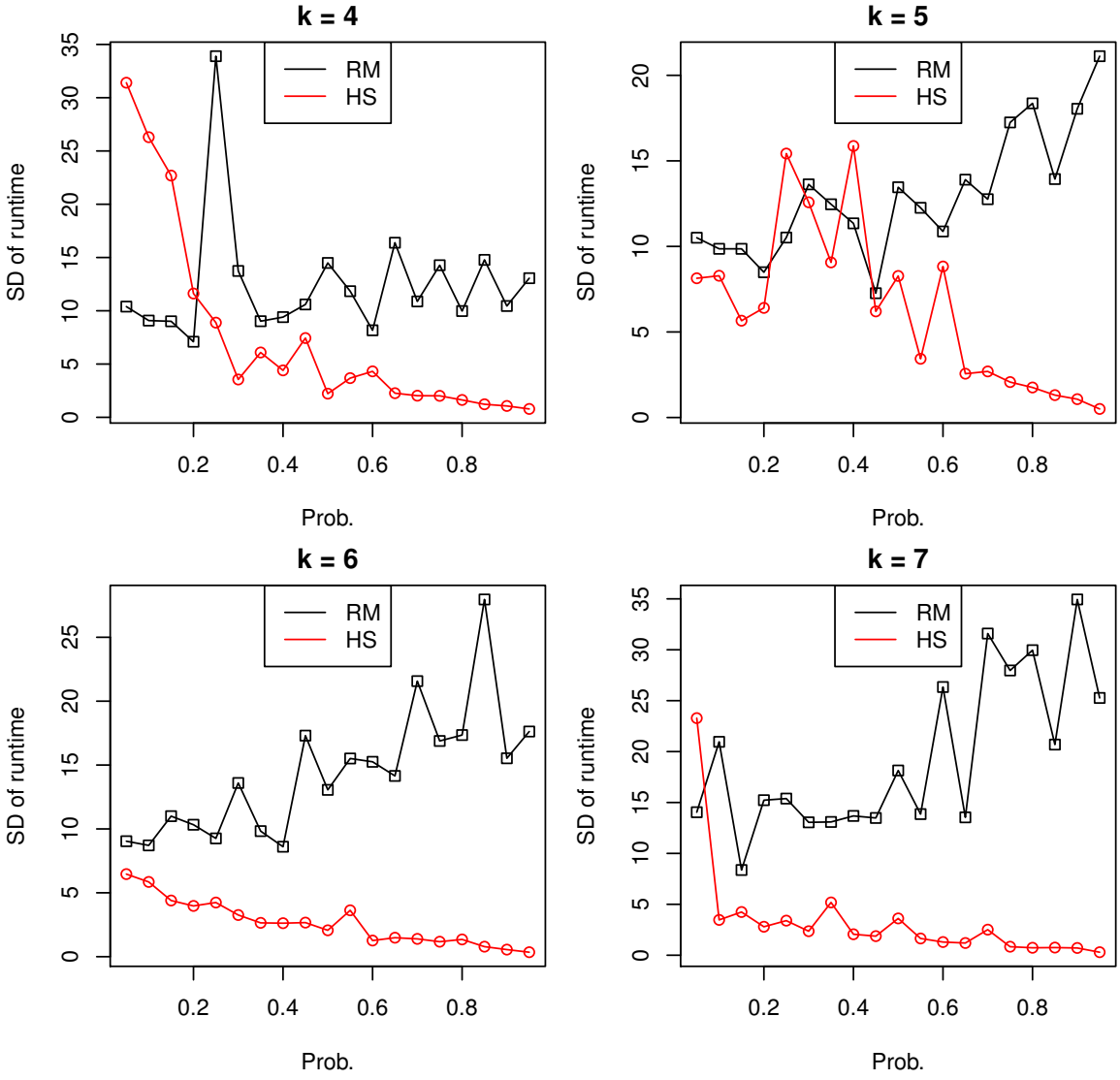


Figure B.1.: Standard deviations of the running times.

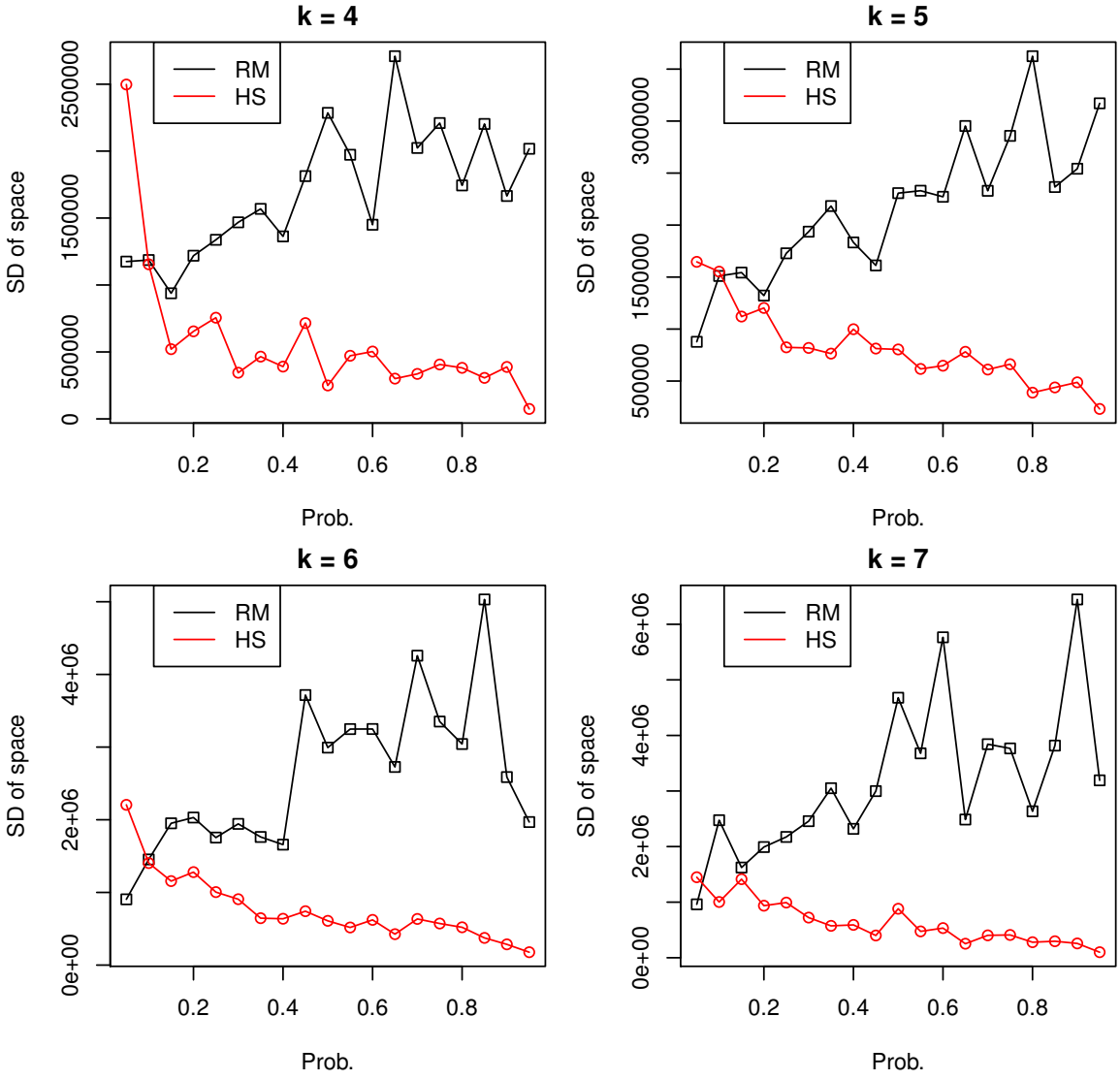


Figure B.2.: Standard deviation of the space usage.

B.2. Randomized and Deterministic Implicit Minimum Spanning Tree Algorithms

B.2.1. MST Instances

	Name	Size	Edges		Name	Size	Edges
1	saylr3	1000	1375	41	mhd1280b	1280	10749
2	sherman1	1000	1375	42	plbuckle	1282	14681
3	tub1000	1000	1498	43	jagmesh9	1349	3876
4	dwt_1005	1005	3808	44	rail_1357	1357	3814
5	dwt_1007	1007	3784	45	jagmesh6	1377	3808
6	jagmesh2	1009	2928	46	ex23	1409	21147
7	lshp1009	1009	2928	47	spiral	1434	8397
8	cage8	1015	4994	48	jagmesh4	1440	4032
9	delaunay_n10	1024	3056	49	msc01440	1440	22415
10	M20PI_n1	1028	1033	50	bcspwr06	1454	1923
11	S20PI_n1	1028	1033	51	bcsstm12	1473	9093
12	orsirr_1	1030	2914	52	bcsstk11	1473	16384
13	stufe	1036	1868	53	bcsstk12	1473	16384
14	rajat04	1041	4317	54	qh1484	1484	2492
15	msc01050	1050	14053	55	lshp1561	1561	4560
16	can_1054	1054	5571	56	netscience	1589	2742
17	can_1072	1072	5686	57	ex4	1601	15349
18	bcsstk08	1074	5943	58	bcspwr07	1612	2106
19	lock1074	1074	25275	59	bcspwr08	1624	2213
20	bcsstk09	1083	8677	60	ex7	1633	26455
21	bcsstk10	1086	10492	61	ex6	1651	23941
22	bcsstm10	1086	10503	62	filter2D	1668	4541
23	jagmesh3	1089	3136	63	bcspwr09	1723	2394
24	sherman4	1104	1341	64	ex33	1733	10228
25	email	1133	5451	65	bcsstk14	1806	30824
26	1138_bus	1138	1458	66	adder_trans_01	1814	6387
27	jagmesh7	1138	3156	67	adder_trans_02	1814	6387
28	jagmesh8	1141	3162	68	ex3	1821	25432
29	ex32	1159	5092	69	nasa1824	1824	18692
30	eris1176	1176	8688	70	ukerbe1_dual	1866	3538
31	jagmesh5	1180	3285	71	rajat12	1879	5528
32	M20PI_n	1182	1198	72	lshp1882	1882	5511
33	S20PI_n	1182	1198	73	plsk1919	1919	4831
34	fpga_trans_01	1220	3114	74	plat1919	1919	15240
35	fpga_trans_02	1220	3114	75	bcsstk26	1922	14207
36	bcsstk27	1224	27451	76	rajat02	1960	4621
37	bcsstm27	1224	27451	77	netz4504	1961	2578
38	dwt_1242	1242	4592	78	bwm2000	2000	2998
39	rdb1250	1250	3025	79	Trefethen_2000	2000	19953
40	lshp1270	1270	3699				

B.2.2. Deterministic Implicit MST Algorithms

The algorithms are using some priority functions for choosing the smallest edge out of a set of candidates. The definition of these functions is as follows. The function $\Pi_1((x,y,d),(x',y',d'))$ (which is used in [15]) is equal to 1 iff one of the following is fulfilled

- $|d| < |d'|$,
- $|d| = |d'|$ and $\min(|x|,|y|) < \min(|x'|,|y'|)$, or
- $|d| = |d'|$ and $\min(|x|,|y|) = \min(|x'|,|y'|)$ and $\max(|x|,|y|) < \max(|x'|,|y'|)$.

The second function is defined by

$$\begin{aligned}\Pi_2((x,y,d),(x',y',d')) &= |d| < |d'| \vee (|d| = |d'| \wedge |x| < |x'|) \\ &\quad \vee (|d| = |d'| \wedge |x| = |x'| \wedge |y| < |y'|).\end{aligned}$$

Now, we can present the three deterministic implicit minimum spanning tree algorithms.

Algorithm 15 Implicit Variant of Prim's Minimum Spanning Tree Algorithm

Input: Weighted graph $\chi_E(x,y,d)$ and $\chi_V(x))$
Output: Minimum spanning tree $MST(x,y)$

$R(x) = 0$, $MST(x,y) = 0$
while $R(x) \not\equiv \chi_V(x)$ **do**
  ▷ Determine start node and mark it as connected
  $Resid(x) = \chi_V(x) \wedge \overline{R(x)}$
  $R(x) = Resid(x) \wedge \overline{(\exists y)(Resid(y) \wedge (|x| < |y|))}$
  repeat
    ▷ Select candidates for insertion
    $T(x,y,d) = \chi_E(x,y,d) \wedge R(x) \wedge \overline{R(y)}$
    ▷ Select the smallest candidate
    $C(x,y) = (\exists d)(T(x,y,d) \wedge \overline{(\exists x',y',d')(T(x',y',d') \wedge \Pi_2((x',y',d'),(x,y,d)))})$
    ▷ Add selected edge
    $C(x,y) = C(x,y) \vee C(y,x)$
    $MST_{old}(x,y) = MST(x,y)$
    $MST(x,y) = MST(x,y) \vee C(x,y)$
    ▷ Mark node connected by the new edge
    $R(x) = R(x) \vee (\exists y) C(x,y)$
  until $MST_{old}(x,y) \equiv MST(x,y)$
end while
return $MST(x,y)$

Algorithm 16 Implicit Minimum Spanning Tree Algorithm by Bollig [15]

Input: Weighted graph $\chi_E(x, y, d)$ and $\chi_V(x))$
Output: Minimum spanning tree $MST(x, y)$

$MST(x, y) = 0$
repeat
  $\triangleright$ Compute connected components of the forest
  $R(x, y) = findTransitiveClosure(MST(x, y))$
  $\triangleright$ Choose smallest edges
  $C(x, y) = \exists d : \chi_E(x, y, d) \wedge \overline{R(x, y)} \wedge \overline{\exists y', z, d' : R(x, z) \wedge \chi_E(z, y', d') \wedge \overline{R(z, y')} \wedge \Pi_1((z, y', d'), (x, y, d))}$
  $C(x, y) = C(x, y) \vee C(y, x)$
  $MST_{old}(x, y) = MST(x, y)$
  $MST(x, y) = MST(x, y) \vee C(x, y)$
until $MST_{old}(x, y) \equiv MST(x, y)$
return $MST(x, y)$

Algorithm 17 Implicit Variant of Kruskal's Minimum Spanning Tree Algorithm

Input: Weighted graph $\chi_E(x, y, d)$ and $\chi_V(x))$
Output: Minimum spanning tree $MST(x, y)$

$MST(x, y) = 0$
$\triangleright$ Each node is a single connected component
$R(x, y) = \chi_V(x) \wedge (|x| = |y|)$
repeat
  $\triangleright$ Select all edges connecting nodes from different components
  $T(x, y, d) = \chi_E(x, y, d) \wedge \overline{R(x, y)}$
  $\triangleright$ Choose the smallest edge
  $E(x, y) \leftarrow (\exists d)(T(x, y, d) \wedge \overline{(\exists x', y', d')(T(x', y', d') \wedge \Pi_2((x', y', d'), (x, y, d)))})$
  $E(x, y) \leftarrow E(x, y) \vee E(y, x)$
  $\triangleright$ Update information about connected nodes
  $R(x, y) = R(x, y) \vee (\exists x', y')(R(x, y') \wedge E(y', x') \wedge R(x', y))$
  $MST_{old}(x, y) = MST(x, y)$
  $MST(x, y) = MST(x, y) \vee E(x, y)$
until $MST_{old}(x, y) \equiv MST(x, y)$
return $MST(x, y)$