PSEUDOFOREST PARTITIONS AND THE APPROXIMATION OF CONNECTED SUBGRAPHS OF HIGH DENSITY

Dissertation
zur Erlangung des Grades
„Doktor der Naturwissenschaften"
am Fachbereich Physik, Mathematik und Informatik
der Johannes Gutenberg-Universität in Mainz

vorgelegt von

Markus Andreas Daniel Blumenstock

geboren in Frankfurt am Main

Mainz, den 8. Januar 2020
Typographisch korrigierte Fassung vom 19. Juni 2020

ERSTBERICHTERSTATTER:
Prof. Dr. Ernst Althaus
ZWEITBERICHTERSTATTER:
Prof. Dr. Andreas Hildebrandt
DRITTBERICHTERSTATTER:
Prof. Dr. Ulrich Meyer

Datum des Kolloquiums: 12.11.2019 D77

ZUSAMMENFASSUNG

In dieser Arbeit werden das Problem, in einem einfachen Graphen einen Subgraphen mit maximaler Dichte d^* zu finden, sowie verwandte Probleme diskutiert. Die ganzzahlige Variante des dazu dualen Problems ist, eine Orientierung mit kleinstmöglichem maximalen Eingangsgrad zu finden. Dieser Eingangsgrad ist gleich der kleinsten Anzahl an Pseudowäldern, in die der Graph zerlegt werden kann, und wird daher als Pseudoarborizität p bezeichnet.

Es wird gezeigt, wie Kowaliks Approximationsschema dazu verwendet werden kann, die balancierte binäre Suche des Algorithmus von Gabow und Westermann zu beschleunigen und eine Laufzeit von $\mathcal{O}(m^{3/2}\sqrt{\log\log p})$ für die Bestimmung von p zu erreichen, wobei m die Anzahl der Kanten des Graphen ist. Ein Subgraph mit Dichte größer $\lceil d^* \rceil - 1$ kann mit derselben asymptotischen Laufzeit bestimmt werden. Es werden zudem Laufzeiten verschiedener Algorithmen zur Bestimmung von p experimentell verglichen. Mit aktueller Hardware können die Berechnungen für soziale Netzwerke mit hunderten Millionen Kanten in wenigen Minuten ausgeführt werden.

Aufbauend auf Kowaliks Approximationsschema wird ein Approximationsschema für die Arborizität eines Graphen mit derselben Laufzeit entwickelt. Im Gegensatz zu früheren Ansätzen approximiert dieses nicht nur den Wert der Arborizität, sondern berechnet auch eine dazugehörige Zerlegung in Wälder. Dies wird durch eine schnelle Konvertierung von k Pseudowäldern in $k+1$ Wälder erreicht.

Basierend auf dem Konzept der maximalen Dichte kann ein gemischt-ganzzahliges lineares Programm (MILP) aufgestellt werden, um zusammenhängende Subgraphen zu finden, die eine lineare Funktion minimieren. Dieses Problem, das eng mit dem Steinerbaumproblem auf Graphen verwandt ist, ist im Allgemeinen NP-schwer. Für den Fall, dass die Anzahl der Knoten des Subgraphen genau k betragen soll, wird das MILP mit einer anderen bestehenden Formulierung, den *generalized subtour elimination constraints* (*GSEC*), sowohl theoretisch als auch experimentell verglichen.

Eine weiteres Thema, das beleuchtet wird, ist die Approximation des Steinerbaumproblems auf Graphen mithilfe der *bidirected cut relaxation* (*BCR*), welche äquivalent zu *GSEC* ist. Es wird gezeigt, dass der Algorithmus von Byrka et al. eine 1,354-Approximation berechnet, wenn die Instanzen keine Steinerklauen enthalten. Es wird zudem eine Idee zur Generierung schwieriger Instanzen vorgestellt, die jedoch nicht in verbesserten unteren Schranken an das *integrality gap* von *BCR* resultiert.

ABSTRACT

This thesis deals with the problem of finding a subgraph of maximum density d^* in a simple graph and related problems. The integral variant of the dual problem is to find an orientation with the smallest possible maximum indegree. This indegree is equal to the smallest number p of pseudoforests into which the graph can be partitioned, and is therefore called pseudoarboricity.

It is shown that Kowalik's approximation scheme can be employed to accelerate the balanced binary search of the Gabow–Westermann algorithm in order to obtain a runtime of $\mathcal{O}(m^{3/2}\sqrt{\log\log p})$ for determining p, where m is the number of edges in the graph. A subgraph of density greater than $\lceil d^* \rceil - 1$ can be determined within the same asymptotic runtime. In addition, the runtimes of several algorithms for determining p are compared experimentally. On current hardware, the computations can be carried out in minutes for social networks with hundreds of millions of edges.

Building upon Kowalik's approximation scheme, an approximation scheme for the arboricity with the same asymptotic runtime is developed. In contrast to previous approaches, it does not only approximate the value of the arboricity, but also computes a corresponding forest partition. This is achieved using a fast conversion of k pseudoforests into $k+1$ forests.

Based on the concept of maximum density, a mixed integer linear program (MILP) can be formulated for finding a connected subgraph that minimizes a linear function. This problem, which is closely related to the Steiner tree problem on graphs, is NP-complete in general. For the case where the number of vertices is required to be exactly k, the MILP is compared to an existing formulation, the generalized subtour elimination constraints (*GSEC*), both in theory and in practice.

Another topic under scrutiny is the approximation of the Steiner tree problem on graphs with the bidirected cut relaxation (*BCR*), which is equivalent to *GSEC*. It is shown that the algorithm of Byrka et al. computes a 1.354-approximation if the instances do not contain Steiner claws. Furthermore, an idea for generating hard instances is presented, yet it does not result in improved lower bounds on the integrality gap of *BCR*.

ACKNOWLEDGEMENTS

First and foremost, I want to thank my advisor, Professor Ernst Althaus. He always had a friendly ear to my questions and valuable advice, and he let me freely pursue my interests in both research and teaching to a remarkable degree.

I am also indebted to the supervisor of my three-month stay at the University of Waterloo in Canada, Professor Jochen Könemann. He took a lot of time to meet with me and discuss ideas, and invited me to a workshop at the Banff International Research Station.

My colleagues at the Institute of Computer Science in Mainz also deserve credit for the wonderful working climate and the occasional get-together. I want to highlight the members of the Algorithmics group: Björn Beber, Domenico Mosca, Udo Muttray, and Sarah Ziegler, as well as 'honorary member' Frank Fischer.

Last, but not least, I want to thank my wife Katharina for her support and our son Adrian, who fills both our lives with joy.

CONTENTS

1 INTRODUCTION

If in a quadrilateral one asks for [. . .]
the shortest connection system in the plane, then [. . .]
you obtain quite an interesting mathematical problem [. . .];
as a matter of fact, I have on occasion considered the rail road connection
between Harburg, Bremen, Hannover and Braunschweig,
and I myself have thought that this problem would be
an excellent prize problem for our students.

— Carl F. Gauss, letter to Heinrich C. Schumacher (1836)
Translated from German [Bra+14]

1.1 HISTORICAL BACKGROUND AND MOTIVATION

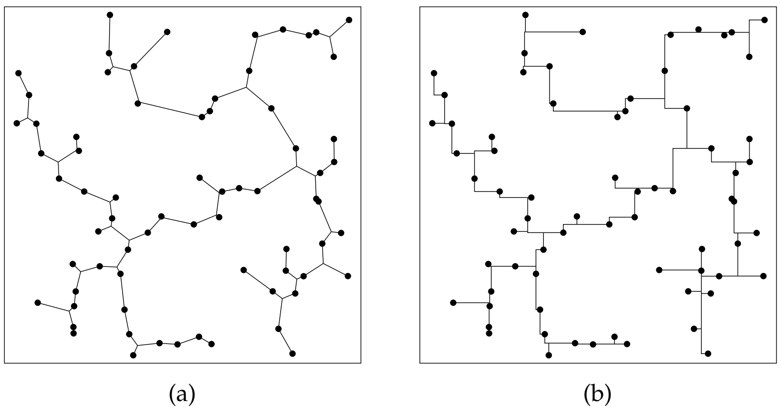


Figure 1.1: Examples of Steiner trees in the plane for a set of 70 randomly generated terminals (dots). The images were created with the GeoSteiner software package [Juh+18]. (a) A Steiner tree for Euclidean distances. (b) A Steiner tree for rectilinear distances.

Given a finite set of points in the plane, the Euclidean Steiner tree problem is to connect them with lines of minimum total Euclidean length. In order to do so, one may choose additional points where lines can end. These points are called Steiner points. The connecting lines in an optimum solution form a tree. An example can be seen in Figure 1.1a. The problem, which dates from the 19th century [Bra+14], was shown to be NP-hard [GGJ77], but is not known to be in NP. The rectilinear variant of the problem, where the Euclidean distance is replaced by the L_1 distance (Figure 1.1b), is NP-complete when the point coordinates are required to be integers [GJ77]. It has applications in circuit design, see [GJ77; FR83] for the relevant literature.

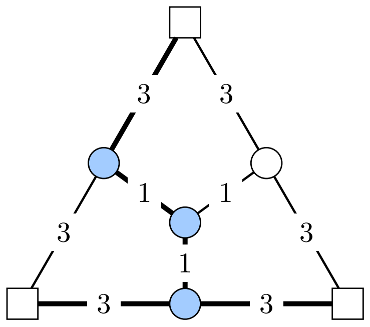


Figure 1.2: An instance of the Steiner tree problem in graphs. The squares are terminal vertices that must be selected, the circles are the optional Steiner vertices. An optimal solution of cost eleven is shown with bold edges and blue Steiner vertices. Solutions that do not use the Steiner vertex in the center have cost of at least twelve.

The related Steiner tree problem in graphs is one of the best-studied problems in computer science. The goal is to find a tree of minimum weight spanning a set of given terminal vertices in an undirected graph with nonnegative edge weights. Here, the non-terminal vertices are the optional Steiner vertices. An example can be seen in Figure 1.2. The problem is a generalization of the minimum spanning tree problem, in which all vertices are to be spanned. The latter was first described in 1926 by Borůvka [Bor26] with an application to power grids in Moravia in mind. While the minimum spanning tree problem can be solved in polynomial time, the Steiner tree problem in graphs is NP-complete [PS02]. There are several integer linear programs (ILPs) that model the problem [GM93; PD03].

The main ingredient of these ILPs is modeling the connectivity of the subgraph defined by the solution's assignment of vertex and edge variables. For this reason, they can be easily adapted to search for connected subgraphs that minimize (or maximize) a linear function, a problem that is in general also NP-complete. This problem has applications in bioinformatics, see for example [Ide+02; Bac+11; Alt+14], and these applications were the initial motivation for this thesis. In this area, the vertices typically represent genes or proteins, and the edge or vertex weights are determined from gene expression values [Sub+05; Din+08; Gei+11]. In some settings (e.g., [Bac+11]), the input graph is directed, and the vertices in the solution should be reachable in the directed sense from a root vertex. This root vertex is interpreted as the so-called *key player* in a regulatory cascade. There are other approaches for identifying key players or otherwise 'biologically central' genes, for example via the dominating set problem [Mil+11; Naz+16] or finding paths of a specified length between certain proteins of interest [Zha+08].

Another related and well-known NP-complete problem that also generalizes the minimum spanning tree problem is the *k*-cardinality

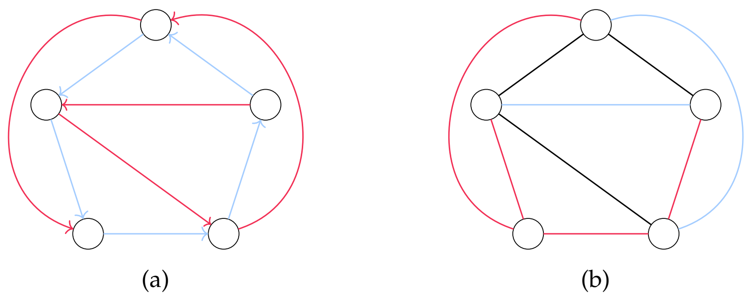


Figure 1.3: (a) A partition of a simple graph into two pseudoforests, indicated by red and blue edges. An orientation of the graph with maximum indegree two is given by arrowheads. (b) A partition of the same graph into three forests, indicated in red, blue, and black. A partition into two forests is not possible.

tree (or k-MST) problem. It asks for a tree of k vertices with minimum total edge weight. It has applications in oil field leasing and facility layout planning [Ehr+97].

NP-complete problems can be solved in polynomial time if and only if the complexity classes P and NP are equal, which is widely believed not to be true [Aar17]. Hence, one may turn to approximation algorithms that run in polynomial time. For the Steiner tree problem in graphs, the currently best approximation algorithm has an approximation factor of $\ln(4) + \epsilon < 1.387$ [Byr+13]. It is based on the hypergraphic relaxation (*HYP*) [War98; PD03]. Better approximation factors can be given in (uniformly) quasi-bipartite instances [Grö+02; Fun+12; Byr+13], where the hypergraphic relaxation is as strong as the well-known bidirected cut relaxation (*BCR*) [CKP10a; Goe+12; Fun+12]. *BCR* has great practical value, but is not well-understood. Its integrality gap is known to lie between 1.16 and 2 [Byr+13].

Since solving above problems is NP-hard, another avenue for practical considerations is to find connected subgraphs of maximum density. This so-called densest subgraph problem is solvable in polynomial time, even with nonnegative vertex and edge weights [Gol84]. The problem and its variants have been applied to clustering and community detection [AC09; DGP09; SG10; Ang+14; ELS15], as well as bioinformatics [Sah+10; Ma+17]. Curiously, the dual problem can be used as a basis for a mixed-integer linear programming (MILP) formulation for the Steiner tree problem in graphs and related problems [Coh10; Alt+14]. In contrast to previous approaches, this MILP formulation requires only a linear number of constraints and variables. This connection lead us to investigate the following problems. The dual problem of the densest subgraph problem is the smallest maximum indegree fractional orientation problem [Cha00a]. Its integral variant, which can be used to create data structures for edge membership queries in a graph [KNR92; CE91; AAR95; AMZ97; BF99], is equivalent to finding a partition of the graph into the smallest possible number of pseudoforests [Bez00; Kow06]. This number is called the pseudoarboricity. An example of an optimal pseudoforest partition can be seen in Figure 1.3a on the preceding page. The pseudoarboricity problem is a special case of the matroid partitioning problem and was, along with the analogously defined arboricity for forest partitions, addressed specifically by Picard and Queyranne [PQ82] and Gabow and Westermann [Wes88; GW92; Gab98]. The strong relationship between pseudoforest and forest partitions can be exploited algorithmically [Wes88; GW92; Kow06]. While there is an approximation scheme for the pseudoarboricity, previous approximation schemes [PST91; Kow06; TG16] for the arboricity were nonconstructive, i.e., only its value was approximated and no corresponding forest partition was computed. Figure 1.3b on the previous page shows an optimal forest partition of a graph.

The arboricity is an often-used measure for the density of a graph. Many results are stated for graphs of bounded arboricity, where some NP-complete problems become tractable [AG08; ELS13]. For several algorithms, it is possible to show better runtime estimates [CN85; GV08; ELS13] or approximation factors [BU17]. However, only a minority of algorithms actually use forest partitions explicitly.[1] Distributed algorithms that do use forest partitions exist for the maximal independent set and graph coloring problems [BE10] and the minimum dominating set problem [LW10].

1.2 STRUCTURE AND OUTLINE OF THE THESIS

This thesis deals with two classes of problems regarding connected subgraphs: polynomial-time solvable problems and NP-complete problems. Well-known definitions and theorems required for the thesis are given in Chapter 2. These are from graph theory, probability theory, complexity theory, algorithm design, linear algebra, matroid theory, linear programming, and network flow theory. They are typically covered to a large extent in computer science curricula and repeated here for completeness. It is assumed the reader is familiar with set theory and calculus on the undergraduate level. The theory of almost unit capacity networks in Section 2.15 was published in [Blu16] as a result of the writing of this thesis. These results are, however, straightforward generalizations of theorems by Even and Tarjan [ET75].

Chapters 3 to 12 deal with polynomial-time solvable problems: the densest subgraph problem, the smallest maximum indegree orientation problem, and the pseudoarboricity and arboricity problems.

In Chapters 3 and 4, we give an extensive review of the literature on the densest subgraph problem and its dual, the smallest max-

1 In some papers (e.g., [AMZ97]), forest partitions are used in order to obtain orientations of small maximum indegree, but this could be achieved from pseudoforest partitions.

imum (fractional) indegree orientation problem. We exhibit some mistakes made in these papers, and complement important results by investigating the relationship of various approaches and giving alternative proofs. We recast the balanced binary search technique, which was used by Gabow and Westermann [Wes88; GW92] for the pseudoarboricity problem, in terms of flows for the smallest maximum indegree orientation problem in Chapter 5. This technique reduces the logarithmic factor incurred by a binary search. Then, using Kowalik's approximation scheme [Kow06], we shrink the search interval and reduce this factor even further in Chapter 6.

In Chapter 7, we concern ourselves with a linear-time algorithm that can compute a $1/2$-approximation for the densest subgraph problem and 2-approximations for the pseudoarboricity and arboricity problems [Epp94; AAR95; AMZ97; Cha00a; Bez00; GP07]. We show that a modification of this algorithm for the pseudoarboricity problem [Asa+07], which has a slightly better approximation factor and was previously known to run in quadratic time, can be implemented in linear time. We also reduce the runtime of Kowalik's approximation scheme for fixed $\epsilon > 0$ by terminating the binary search early.

The smallest maximum indegree orientation problem is equivalent to the pseudoarboricity problem, which is closely related to the arboricity problem. These latter two are special cases of the covering problem for matroids. They are discussed in Chapter 8.

We propose conversions of a partition of k pseudoforests into a partition of $k+1$ forests in Chapters 9 and 10: For $k \leq 3$, the conversion runs in linear time. For $k \geq 4$, we give an algorithm that runs in near-linear time when k is fixed. This improves upon the divide-and-conquer conversion by Gabow and Westermann [Wes88; GW92]. We also show that one of the runtime analyses of their algorithm is erroneous. For every fixed $\epsilon > 0$, our results imply a constructive near-linear time $(1+\epsilon)$-approximation algorithm for the arboricity.

In Chapter 11, we use the 2-approximation algorithm from Chapter 7 to obtain a linear-time preprocessing that reduces the graph size while preserving the maximum density and (pseudo-)arboricity. Using an algorithm of Gabow [Gab98], we are able to show a new conditional runtime estimate for the pseudoarboricity problem.

An experimental runtime comparison of algorithms for the pseudoarboricity problem based on maximum flow algorithms and linear programs is conducted in Chapter 12.

Chapters 13 to 19 deal with NP-complete subgraph problems. Chapter 13 gives an introduction to the topic. In Chapter 14, we investigate an integer linear program for the k-cardinality tree problem based on the maximum density and compare it to existing formulations. We show that the intersection of the polytope defined by an MILP based on orientations [Coh10; Alt+14] and the polytope defined by the generalized subtour elimination constraints (*GSEC*) [Fis+94; Chi+10] is a subset of either of these polytopes. Experiments for the MILP, *GSEC* and the intersection are reported in Chapter 15.

Chapters 16 to 18 deal with the Steiner tree problem in graphs. We review existing algorithms and ILPs for the problem in Chapter 16. A randomized approximation algorithm by Byrka et al. [Byr+13] based on the hypergraphic relaxation is reviewed in Chapter 17. We modify the analysis for a better approximation factor of 1.354 in the special case of claw-free instances, and 1.25 for claw-free instances with uniform weights. Chapter 18 deals with instances that exhibit a large integrality gap for the bidirected cut relaxation. Based on a new idea for entangling two instances derived from the set cover problem, we conduct an experimental study in hope to find instances with larger integrality gaps than previously known, to no avail.

In Chapter 19, we review the maximum weight connected subgraph problem. We devise some preprocessing rules that can reduce instances considerably, which is demonstrated on real-world data, and briefly review existing algorithmic approaches.

The thesis is concluded in Chapter 20 with several open questions for future research.

1.3 COLLABORATION AND PUBLICATIONS

With the exception of the widely known theorems in the introductory chapter, all results that are not explicitly designated otherwise are contributions by the author.

Some results from Chapters 3, 4, 6, 11, and 12 were part of a conference paper at the 18th Workshop on Algorithm Engineering and Experiments (ALENEX 2016) [Blu16].

The results of Chapters 7 and 9 were made public in a preprint on arXiv[2]. The results of Chapter 10 were obtained in collaboration with Frank Fischer and have been accepted to the 46th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2020), which will be held in Limassol, Cyprus, January 20-24, 2020. An extended preprint that contains all these results is available.[3]

An earlier paper [Alt+14] of the author on a problem of connected subgraphs with k vertices (see Chapter 14) is cited because a portion of it was done as part of his Master's curriculum, and it contains considerable contributions by the other authors. Its results are extended in this thesis.

The preprocessing rules from Chapter 19 were presented at the 11th DIMACS Implementation Challenge workshop [AB14] in a collaboration with Ernst Althaus.

2 https://arxiv.org/abs/1811.06803v2
3 https://arxiv.org/abs/1811.06803

2 PRELIMINARIES

On a visit to the NASA space center,
President Kennedy spoke to a man
sweeping up in one of the buildings.
"What's your job here?", asked Kennedy.
"Well, Mr. President," the janitor replied,
"I'm helping to put a man on the moon."

— Anonymous

In this chapter, we give the theoretical foundations for the following chapters. The material in Sections 2.1-2.13 is frequently covered in mathematics and computer science curricula. The lemmata and theorems in Sections 2.14 and 2.15 are slightly generalized, but follow the original proofs quite closely and should hence not be regarded as proper contributions.

2.1 BASIC DEFINITIONS

2.1.1 *Set-Theoretic Foundations*

As the basis of our studies, we use Zermelo–Fraenkel set theory, although we nowhere near employ its full power.

We denote the set of natural numbers $\{1, 2, 3, \dots\}$ by $\mathbb{N}$ and the set of nonnegative integers $\{0, 1, 2, 3, \dots\}$ by $\mathbb{N}_0$. For $n \in \mathbb{N}$, we denote $[n] = \{1, \dots, n\}$. The set of integers is denoted by $\mathbb{Z}$, the set of rational numbers by $\mathbb{Q}$, and the set of real numbers by $\mathbb{R}$. To restrict these sets to nonnegative or positive numbers, we use sub- and superscripts as in $\mathbb{R}_0^+$ and $\mathbb{R}^+$, respectively. The empty set is denoted by $\emptyset$ and the power set of a set A by $\mathcal{P}(A)$.

We assume familiarity with propositional calculus (see Section 2.1.2) and the existential and universal quantifiers $\exists, \forall$, as well as set-theoretic relations and operations such as the element ($\in$) and subset ($\subseteq$) relations, set product ($\times$), set difference ($\setminus$), intersection ($\cap$), and union ($\cup$). Two sets A, B are *disjoint* if $A \cap B = \emptyset$. The disjoint union, i.e., the union together with the assertion that the sets are disjoint, is denoted by $\dot\cup$. If a set A is the disjoint union of a family of nonempty sets, this family forms a *partition* of A. The cardinality of a finite set A is denoted by $|A|$. For convenience, we define the set of 'unordered pairs without loops', $\binom{A}{2} := \{\{a, b\} \mid a, b \in A, a \neq b\}$.

Given two sets A, B, a *binary relation* $\sim$ is a subset of $A \times B$. We write $a \sim b \Leftrightarrow (a, b) \in {\sim}$. If $A = B$, we say that $\sim$ is *reflexive* if $a \sim a$

for all $a \in A$, *symmetric* if $a \sim b \Rightarrow b \sim a$ for all $a, b \in A$, and *transitive* if $a \sim b, b \sim c \Rightarrow a \sim c$ for all $a, b, c \in A$. A binary relation that is reflexive, symmetric and transitive is called an *equivalence relation*. The *equivalence class* of $a \in A$ is the set $\{b \in A \mid b \sim a\}$. The equivalence classes of A form a partition of A, denoted by $A/\sim$.

A *function* $f : A \to B$ is a relation $f \subseteq A \times B$ that is left-total and right-unique, i.e., every element in A is *mapped* to a unique element in B. The set $f(X) := \{f(x) \mid x \in X\}$ is the *image* of f on $X \subseteq A$.

A function that maps any two distinct elements in A to distinct elements in B is called *injective*. If for every $b \in B$, there exists $a \in A$ such that a is mapped to b, the function is called *surjective*. A function that is both injective and surjective is called *bijective*. A set S is *countable* if it is finite or there exists a bijection to $\mathbb{N}$.

For $n \in \mathbb{N}$, the *symmetric group* Σ_n is the set of bijections on $[n]$. For $\sigma \in \Sigma_n$, the set of its *inversions* is

$$\text{inv}(\sigma) := \{(i,j) \in [n]^2 \mid i < j, \sigma(i) > \sigma(j)\}$$

and its *signum* is $\text{sgn}(\sigma) := (-1)^{|\text{inv}(\sigma)|}$.

We define the *open and closed intervals*

$$(a,b) := \{x \in \mathbb{R} \mid a < x < b\}, \quad [a,b] := \{x \in \mathbb{R} \mid a \leq x \leq b\},$$

respectively, and likewise the *half-open intervals* $(a,b]$ and $[a,b)$.

The *ceiling* $\lceil \cdot \rceil : \mathbb{R} \to \mathbb{Z}$ and *floor* functions $\lfloor \cdot \rfloor : \mathbb{R} \to \mathbb{Z}$ are defined as

$$\lceil x \rceil := \min\{n \in \mathbb{Z} \mid n \geq x\}, \quad \lfloor y \rfloor := \max\{n \in \mathbb{Z} \mid n \leq x\}.$$

A function $f : A \times A \to \mathbb{R}_0^+$ is a *metric* or *distance* if for all $a, b, c \in A$,

1. $f(a,b) = f(b,a)$,
2. $f(a,b) = 0 \Leftrightarrow a = b$,
3. $f(a,c) \leq f(a,b) + f(b,c)$.

We assume the reader is familiar with norms such as the absolute value and the Euclidean norm. A norm $\|\cdot\|$ on a vector space A induces a metric via $f(a,b) = \|a - b\|$.

2.1.2 *Propositional Calculus*

Boolean variables $x_1, \ldots, x_n$ take values in $\{0,1\}$, where 1 and 0 are interpreted as 'true' and 'false', respectively. An element in $\{0,1\}^n$ is called an *assignment* to $x_1, \ldots, x_n$. *Boolean formulae* are defined inductively as follows:

- 1 and 0 are formulae.
- A Boolean variable x is a formula.

- If ϕ is a formula, so is (ϕ).
- The negation $\neg\phi := 1 - \phi$ ('not') of a formula is a formula.
- The conjunction $\phi \wedge \psi := \min(\phi, \psi)$ ('and') of formulae ϕ, ψ is a formula.

We use the convention that $\neg$ has higher precedence than $\wedge$, but expressions in parentheses are evaluated first. For convenience, we can define further connectives, in further descending order of precedence:

- Disjunction: $\phi \vee \psi := \neg(\neg\phi \wedge \neg\psi)$ ('or'),
- Implication: $\phi \Rightarrow \psi := \phi \wedge \neg\psi$,
- Equivalence: $\phi \Leftrightarrow \psi := \phi \Rightarrow \psi \wedge \psi \Rightarrow \phi$.

A *literal* is either a Boolean variable x or its negation $\neg x$. An assignment of $x_1, \dots, x_n$ for which ϕ evaluates to 1 is called *satisfying*. If a satisfying assignment exists, ϕ is called *satisfiable*. Two formulae ϕ, ψ are *equisatisfiable* if they are both satisfiable or both not satisfiable.

A formula $\phi(x_1, \dots, x_n)$ is in *conjunctive normal form* (CNF) if

$$\phi = \bigwedge_i \bigvee_j l_{ij}$$

where each l_{ij} is a literal of one of the $x_1, \dots, x_n$. The formula is in k-CNF if every *clause* $C_i = \bigvee_j l_{ij}$ has at most k literals.

2.2 GRAPHS

2.2.1 *Vertices and Edges*

Let V be a nonempty set and let $E \subseteq V \times V$. Elements of V are called *vertices* and elements of E are called (directed) *edges*. For an edge $(u, v) \in E$, u and v are called its *endpoints* or *end vertices*, and u is v's predecessor. We sometimes write $u \to v$ or $v \leftarrow u$ for (u, v) to emphasize the direction. The tuple (V, E) is called a *graph*. In this thesis, we will only deal with finite graphs, i.e., V is always a finite set. We implicitly assume that $V = \{1, 2, \dots, |V|\}$ when it is useful. A graph is said to be *complete* if $E = V \times V$. Sometimes, we allow multiple edges between two vertices, this can be formalized by a function $m : E \to \mathbb{N}_0$. Such a graph is called a *multigraph*.

For convenience, we will sometimes use the word *node* for a vertex in auxiliary structures to distinguish it from vertices in the input graph. In flow networks (Section 2.12), we will call directed edges *arcs*.

2.2.2 *Undirected and Simple Graphs*

A graph is *undirected* if for every $(u,v) \in E$, it follows that $(v,u) \in E$. In this case, we treat (u,v) and (v,u) as equivalent, and sometimes write uv. Formally, this corresponds to a set $E/\sim$, where $\sim$ is the equivalence relation defined by $(u,v) \sim (v,u)$. However, we simply write E for short, and an edge is only counted once in $|E|$.

A graph is *loop-free* if $(v,v) \notin E$ for all $v \in V$. An undirected, loop-free graph is called *simple*. We will often refer to simple graph as graphs for short, as they are our main focus. The definition of completeness carries over in the obvious way. A complete simple graph on n vertices is denoted by K_n.

2.2.3 *Adjacency and Incidence*

In simple graphs, an edge e is called *incident* to $u \in V$ if one of its endpoints is u, and in this case u is also said to be incident to e. The endpoints of e are *adjacent* to each other. The set of vertices adjacent to a vertex v is called its *neighborhood* $N(v)$. A vertex with an empty neighborhood is called *isolated*. If two distinct edges share an end vertex, they are also said to be *adjacent*.

The *incidence matrix* of a simple graph is a matrix (Section 2.9) $I \in \{0,1\}^{|V|\times|E|}$ where every row is associated with a vertex and each column with an edge. Every column corresponding to an edge uv contains exactly two ones, one in row u and one in row v.

The *adjacency matrix* of a simple graph is a matrix $A \in \{0,1\}^{|V|\times|V|}$ with $A_{uv} = 1$ if and only if $uv \in E$. For runtime and space efficiency, it is often necessary to store only the neighbors of a vertex in a list, the *adjacency list*.

A simple graph $G = (V,E)$ is said to be *bipartite* if there exists a partition $V = V_1 \dot\cup V_2$ such that every edge has one endpoint in V_1 and one in V_2.

Two graphs $G_1 = (V_1, E_1), G_2 = (V_2, E_2)$ are said to be *isomorphic* if there exists a bijection

$$f : V_1 \longrightarrow V_2$$

such that $(u,v) \in E \Leftrightarrow (f(u), f(v)) \in E_2$.

2.2.4 *Degrees and Orientations*

In directed graphs, the integer $|\{(u,v) \mid u \in V\}|$ for $v \in V$ is called the *indegree* of v, denoted by $\text{indeg}_G(v)$. Likewise, for a vertex $u \in V$ the integer $|\{(u,v) \mid v \in V\}|$ is called the *outdegree* of u, denoted by

$\text{outdeg}_G(u)$. We drop the index where the graph is clear from context. Clearly, we have

$$\sum_{v\in V} \text{indeg}(v) + \text{outdeg}(v) = |E|.$$

In a simple graph, the size of the neighborhood of u is called the *degree* of u, denoted by $\deg_G(u)$. The *degree sum formula*, due to Euler [Eul36], states that for a simple graph $G = (V, E)$,

$$\sum_{v\in V} \deg(v) = 2|E|. \tag{2.1}$$

This is because by summing the degrees, every edge is counted once from each of its endpoints. We will frequently use this formula throughout this thesis. The *maximum degree* among the vertices in a simple graph G is denoted by $\Delta(G)$. Given a simple graph $G = (V, E)$, an *orientation* of G is a directed graph $\vec{G} = (V, \vec{E})$ with $|\vec{E}| = |E|$ in the aforementioned sense and $E \subseteq \vec{E}$, i.e., for every edge in G one of two possible directions is chosen. If for some $d \in \mathbb{N}_0$, $\text{indeg}_{\vec{G}}(v) \leq d$ holds for every $v \in V$, then $\vec{G}$ is said to be a *d-orientation*. Fractional orientations will be introduced in Section 3.5.

2.2.5 *Subgraphs, Induced Subgraphs, and Matchings*

A graph $H = (V_H, E_H)$ is a *subgraph* of a graph $G = (V, E)$ if $V_H \subseteq V$ and $(u, v) \in E_H \Rightarrow (u, v) \in E$. We write $H \subseteq G$ for short and $H \subsetneq G$ if $E_H \neq E$. For subgraphs $H_1, H_2 \subseteq G$, we use the shorthand $H_1 \cup H_2$ for the graph that consists of the union of the edges of H_1 and H_2 on the union of the vertices of H_1 and H_2. We analogously define $H_1 \cap H_2$, with the exception that the intersection of the vertex sets must not be disjoint.

A subgraph H is *induced* if from $(u, v) \in E$ and $u, v \in V_H$ it follows that $(u, v) \in E_H$, i.e., every edge present in the graph is always present in the subgraph if its endpoints are in the subgraph. The subgraph induced by a nonempty set $S \subseteq V$ is denoted by $G[S]$. The set of edges of $G[S]$ is denoted by $E[S]$. Likewise, for a set $T \subseteq E$, we define $V[T]$ to be the set of all vertices that are endpoint of at least one edge in T.

A *matching* is a subset of edges $M \subseteq E$ such that every vertex in (V, M) has at most one neighbor.

2.2.6 *Paths, Cycles, and Connected Components*

In a (directed) graph, a *path* of *length* $n - 1$ is a sequence of vertices $(v_1, \ldots, v_n)$ such that $(v_i, v_{i+1}) \in E$ for $i = 1, \ldots, n - 1$. We will sometimes write $v_1 \to v_2 \to v_3$ etc. for a clearer presentation of paths, and $v_1 \rightsquigarrow v_n$ to denote some unspecified path from v_1 to v_n. A path forms a *cycle* if $v_n = v_1$ and $n \geq 2$.

In a *simple path*, no vertex may be visited twice. A *simple cycle* is a cycle that is a simple path (in the sense that $v_1 = v_n$ is only visited once). In simple graphs, we are usually only interested in simple cycles, thus we will call them cycles for short. The only exception in this thesis are *Euler tours*, in which every edge is visited exactly once, but a vertex may be visited several times. A simple graph is said to be *Eulerian* if it has an Euler tour.

Theorem 2.2.1 ([Eul36; HW73])**.** *A simple graph G is Eulerian if and only if G is connected and the degree of every vertex is even.*

Hierholzer's algorithm [HW73] finds an Euler tour, if one exists, in linear time.

If there is a cycle present in a graph, it is called *cyclic*, and *acyclic* otherwise. Note that in a simple graph, a simple cycle must consist of at least three vertices, i.e., two vertices joined by an edge are not considered a simple cycle. In directed graphs, however, 1-cycles (loops) and 2-cycles can exist.

A simple graph is said to be *connected* if there is a path from every $u \in V$ to every $v \in V$. Define $u \sim_c v$ if and only if there is a path from u to v. This is an equivalence relation on V, and we call the subgraphs induced by the equivalence classes in $V/{\sim_c}$ the *connected components* of the graph. If a (sub-)graph (V, E) is connected, it is said to be *spanning* V.

A graph is k-(vertex)-connected if it has at least k vertices and removing any k vertices does not disconnect the graph. 2- and 3-connected graphs are called biconnected and triconnected, respectively. It is possible to determine the biconnected and triconnected components of a graph in linear time [HT73]. A vertex that is in several biconnected components is called a *cut vertex*; removing such a vertex disconnects the graph.

2.2.7 *Trees and Planarity*

A *forest* is an acyclic simple graph. A *tree* is a connected forest. A vertex of degree at most one in a tree is called a *leaf*. If every tree in a forest is a path, the forest is said to be *linear*. A tree of $n \in \mathbb{N}$ vertices is a *star* if it has one vertex of degree $n-1$ and the other vertices (if any) are leaves. A star of four vertices is called a *claw*.

The following elementary lemma is quite useful, and we shall use it several times in this thesis.

Lemma 2.2.2. *A simple graph* (V, E) *is a tree if and only if it is acyclic and* $|E| = |V| - 1$.

A simple graph $G = (V, E)$ is connected if and only if it has a *spanning tree*, i.e., there is a subgraph with vertex set V that is a tree.

A *rooted tree* is a tree with a distinguished vertex $r \in V$, the *root*. The *height* of a rooted tree is the maximum length of a path from r to any vertex $v \in V$. A binary tree is a tree where the degree of every vertex is bounded by three and the root's degree is bounded by two. A *complete binary tree* has 2^i vertices at height i for every $i = 0, \dots, h$, where h is the height of the tree.

Typically, one imagines the edges of the root as directed away from it, and recursively the neighbors of the root as roots of their respective subtrees, such that there is exactly one directed path from r to every vertex $v \in V$. A directed graph with this property is called an *arborescence*.

A *topological ordering* of a directed graph $G = (V, E)$ is an ordering $(v_1, \dots, v_{|V|})$ of the set V such that $v_i \to v_j$ implies $i < j$.

Lemma 2.2.3. *A topological ordering of a directed graph G exists if and only if G is acyclic.*

A graph is called *planar* if it can be represented in the plane by $|V|$ distinct points and simple curves for every edge such that curves only intersect in the vertex points. We omit a formal definition here. The plane is partitioned by such an *embedding* into *faces*. The single unbounded face is also called the *outer face*.

2.3 ASYMPTOTICS

When talking about the runtime of an algorithm, it is often not that important to determine the exact number of steps it takes. Rather, we are usually interested in an upper bound on the number of steps that is tight up to a constant factor and holds for all sufficiently large inputs. For a simplified presentation, we exclude zero from the definition here.

Let $f : \mathbb{R}^+ \to \mathbb{R}^+$. We define

$$\begin{aligned} \mathcal{O}(f) &:= \{g : \mathbb{R}^+ \to \mathbb{R}^+ \mid \exists c > 0 \, \forall x \in \mathbb{R}^+ : g(x) \leq cf(x)\}, \\ \Omega(f) &:= \{g : \mathbb{R}^+ \to \mathbb{R}^+ \mid f \in \mathcal{O}(g)\}, \\ \Theta(f) &:= \mathcal{O}(f) \cap \Omega(f). \end{aligned}$$

Intuitively speaking, $\mathcal{O}(f)$ (read: 'big-oh of f' or sometimes 'oh of f') is the set of functions that dominate f up to a constant factor (asymptotic upper bounds), $\Omega(f)$ is the set of functions that are dominated by f up to a constant factor (asymptotic lower bounds), and $\Theta(f)$ is the set of functions that are asymptotically equivalent to f. Sometimes, one additionally uses

$$\begin{aligned} o(f) &:= \{g : \mathbb{R}^+ \to \mathbb{R}^+ \mid \forall c > 0 \, \forall x \in \mathbb{R}^+ : g(x) < cf(x)\}, \\ \omega(f) &:= \{g : \mathbb{R}^+ \to \mathbb{R}^+ \mid f \in o(g)\}. \end{aligned}$$

The above definitions can be generalized to functions $f : (\mathbb{R}^+)^n \to \mathbb{R}^+$, which is useful when analyzing graphs in terms of $|V|$ and $|E|$, or when the runtime in an approximation scheme depends on an error parameter $\epsilon > 0$. We follow the widespread notation that $g \in \tilde{\mathcal{O}}(f)$ (read: 'soft-oh of f') if $g \in \mathcal{O}(f \log^c f)$ for some $c > 0$. In other words, $\tilde{\mathcal{O}}$ hides polylogarithmic factors. For all asymptotic estimates, the baseless logarithm log is used because a change of base only constitutes a multiplication by a constant factor.

A function that is sometimes used in algorithm analysis is the *iterated logarithm* (to the base two), which for $x > 0$ is defined as

$$\log^*(x) := \begin{cases} 0, & x \leq 1, \\ 1 + \log^*(\log_2(x)), & x > 1. \end{cases}$$

It counts how many times the binary logarithm has to be applied to reach a small constant (here, one). It grows very slowly, for example, consider

$$x = 2^{2^{2^{2^2}}} \approx 2 \cdot 10^{19{,}728}.$$

The iterated logarithm of x is the height of the exponential tower, namely $\log^*(x) = 5$. This function can be regarded as constant for all practical purposes.

For ease of notation, we define *functional iteration* as follows:

$$f^{(i)}(n) := \begin{cases} n, & \text{if } i = 0, \\ f(f^{(i-1)}(n)), & \text{if } i \geq 1. \end{cases}$$

A function that grows even more slowly than the iterated logarithm is the *inverse Ackermann function*. Ackermann [Ack28] and Sudan [Sud27] showed the existence of computable functions (Section 2.5) that grow faster than all primitive-recursive functions and are, in particular, not primitive-recursive. A popular simplified variant was described by Péter [Pét35]. The textbook of Cormen et al. [Cor+01] uses the following definition. For $k \geq 0$ and $j \geq 1$, define

$$A_k(j) := \begin{cases} j + 1, & \text{if } k = 0, \\ A_{k-1}^{j+1}(j), & \text{if } k \geq 1. \end{cases}$$

The inverse Ackermann function is defined as follows:

$$\alpha(n) := \min_{k \in \mathbb{N}_0} \{A_k(1) \geq n\}.$$

We have $\alpha(n) \leq 4$ for all $n \leq 2^{2048} \approx 3.23 \cdot 10^{616}$, and this is a rather conservative estimate [Cor+01]. La Poutré [Pou90] uses a different, but asymptotically equivalent definition of the Ackermann function.

We omit the details and only state the first three 'row inverses' to illustrate their slow growth:

$$\begin{aligned}
&\alpha(1,n) := \lceil \log n \rceil\,,\\
&\alpha(2,n) := \min_{k\in\mathbb{N}_0}\left\{\left\lceil \log^{(k)} n\right\rceil = 1\right\} = \log^*(n),\\
&\alpha(3,n) := \min_{k\in\mathbb{N}_0}\left\{\left\lceil \log^{*(k)} n\right\rceil = 1\right\},\\
&\qquad\ldots
\end{aligned}$$

In this thesis, we will sometimes assume for runtime estimates that no vertex is isolated, and thus the number of edges $|E|$ is at least half the number of vertices. We do this because for the problems we consider, isolated vertices can be ignored, removed or dealt with separately. Thus, we use $\mathcal{O}(|E|)$ instead of $\mathcal{O}(|V| + |E|)$ in many places.

Often one considers a *family* (an indexed set) of instances, typically graphs, with certain properties. Sometimes better asymptotic estimates can be given for a certain family.

2.4 PROBABILITY THEORY

We give an introduction to (discrete) probability theory. Since the set of elementary events is always countable in this thesis, we can use its power set as the set of all events. (This is an example of a σ-algebra.) We largely follow the textbook by Mitzenmacher and Upfal [MU05], which also uses this approach for a shorter and more accessible presentation.

Definition 2.4.1. Let S be a countable set, the set of *elementary events*. A subset $A \subseteq S$ is called an *event*. A *discrete probability distribution* on S is a function

$$\Pr : 2^S \to [0,1]$$

that satisfies for a sequence $(A_i)_{i\in\mathbb{N}}$ of disjoint events $A_i \subseteq S$

$$\begin{aligned}
\Pr(S) &= 1,\\
\Pr\left(\bigcup_{i\in\mathbb{N}} A_i\right) &= \sum_{i\in\mathbb{N}} \Pr(A_i).
\end{aligned}$$

We write $\Pr[\cdot]$ instead of $\Pr(\cdot)$. Two events $A, B \subseteq S$ are said to be *independent* if $\Pr[A \cap B] = \Pr[A] \cdot \Pr[B]$.

The following statements are readily proved: For any two events A, B, we have $\Pr[A \cup B] = \Pr[A] + \Pr[B] - \Pr[A \cap B]$. The *union bound* is the simple fact

$$\Pr\left(\bigcup_{i\in\mathbb{N}} A_i\right) \le \sum_{i\in\mathbb{N}} \Pr(A_i)$$

for events $(A_i)_{i\in\mathbb{N}}$. We say that an event A depending on $n \in \mathbb{N}$ happens *with high probability* if $\Pr[A] \geq 1 - n^{-c}$ for some $c > 0$.

Definition 2.4.2 (Random Variable)**.** Let $X : S \to \mathbb{R}$ be a function, a *real-valued random variable*, which we will call *random variable* for short. For $x \in \mathbb{R}$, we write '$X = x$' as a shorthand for the event $\{s \in S \mid X(s) = x\}$. Two random variables X, Y are *independent* if for all $V, W \subseteq \mathbb{R}$, the events

$$A = \{s \in S \mid A(s) \in V\}, \qquad B = \{s \in S \mid B(s) \in W\}$$

are independent.

Definition 2.4.3 (Expected Value)**.** The *expected value* (or *expectation*) of a random variable X is

$$\mathbf{E}[X] := \sum_{x\in\mathbb{R}} x \Pr[X = x],$$

if it exists.

The *average* is a special case of the expected value for the *uniform distribution*, where all elementary events have equal probability.

Theorem 2.4.4 (Linearity of Expectation)**.** *Let X, Y be a random variables, and $a \in \mathbb{R}$. Then*

$$\mathbf{E}[aX + Y] = a\mathbf{E}[X] + \mathbf{E}[Y].$$

Note that the above theorem does not require independence of the random variables. As we shall consider an algorithm that may potentially run endlessly in Chapter 17, we need the following extension to infinite sums of expected values.

Theorem 2.4.5. *Let $(X_i)_{i\in\mathbb{N}}$ be a sequence of real-valued random variables with $\sum_{i\geq 1} \mathbf{E}[|X_i|] < \infty$. Then*

$$\mathbf{E}\left[\sum_{i\geq 1} X_i\right] = \sum_{i\geq 1} \mathbf{E}[X_i]. \tag{2.2}$$

We next turn to conditional probabilities.

Definition 2.4.6. Let A, B be events such that $\Pr[B] > 0$. The *conditional probability* that event A occurs given that B occurs is

$$\Pr[A \mid B] := \frac{\Pr[A \cap B]}{\Pr[B]}.$$

If A, B are independent, we have $\Pr[A \mid B] = \Pr[A]$, which is intuitive.

Definition 2.4.7. Let X be a random variable and let A be an event with $\Pr[A] > 0$. Then the *conditional expectation* of X given that A occurs is

$$\mathbf{E}[X \mid A] := \sum_{x \in \mathbb{R}} x \Pr[X = x \mid A].$$

The following lemma is obvious.

Lemma 2.4.8. *Let X, Y be random variables. Then*

$$\mathbf{E}[X] = \sum_{\substack{y \in \mathbb{R} \\ \Pr[Y=y]>0}} \Pr[Y = y] \mathbf{E}[X \mid Y = y].$$

2.4.1 *The Coupon Collector Problem*

Imagine a collector buys items, and each item comes with one of n distinct coupons, uniformly distributed. How many items must the collector buy in expectation until she has collected all coupons? Let e_i denote the expected number of buys the collector has to make in order to increase the number of distinct collected coupons from i to $i + 1$. When the next buy is made, the coupon has not been collected yet with probability $(n - i)/n$. It has been collected already with probability i/n, in this case we are in the same situation as before with i distinct coupons. Hence

$$e_i = 1 + \frac{i}{n} e_i. \tag{2.3}$$

It can be shown[1] that $e_i < \infty$, therefore we may rearrange (2.3) to

$$e_i = \frac{n}{n - i}. \tag{2.4}$$

The expected number of buys is now

$$\sum_{i=0}^{n-1} e_i = \sum_{i=0}^{n-1} \frac{n}{n - i} = n \sum_{i=1}^{n} \frac{1}{i} = nH_n,$$

where H_n is called the n-th *Harmonic number*, which is bounded as $\mathcal{O}(\log n)$. The standard proof is via the lower sum of

$$\int_1^x \frac{1}{s} ds = \ln x.$$

A proof that avoids calculus is possible via proving $\sum_{i=1}^{2^k-1} \frac{1}{i} \leq k$ by induction on k. We can conclude that $\mathcal{O}(n \log n)$ buys have to be made in expectation in order to collect all coupons.

1 In fact, e_i is geometrically distributed. However, only assuming that e_i is finite, its value follows from the proof at hand.

2.5 ALGORITHMS AND MACHINE MODELS

An algorithm is a finite set of instructions that is run on an input and, in case it terminates, has an output. Formally, it can be defined as a Turing machine [Tur36], which can be shown to be equivalent to a program on a random-access machine [CR73]. The latter is similar to present-day computers. We will give a brief overview.

A *random-access machine* (RAM) has registers in which it can store integers. Rational numbers can be represented with two integers. Some of these registers may hold the input value(s), and some may be used for the output. The *input size* is the total length of the input values in binary representation. One designated 'working register' is called *accumulator*, it holds the result of the current computation and is initially zero. An operation such as 'ADD 5' would cause the accumulator to increase its content by 5. Basic operations are addition, subtraction, multiplication and division. They can also address a register rather than a constant, for example 'ADD *5' to add the content of register 5 to the accumulator.

A program for a random access machine is a finite sequence of instructions. After an instruction has been carried out, the next instruction in the sequence is loaded. When the end of the sequence has been reached, the program halts and its output is in the registers. In addition to the arithmetic operations described above, there is an operation 'JZERO i' that jumps to instruction number i if the accumulator's content is zero, otherwise the next instruction is loaded. It is easy to see that loops can be implemented using the JZERO operation and registers for counting.

An algorithm is said to be *partially correct* if it outputs the correct solution (with regard to the task we expect it to do) if it terminates. If in addition the algorithm always terminates, it is said to be *(totally) correct*.

An obvious obstacle are irrational numbers, which cannot be represented exactly in a computer in an explicit way.[2] We will assume that either only natural numbers are allowed in registers (negative numbers and fractions can be represented with additional registers), or that the machine is powerful enough to somehow handle irrational numbers without an actual binary representation. Even in the latter scenario, irrational numbers may be problematic. For example, the Ford–Fulkerson algorithm need not terminate for irrational capacities, but there are algorithms that always terminate (see Sections 2.12.4 and 2.13). We will also state theorems such as 'Given a flow, it can be converted into ...'. Even if the total value of the flow is an integer, the flow going through an arc could be, say, π and on another it could be $4 - \pi$, adding to 4. Reasonable maximum flow algorithms on

2 A subset of the irrational numbers are the *uncomputable numbers*. For such a number, no algorithm exists that can compute the i-th digit in finite time for every index i.

networks with rational capacities do not introduce irrational numbers, however.

A tricky question a theoretical computer scientist has to deal with is how to count the number of computational steps or the cost of the operations carried out by an algorithm, and which operations algorithms are allowed to be performed in the first place. The number of steps are often referred to as *time*, as they directly translate to time in the physical sense for a given number of operations per second. The maximum amount of space the algorithm requires at any time during the computation is also sometimes of interest.

A precise way of accounting is the *logarithmic cost* model (or Turing machine model), in which the number of bits involved in an operation is counted. This model is reasonable when the numbers during the computation become large.

In the *uniform cost* model (or arithmetic model), every operation costs a single unit. This is somewhat reasonable because current processor architectures perform an addition of two 64-bit numbers 'in one go' (taking a few clock cycles), and not 'bit-by-bit'. This limiting size is called the *word size*. If the operands of an operation exceed 64 bits, the processor has to perform multiple 64-bit operations. If, for example, the edge capacities in some graph-theoretic problem are bounded by 2^{64}, which they are for most applications, the uniform cost model is a good choice. It suffices for the reader to assume the uniform cost model for the results in this thesis. However, for complexity-theoretic considerations, machines with unlimited precision are unreasonably powerful [BMS81].

Fredman and Willard introduced the *transdichotomous model* [FW93]. It assumes that multiplication, addition and their inverse operations can be performed in constant time on numbers of size $\mathcal{O}(\log n)$, where n is the size of the total input. This is a reasonable compromise between the two models (hence the name).[3] The shift from 32-bit architectures to 64-bit architectures in the 2000s could be seen as such an adaptation of the machines to growing demands. We shall use this model as a formal basis in this thesis unless stated otherwise.

2.6 COMPLEXITY CLASSES

Problems in computer science can be stated in the form of yes-or-no questions (decision problems), as tasks of determining an optimal value (evaluation problems), and of finding a solution that attains the optimal value (search problems).

Formally, decision problems are represented as sets of words over a finite set, the *alphabet*. Given an alphabet Σ, the set Σ^* denotes all finite tuples, called *words* or *strings*, that can be formed with elements

3 Note that when sorting integers, the transdichotomous model is powerful enough to break the $\Omega(n \log n)$ lower bound for *comparison-based* sorting [FW93].

in Σ. This includes the unique *empty word* of length zero. A set $L \subseteq \Sigma^*$ is referred to as a *language*, or occasionally (somewhat imprecise) as a problem. It is sufficient to consider the binary alphabet $\{0,1\}$, other alphabets can be simulated by appropriate encoding. The *characteristic function* of L is the indicator function

$$\chi_L : \Sigma^* \longrightarrow \{0,1\}$$

with $\chi_L(x) = 1 \Leftrightarrow x \in L$.

If there is an algorithm that computes a function f, i.e., outputs the function value $f(x)$ for every argument x in finite time, then f is called *computable*. If the characteristic function of a language L is computable, L is said to be *decidable*. There are undecidable problems, the most notable being the halting problem [Tur36; Soa16].

The decidability of problems is investigated in computability theory. On the other hand, complexity theory focuses on decidable problems, and further distinguishes them according to their time or space complexity, to name the most commonly measured resources. Of particular interest are problems that can be decided in *polynomial time* (usually measured on a Turing machine), where the runtime of an algorithm is bounded by a polynomial in the input size.[4] The class of these problems is denoted by P. Cobham's thesis [Cob65] identifies problems that can be efficiently solved in practice with problems in the class P. While the thesis has proved its worth in many cases, a runtime of, say, $\mathcal{O}(n^{1000})$ is clearly prohibitively large even for moderate values of n. In fact, the time hierarchy theorem [HS65; AB09] guarantees that problems exist that are solvable in $\mathcal{O}(n^{1000})$ time but not in, say, $\mathcal{O}(n^{999})$ time. While the runtime in complexity theory usually refers to Turing machines, a random-access machines can simulate a Turing machine running in time $\mathcal{O}(T(n))$ in time $\mathcal{O}(T(n)^3)$ [CR73]. The following definitions hence apply for random-access machines as well.

The class NP is the set of problems X that can be verified by an algorithm V that runs in polynomial time, the *verifier*: An instance $I \in \Sigma^*$ is in X if and only if there exists a *certificate* C such that $V(I,C)$ outputs 1 in polynomial time.

Intuitively speaking, problems can be transformed into other problems such that solving the transformed problem solves the original. A (many-one) reduction from $A \subseteq \Sigma^*$ to be $B \subseteq \Gamma^*$ is a computable function $f : \Sigma^* \to \Gamma^*$ with $a \in A \Leftrightarrow f(a) \in B$. If the reduction can be computed in polynomial time, it is said to be a *polynomial-time reduction* or *Karp reduction*. In this case, we write $A \leq_p B$.

There is another, more general type of reduction that amounts to calling an algorithm as a subroutine ('black box'): A language A is

4 One further distinguishes strongly and weakly polynomial time. An algorithm that uses polynomial space and whose number of steps in the unit-cost model is bounded by a polynomial in the number of integers in the input is said to be *strongly polynomial*. It can then be turned into a polynomial-time algorithm in the logarithmic cost model. Otherwise, it is said to be weakly polynomial.

said to be *Turing-reducible* to B if A can be decided given an *oracle* for B: The algorithm is given the power to query whether a string is in B; the answer to this question is correctly returned.[5] If in addition, the algorithm runs in polynomial time, in particular with a polynomial number of polynomially-sized queries to the oracle, A is said to be *Cook-reducible* to B. While Cook's original paper [Coo71] uses Cook reductions to define NP-hardness, most textbooks use the simpler Karp reductions.

A language L is said to be *NP-hard* if $K \leq_p L$ for every K in NP. If a language is both in NP and NP-hard, it is called NP-complete. It is widely conjectured that NP does not equal P, and many results depend on this assumption [Aar17]. Note that this conjecture does not imply that NP-complete problems require exponential time to solve, i.e., every algorithm that decides the problem would need $\Omega(2^{n^c})$ time for some $c > 0$. It could be the case that, say, $\mathcal{O}(n^{\log n})$ time suffices.

Theorem 2.6.1. *Let $A \subseteq \Sigma^*, B \subseteq \Gamma^*$ be languages, and let $A \leq_p B$.*

1. *If A is NP-hard, then B is also NP-hard.*
2. *If B is in NP, then A is in NP.*
3. *If B is in P, then A is in P.*

The proof of this theorem is straightforward, and 1. and 2. also hold for Cook reductions when altering the hardness definition to Cook reductions. But are there problems that are NP-hard, or even NP-complete?

The halting problem is easily shown to be NP-hard, but it is not NP-complete. An example of an NP-complete problem is deciding whether a proof of some length n exists for a given mathematical statement [AB09].[6] While the generality of this problem makes it plausible that the problem is hard to solve, and indeed the NP-completeness proof is easy, the Cook–Levin Theorem establishes NP-completeness of a *combinatorial* problem that appears to be much simpler at first glance.

Definition 2.6.2 (Satisfiability Problem (SAT))**.** Given a formula ϕ in conjunctive normal form, is there a satisfying assignment for ϕ?

Theorem 2.6.3 (Cook–Levin Theorem, [Coo71; Lev73])**.** *SAT is NP-complete.*

Since checking a given assignment for a SAT instance is possible in polynomial time, SAT is in NP. A rough sketch of the NP-hardness proof is as follows. By definition, every problem X in NP has a

5 Note that an undecidable language L becomes decidable for machines with access to an L-oracle. However, there is a 'halting problem' that is undecidable for L-oracle machines as well [Soa16, Section 3.4.2].

6 This problem was first mentioned in a letter of Kurt Gödel to John von Neumann in 1956 [Lip10].

polynomial-time verification algorithm in form of a Turing machine T_X. Given T_X, the execution of T_X on an X-instance I can be encoded in a SAT formula that is satisfiable if and only if $T_X(I)$ accepts. Therefore $X \leq_p$ SAT.

If the instances of SAT are restricted to be in 3-CNF, the problem is called 3SAT. A SAT instance can be transformed into an equisatisfiable 3SAT instance in polynomial time by splitting clauses and adding linking variables. This establishes NP-completeness of 3SAT and is useful to show NP-hardness of several problems such as the clique problem.

Definition 2.6.4 (Clique Problem)**.** Given a simple graph G and $k \in \mathbb{N}$, is there a complete subgraph of k vertices (a k-clique)?

NP-hardness of the clique problem can be proved by transforming a formula ϕ in 3-CNF with k clauses into a graph that contains a k-clique if and only if ϕ is satisfiable [Kar72].

2.7 APPROXIMATION ALGORITHMS

Sometimes, exact computation is too time-demanding, hence one may turn to approximation algorithms. For $c \geq 1$, a c-approximation algorithm for a minimization problem with optimum value OPT is a polynomial-time algorithm whose output has value at least OPT and at most $c \cdot OPT$. Sometimes, the value itself is the output, as opposed to the solution attaining the value. In a maximization problem, the value must be between $c \cdot OPT$ and OPT for $c \leq 1$. The algorithm may be randomized and the approximation factor may be achieved in expectation only. The value c is called the *approximation factor* or the *approximation ratio*.

In a minimization problem, a polynomial-time approximation scheme (PTAS) is an algorithm that computes a $(1+\epsilon)$-approximation within a runtime that is polynomial in the input size for every fixed $\epsilon > 0$. If the runtime is polynomial in both the input size and ϵ, it is called a *fully polynomial-time approximation scheme* (FPTAS). The definition is analogous for maximization problems with $(1-\epsilon)$.

An excellent resource on the topic is Vazirani's textbook [Vaz01].

2.8 BREADTH-FIRST AND DEPTH-FIRST SEARCH

Two of the most basic graph algorithms are breadth-first search (BFS) and depth-first search (DFS). DFS was first described by Trémaux in 1876 (see [Luc82]). BFS was first described by Zuse in his doctoral thesis in 1945, which was rejected for formal reasons and published only much later [Zus72]. Excellent resources on the topic are the textbooks by Cormen et al. [Cor+01] and Even [Eve11].

Algorithm 2.1: Breadth-first search on a graph $G = (V, E)$ with start vertex $s \in V$. In addition, distance labels are computed that can be used to build the level graph of G.

```
visited[$s$] = true
predecessor[$s$] = null
distance[$s$] = 0
Queue $Q$ = $\{s\}$
while $Q \neq \emptyset$ do
    $u$ = $Q$.removeFirst();
    foreach $(u, v) \in E$ do
        if visited[$v$]=false then
            visited[$v$] = true
            distance[$v$] = distance[$u$]+1
            predecessor[$v$] = $u$
            $Q$.addLast($v$)
```

Let $G = (V, E)$ be a (directed or undirected) graph and $s \in V$ be a start vertex. A vertex $v \in V$ is *reachable* from s if there is a path $s \rightsquigarrow v$. Reachability is easily seen to be reflexive and transitive. In undirected graphs, it is also symmetric.

Using transitivity, BFS and DFS compute the set of all vertices reachable from s. The main difference is the order in which vertices are visited. Using either of the two, it is straightforward to compute the connected components of a simple graph. Moreover, a path from s to each vertex v in the same connected component can be reconstructed: Both algorithms store the predecessor for each vertex v on the BFS (or DFS) path from s to v. It is also easy to modify the algorithms in order to determine whether the component contains a cycle, which we shall use extensively in Chapters 9 and 10.

Breadth-first search can be implemented to find the shortest distances (measured by the number of edges) from s to every vertex v in the same connected component. We will use this to construct a *level graph* in Dinitz's algorithm (Section 2.13). BFS can also be used to recognize whether a graph is bipartite.

BFS is implemented with a *queue*: When a vertex is first discovered in the search, it is added at the end of the queue. Elements are retrieved from the start of the queue (the 'first-in, first-out' principle). A pseudocode description of BFS can be found in Algorithm 2.1 on this page. In DFS, a *stack* is used: When a vertex is first discovered, it is put on top of the stack. Elements are retrieved from the top of the stack (the 'last-in, first-out' principle). A pseudocode description of DFS can be found in Algorithm 2.2 on the next page.

DFS and BFS can both be used to find augmenting paths in flow networks (see Subsection 2.12.3), which is done by checking whether a 'sink' vertex t can be reached from a 'source' vertex s in the residual network.

Algorithm 2.2: Depth-first search on a graph $G = (V, E)$ with start vertex $s \in V$.

```
visited[$s$] = true
predecessor[$s$] = null
Stack $S = \{s\}$
while $S \neq \emptyset$ do
    $u = S$.removeTopElement();
    foreach $(u, v) \in E$ do
        if visited[$v$] = false then
            visited[$v$] = true
            predecessor[$v$] = $u$
            $S$.addOnTop($v$)
```

2.9 LINEAR ALGEBRA

We will only consider the vector space $\mathbb{R}^n$ (over the field $\mathbb{R}$) in our studies. An element

$$x = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} \in \mathbb{R}^n$$

is called a (column) vector. While the arrangement of the entries x_i in vertical or horizontal order does not change the properties of x, it is useful to think of vectors as columns by default. A row vector is written as the *transpose* $x^T = (x_1, \dots, x_n)$. For vectors, we write $x \leq y$ if $x_i \leq y_i$ for all $i \in \{1, \dots, m\}$, and likewise for $\geq$.

A *real matrix* $A \in \mathbb{R}^{m \times n}$ is an array

$$\begin{pmatrix} a_{11} & \cdots & a_{1n} \\ a_{21} & \cdots & a_{2n} \\ a_{m1} & \cdots & a_{mn} \end{pmatrix}.$$

We may also write A_{ij} for a_{ij}. The matrix can be viewed as an ordering of n column vectors

$$A^{(j)} = \begin{pmatrix} a_{1j} \\ \vdots \\ a_{mj} \end{pmatrix}, \qquad j = 1, \dots, n,$$

or m row vectors $a_i = (a_{i1}, \dots, a_{in}), i = 1, \dots, n$.

The transpose $A^T \in \mathbb{R}^{m\times n}$ of a matrix is defined as

$$A^T_{ij} := A_{ji}, \qquad (i,j) \in \{1,\dots,m\} \times \{1,\dots,n\}.$$

The product of two matrices $A \in \mathbb{R}^{m\times n}, B \in \mathbb{R}^{n\times k}$ is defined as

$$(A \cdot B)_{ij} := \sum_{l=1}^{n} A_{il} B_{lj}.$$

The matrix product can be computed with $\mathcal{O}(mnk)$ multiplications and additions of real numbers. The matrix-vector product Ax is this product for $B = x$. The *inner product* (also called *dot product* or *scalar product*) of two vectors $x, y \in \mathbb{R}^n$ is $y^T x = x^T y$.

A collection of vectors $x_1, \dots, x_k$ is said to be *linearly independent*, if from

$$\sum_{i=1}^{k} \lambda_i x_i = 0$$

it follows that $\lambda_i = 0$ for all $i \in \{1,\dots,k\}$. Otherwise, the vectors are called *linearly dependent*.

For $n \in \mathbb{N}$, we denote the $(n \times n)$ *identity matrix* by

$$I_n := \begin{pmatrix} 1 & & \\ & \ddots & \\ & & 1 \end{pmatrix},$$

all non-diagonal entries being zero. A square matrix $A \in \mathbb{R}^{n\times n}$ is *invertible* if there exists $A^{-1} \in \mathbb{R}^{n\times n}$ such that $AA^{-1} = I_n = A^{-1}A$. A square matrix is invertible if and only if its rows (or columns) are linearly independent. A non-invertible square matrix is called a *singular* matrix.

The *determinant* of a square $(n \times n)$-matrix A is defined as follows.

$$\det(A) := \sum_{\sigma\in\Sigma_n} \operatorname{sgn}(\sigma) \prod_{i=1}^{n} a_{i,\sigma(i)}.$$

A matrix has determinant zero if and only if it is singular.

2.10 MATROID THEORY

Given a (finite) *ground set* S, a *matroid* is a set system $M \subseteq \mathcal{P}(S)$ with the following properties:

1. $\emptyset \in \mathcal{M}$,
2. $X \in \mathcal{M}, Y \subseteq X \Rightarrow Y \in \mathcal{M}$,
3. $X, Y \in \mathcal{M}, |X| > |Y| \Rightarrow \exists x \in X \setminus Y : Y \cup \{x\} \in \mathcal{M}$.

A set system satisfying only 1. and 2. is called an *independence system*.

The restriction that S be finite is not necessary, but simplifies the presentation in our computational setting. A matroid is called *loopless* if for every element $x \in S$ we have $\{x\} \in \mathcal{M}$ (an x with $\{x\} \notin S$ is called a *loop*). In this thesis, all examples of matroids are loopless, and the definition is only needed for technical reasons.

The sets $X \in M$ are called *independent* sets, the sets in $\mathcal{P}(S) \setminus \mathcal{M}$ are called *dependent*. This is a generalization of independence in linear algebra, as the following proposition shows.

Proposition 2.10.1. *Let A be a matrix and S be the set of its columns. Let*

$$\mathcal{I} = \{X \subseteq S \mid \textit{the elements of } X \textit{ are linearly independent}\}.$$

Then $\mathcal{I}$ is a matroid.

For a matroid $\mathcal{M}$ over S and $X \subseteq S$, the *rank* $\rho_{\mathcal{M}}(X)$ is the maximum cardinality of an independent subset of X. We may drop the subscript when the matroid is clear from context.

The *rank* of a matroid $\mathcal{M}$ is $\rho(\mathcal{M}) := \max_{X \in \mathcal{M}} \rho_{\mathcal{M}}(X)$. In the situation of Proposition 2.10.1, the rank of a subset X of columns is exactly the rank of the matrix defined by these columns.

2.11 LINEAR PROGRAMMING

Many optimization problems can be modeled by linear inequalities and a linear objective. For an introduction to linear optimization, we recommend the textbooks by Bertsimas and Tsitsiklis [BT97] and Schrijver [Sch99].

2.11.1 *Linear Programs and the Canonical Form*

In a *linear program* (LP), one optimizes a linear cost function $c^T x$ defined by a cost vector $c \in \mathbb{R}^n$ over a set of variables $x \in \mathbb{R}^n$, which is constrained by m *linear constraints*. For $i \in \{1, \dots, m\}$, the i-th constraint has one of the three following possible forms

$$a_i^T x \geq b_i, \tag{2.5}$$
$$a_i^T x \leq b_i, \tag{2.6}$$
$$a_i^T x = b_i, \tag{2.7}$$

where $a_i \in \mathbb{R}^n$ and $b_i \in \mathbb{R}$. Note that an equality (2.7) can be expressed by two inequalities (2.5) and (2.6). An inequality of the type '$\leq$' can be turned into a '$\geq$'-inequality by multiplying it with -1, and vice versa.

The linear cost function is also called the *objective function*. By default, we consider minimization problems. One can turn a minimization problem into a maximization problem by multiplying the the cost vector c by -1, and vice versa.

We say that a linear program is in *canonical form* if it is of the form

$$\begin{array}{ll} \min & c^T x \\ \text{s.t.} & Ax \geq b, \\ & x \in \mathbb{R}^n, \end{array}$$

for $A \in \mathbb{R}^{n \times m}$, $c \in \mathbb{R}^n$, $b \in \mathbb{R}^m$. By the above considerations, every linear program can be re-written in canonical form.

A vector $x \in \mathbb{R}^n$ is called a *feasible solution* to the linear program if it satisfies all m constraints. Often, bounds on a variable x_i are imposed. Although this could be modeled by constraints in the matrix A, it is commonplace and useful to write down these constraints separately.

If $x \in \mathbb{R}^n$ is a feasible solution and minimizes the objective, it is called an *optimal feasible solution*, and the objective value it attains is called the *optimum value*. If there is no feasible solution, we call the linear program *infeasible*, and *feasible* otherwise. If the program is feasible but the minimum does not exist, we call the linear program *unbounded*.

2.11.2 *Polyhedra and Extreme Points*

A linear program in canonical form defines a *polyhedron*

$$P = \{x \in \mathbb{R}^n \mid Ax \geq b\},$$

it is the intersection

$$\bigcap_{i \in \{1,\ldots,m\}} \{x \in \mathbb{R}^n \mid a_i^T x \geq b_i\}$$

of 'half-spaces'. If the polyhedron is bounded, it is called a *polytope*. If the linear program is unbounded with respect to the objective, then the polyhedron is unbounded. The reverse does not hold in general.

For points $x_1, \ldots, x_k$ and coefficients $\lambda_1, \ldots, \lambda_k \in [0, 1]$ that satisfy $\sum_{i=1}^k \lambda_i = 1$, the sum

$$\sum_{i=1}^k \lambda_i x_i$$

is called a *convex combination*. A set $X \subseteq \mathbb{R}^n$ is *convex* if every convex combination of two points in X is also in X. The *convex hull* of a set of points is the set of all convex combinations of these points.

It is easy to show that every polyhedron is convex. A point $x \in P$ that cannot be expressed as a convex combination of two points $x_1, x_2 \neq x$ in P is called an *extreme* point. If for a point $x \in P$ there is a *hyperplane*

$$H = \{y \in \mathbb{R} \mid c^T y = b'\}$$

with $x \in H$, but $y \notin H$ for every $y \in P \setminus \{x\}$, then x is called a *vertex* of P.

A linear inequality is *tight* (or *active*) if it holds with equality. For the following discussion, we extend the definition of polyhedra to allow for equalities. A point $x \in \mathbb{R}^n$ is called a *basic solution* if all equality constraints are active at x, and there are at least n linearly independent active constraints for x. If in addition, $x \in P$, it is called a *basic feasible solution*. If there are more than n constraints active at x, it is called *degenerate*.

Theorem 2.11.1. *For a polyhedron $P \subseteq \mathbb{R}^n$ with $x \in P$, the following are equivalent:*

1. *x is an extreme point,*
2. *x is a vertex,*
3. *x is a basic feasible solution.*

2.11.3 *Duality*

Definition 2.11.2 (Dual linear program)**.** Given a linear program

$$\begin{array}{rr} \max & c^T x \\ \text{s.t.} & Ax \le b, \\ & x \ge 0, \end{array}$$

called the *primal* program, the *dual* linear program is given by

$$\begin{array}{rr} \min & y^T b \\ \text{s.t.} & A^T y \ge c, \\ & y \ge 0. \end{array}$$

The following lemma justifies the term 'duality'.

Lemma 2.11.3. *The dual program of the dual program is the primal program.*

Proof. By multiplying the objective function of the minimization problem with -1, one obtains a maximization problem, and by multiplying the constraints with -1, the '$\geq$'-inqualities become '$\leq$'-inequalities. Since $(A^T)^T = A$ and $y^T b = b^T y$ the claim follows. □

By noting that $A^T y = y^T A$, it is straightforward to prove that the optimal objective value of the dual problem is an upper bound on the optimal objective value of the primal problem.

Theorem 2.11.4 (Weak Duality Theorem)**.** *If $x \in \mathbb{R}^n$ is a primal feasible solution, and $y \in \mathbb{R}^m$ is a dual feasible solution, then $c^T x \le y^T b$.*

In particular, if the primal program is unbounded, then the dual program is infeasible.

A central theorem of linear programming theory is that the optimum values are in fact equal.

Theorem 2.11.5 (Strong Duality Theorem [Neu63; GKT51])**.** *If the primal problem has an optimal feasible solution, so does the dual problem, and their optimal objectives are equal.*

A proof of this theorem is possible with a variant of Farkas's lemma.

Theorem 2.11.6 (Farkas's Lemma [Far02])**.** *For $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$, exactly one of the following two systems is feasible:*

$$\{x \in \mathbb{R}^n \mid Ax = b, x \geq 0\},$$
$$\{y \in \mathbb{R}^m \mid A^T y \geq 0, y^T b < 0\}.$$

Corollary 2.11.7. *For $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$, exactly one of the following two systems is feasible:*

$$\{x \in \mathbb{R}^n \mid Ax \leq b, x \geq 0\},$$
$$\{y \in \mathbb{R}^m \mid A^T y \geq 0, y^T b < 0, y \geq 0\}.$$

2.11.4 *Total Unimodularity*

Let us turn to an important special case of constraint matrices. A *submatrix* of a matrix A is created by deleting a collection of rows and columns.

Definition 2.11.8. (Total unimodularity) A matrix A is *totally unimodular* (TU) if every square non-singular submatrix has determinant 1 or -1.

In particular, each entry in a TU matrix is $+1, -1$, or 0.

Lemma 2.11.9. *A matrix A is TU if and only if A^T is TU.*

Lemma 2.11.10 ([HK56])**.** *The incidence matrix of a simple graph G is TU if and only if G is bipartite.*

Theorem 2.11.11 ([HK56])**.** *Let $A \in \{0, 1, -1\}^{m \times n}$ be a TU matrix. For every integral vector $b \in \mathbb{Z}^m$, the vertices of*

$$\{x \in \mathbb{R}^n \mid Ax \leq b, x \geq 0\}$$

are integral.

We say that a polytope is *integral* if all its vertices are integral. Theorem 2.11.11 is useful because of the following fact.

Lemma 2.11.12. *For every nonempty polytope, there is an optimal feasible solution that is an extreme point.*

2.11.5 *Algorithms to Solve Linear Programs*

The simplex method is a classic and often-used algorithm to solve linear programs. While its worst-case running time is exponential, it often performs well in practice. The simplex method was first described by Dantzig [Dan49; Dan51; Dan63] following von Neumann's groundbreaking work in economics [Neu38].

A rough overview of the algorithm (for polytopes) is as follows. Suppose the linear program is feasible and we are given a basic feasible solution. The algorithm moves along the edges of the polytope from vertex to vertex in a direction of reduced cost, i.e., moving along such an edge improves the objective. Once a vertex is reached where no direction of reduced cost exists, we have found an optimal (basic) feasible solution. (This proves Lemma 2.11.12.)

As there may be several directions of reduced cost, one has to choose one using a *pivoting rule*. An iteration takes $\mathcal{O}(m^3 + mn)$ time with Dantzig's rule. However, instances exist where the number of iterations is exponential [KM72]. It is an open question whether variations of the simplex method exist whose runtime is sub-exponential.

Another paradigm for solving LPs are *interior point methods*, which move through the interior of the polyhedron. The first (weakly) polynomial algorithm for solving LPs is the ellipsoid method by Khachiyan [Kha79], who built upon work by Shor [Sho77]. Karmarkar [Kar84] later gave another weakly polynomial algorithm whose practical performance was competitive. The question whether there is a strongly polynomial algorithm for solving LPs is still open.

2.11.6 *Integer Linear Programs*

If all variables are required to take integer values, we speak of *integer linear programs* (ILPs). If this required only for some variables, we speak of *mixed integer linear programs* (MILPs). We shall assume minimization problems in this subsection. The following theorem illustrates how ILPs can model combinatorial problems, and also shows that solving ILPs is presumably hard.

Theorem 2.11.13 ([Kar72])**.** *Solving ILPs is* NP*-complete, even for binary variables.*

Proof sketch. A given solution can be verified in polynomial time. To show NP-hardness, we use a reduction from 3SAT. For every variable x_i of the 3SAT instance $\phi(x_1, \ldots, x_n)$, we use a binary variable x_i in the integer linear program. As a shorthand, let

$$l_{ij} = \begin{cases} x_k, & \text{if } C_{ij} = x_k, \\ (1 - x_k), & \text{if } C_{ij} = \neg x_k. \end{cases}$$

For every clause C_i of the instance, create a constraint

$$l_{i1} + l_{i2} + l_{i3} \geq 1.$$

It is easy to see that ϕ is satisfiable if and only if the ILP has a feasible solution. $\square$

Many combinatorial problems have a straightforward interpretation as ILPs. If the integrality constraints are dropped, the resulting LP is called the *relaxation* of the (M)ILP. A solution to the relaxation is called a *fractional* solution. Feasible fractional solutions provide us with lower bounds to the objective and can help in designing approximation algorithms. For example, a fractional assignment to a variable x can be rounded to the nearest integer. There are also randomized algorithms that interpret such an assignment as a probability if x is a binary variable (possibly after appropriate scaling).

Let us consider two (mixed) integer linear programs on the same variable space that have the same feasible integral solutions with polytopes P_1, P_2 corresponding to their relaxations. Let us further assume for simplicity that the zero vector is a feasible solution for both.

From the standpoint of optimization, if $P_1 \subsetneq P_2$, then P_1 is favorable because there are 'less feasible fractional solutions'. More precisely, if $x \in P_2 \setminus P_1$, without loss of generality a vertex, then there are objective functions such as $\sum_{i=1}^{n} -x_i$ for which $x \in P_2$ is optimal, and no feasible solution in P_1 has the same objective value. Thus, the objective of an optimum feasible solution to P_1 will be closer to the objective value of an optimal *integral* feasible solution than the objective value of x is. Hence, by solving the linear program of P_1 we obtain a better bound. Thus we say that the linear program of P_1 is a *strictly stronger* relaxation for the MILP than the linear program of P_2.

Sometimes, two MILPs have different variable spaces. It may still be possible to compare the strength of their relaxations by an appropriate transformation. For example, one ILP may have one variable x_{uv} for each edge uv in an undirected graph, while another has two variables $y_{(u,v)}, y_{(v,u)}$ for the oriented edges (u,v) and (v,u). By using the projection $x_{uv} = y_{(u,v)} + y_{(v,u)}$, one can compare the polyhedra.

The *integrality gap* is a useful concept for measuring how tight a relaxation is.

Definition 2.11.14. Consider a minimization (M)ILP for instance i that has at least one feasible integer solution. Let $OPT(i)$ and $OPT_f(i)$ denote the optimum feasible integral and fractional solutions, respectively. The ratio

$$\frac{OPT(i)}{OPT_f(i)}$$

is called the *integrality gap* or *integrality ratio* of the instance.[7] The supremum

$$\sup_{i \text{ instance}} \frac{OPT(i)}{OPT_f(i)}$$

over all instances for the problem is called the integrality gap of the (M)ILP.

An integrality gap of at most c often translates into an approximation algorithm with approximation factor c. In reverse, an integrality gap of at least c means that, based on the relaxation alone, we cannot hope for an approximation factor less than c. Of course, using other ideas, one may be able to obtain a factor of less than c.

2.11.7 *Branch-and-Bound and Branch-and-Cut*

In order to solve (M)ILPs, one can use the *branch-and-bound* technique. In the following, let us assume the case of ILPs for a simpler presentation.

Branching refers to the tree that is constructed in the search. The root node corresponds to the relaxation of the given ILP, and is the current node initially. A branching rule creates children for a current node, each with additional constraints such that the optimum solution of the current node must be in one of its children. Ideally, the solution spaces of the children are disjoint.

One branching rule is *variable fixing* that creates children for each of the possible integer values of a chosen variable. For example, if a variable x_i is binary, we can create two branches, one where $x_i = 0$ and one where $x_i = 1$ is mandated. Another rule is *constraint insertion*. For example, for an integer variable x_i we may insert a constraint $x_i \leq k$ for one child and $x_i \geq k + 1$ for the other.

A current node can be 'solved' by branching on it directly or solving the relaxation of its ILP first. If the relaxation is infeasible, we can stop the search in the current node. If the optimum feasible fractional solution is integral, we can also stop the search in this node. It it is not integral, we obtain a lower bound c for the objective value of the ILP in this node. If we somehow know for the global integral objective OPT that $OPT < c$, then we can stop the search in this branch of the tree. Otherwise, we will have to branch.

In order to know that $OPT < c$, we can maintain the value of the best known feasible integral solution, the *incumbent*, because each integral feasible solution provides us with an upper bound on OPT. In order to find such solutions, heuristics can be used, even during the branch-and-bound process. For example, we could apply rounding to

7 If we have $OPT_f(i) = 0$, we use the convention that $\frac{OPT(i)}{OPT_f(i)} = 1$ if $OPT(i) = 0$, and $\frac{OPT(i)}{OPT_f(i)} = \infty$ otherwise.

a feasible fractional solution. If the rounded solution is also feasible, this possibly improves the incumbent.

In order to select the next current node among the unsolved nodes, we need a *search strategy*. There are several search strategies, most commonly depth-first, breadth-first, and best-first. The first two are self-explanatory from Section 2.8. Using the depth-first strategy yields feasible integral solutions faster than breadth-first and is therefore particularly useful when no heuristic for generating good feasible integral solutions is available. The best-first strategy selects the node with the best lower bound, as it is 'the most promising'.

The *cutting-plane method* can be applied if the number of constraints is large, maybe even exponential in the input size. Instead of using all constraints at the beginning, one may start with a smaller subset. Once this restricted program has been solved, one has to determine whether some constraint that has not been used is violated. If no such constraint exists, we are done. Otherwise, we add at least one of these violated constraints and continue solving. Typically, one does not have to add the full set of constraints.

The process of finding violated constraints is called *separation* because a violated constraint is a hyperplane that separates the current feasible solution from the convex hull of the feasible integral solutions.

The cutting-plane method can be integrated into the branch-and-bound method. The result is called the *branch-and-cut* method.

2.12 THE MAXIMUM FLOW PROBLEM

2.12.1 *Flows as Linear Programs*

Given a directed graph $G = (V, E)$ without loops and with two distinguished vertices $s \neq t$, called the *source* and *sink*, and a capacity function $c : E \to \mathbb{R}_0^+$, we call the tuple (V, E, c, s, t) a flow network. Excellent resources are the textbooks [Cor+01; Eve11].

Some authors allow infinite capacities on the arcs as well, which we do not. We call the vertices of the network *nodes* and the edges *arcs* to avoid confusion, unless we want to emphasize that they directly correspond to the vertices and edges of a given graph. Without loss of generality, we can assume no arc enters the source and no arc leaves the sink.[8] This will make the presentation less complicated.

We wish to send *flow* from s to t, i.e., find an assignment $f : E \to \mathbb{R}_0^+$. The source s has an infinite supply of flow and t can absorb an infinite amount of it. An arc can only carry as much flow as its capacity allows, i.e., $f(u,v) \leq c(u,v)$ must hold for every $(u,v) \in E$. If need be, a tuple $(u,v) \notin E$ can be modeled as a zero-capacity arc. For runtime analyses we discard such arcs. Sometimes, we will write $f(v,S)$ for

8 To see this, introduce artificial nodes s', t' with arcs (s', s) and (t, t') of sufficient capacity.

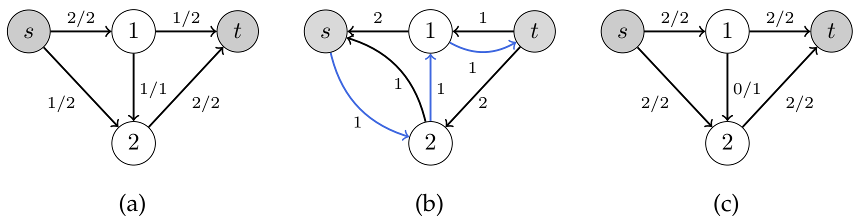


Figure 2.1: (a) A flow of value three. The number left of the slash is the amount of flow the arc carries, the number to its right is the arc's capacity. (b) The flow's residual network. There is an augmenting path $s \to 2 \to 1 \to t$ (blue) that allows augmenting the flow by one. (c) The augmented flow. It is maximum as there is no augmenting path in its residual network (not shown).

$S \subseteq V$ as a shorthand for the total amount of flow going from node v to the nodes in S.

The amount of flow that enters a node $v \notin \{s,t\}$ must be equal to the amount that leaves it. The latter property is called *flow conservation*[9]. A flow is called *integral* if $f(u,v) \in \mathbb{N}_0$ for every arc $(u,v) \in E$. An example of a flow is given in Figure 2.1a. Formally, we can state the maximum flow problem as

$$\begin{aligned}
\max \quad & v && && (2.8)\\
\text{s.t.} \quad & v - \sum_{(v,t)\in E} f(v,t) = 0, && && (2.9)\\
& \sum_{(s,v)\in E} f(s,v) - v = 0, && && (2.10)\\
& \sum_{(u,v)\in E} f(u,v) \\
& \quad - \sum_{(v,u)\in E} f(v,u) = 0, && v \in V \setminus \{s,t\}, && (2.11)\\
& f(u,v) \le c(u,v), && (u,v) \in E, && (2.12)\\
& f(u,v) \ge 0, && (u,v) \in E, && (2.13)\\
& v \ge 0. && && (2.14)
\end{aligned}$$

The amount v that emanates from the source for a feasible assignment f is called the *value* of the flow and denoted by $|f|$. Constraint (2.9) in the formulation is redundant, but adding it makes it easier to establish a connection to the minimum cut problem in the following subsection. If a capacity constraint (2.12) is tight, we call the corresponding arc *saturated*.

9 One could demand flow conservation in s and t as well if an arc (t,s) of sufficiently large capacity is introduced.

2.12.2 *The Minimum Cut Problem*

An s-t-cut in the network is defined as a partition $V = S \mathbin{\dot\cup} T$ with $s \in S, t \in T$. We write (S, T) for short. Given such a cut (S, T), its capacity is defined as

$$c(S,T) := \sum_{u \in S, v \in T} c(u,v).$$

The minimum cut problem is to find a cut in the network with minimum capacity crossing the cut. The maximum flow and minimum cut problems are dual to one another. To see this, we look at the dual program of (2.8)-(2.14).

$$\begin{aligned} \min \quad & \sum_{(u,v)\in E} x_{(u,v)} c(u,v) & & \quad (2.15)\\ \text{s. t.} \quad & y_u - y_v + x_{(u,v)} \geq 0, & (u,v) \in E, & \quad (2.16)\\ & y_t - y_s = 1, & & \quad (2.17)\\ & x_{(u,v)} \geq 0, & (u,v) \in E, & \quad (2.18)\\ & y_v \in \mathbb{R}, & v \in V. & \quad (2.19) \end{aligned}$$

One may wonder how the variables should be interpreted to see that this is indeed a linear program for the minimum cut. As the variables y do not appear in the objective function, and in each constraint, a difference of two such variables appears, one can fix $y_v = 0$ for exactly one $v \in V$ without affecting feasibility or the objective. We do this for $y_s = 0$, and by Constraint (2.17), this determines $y_t = 1$. It is now possible to interpret a variable y_v, when restricted to the range $\{0, 1\}$, as an indicator variable for v being on the t-side of the cut. The following lemma is immediate.

Lemma 2.12.1. *For every cut (S, T), the solution*

$$y_v = \begin{cases} 1, & \text{if } v \in T, \\ 0, & \text{otherwise.} \end{cases}$$

$$x_{(u,v)} = \begin{cases} 1, & \text{if } u \in S \text{ and } v \in T, \\ 0, & \text{otherwise.} \end{cases}$$

is feasible for (2.16)-(2.18), and the objective value (2.15) it attains is the capacity $c(S, T)$.

This lemma shows that the minimum (2.15) is a lower bound to the capacity of any s-t-cut. Furthermore, solutions to the minimum cut LP can be converted into a cut with the same value:

Lemma 2.12.2. *Given a feasible solution to (2.16)-(2.18) with value v, there is a cut (S, T) with capacity $c(S, T) \leq v$.*

In order to prove this, one can use the fact that the minimum cut LP has a TU constraint matrix. Theorem 2.11.11 and Lemma 2.11.12 can then be applied. We can now state the central theorem of network flow theory, the Max-Flow Min-Cut Theorem.

Theorem 2.12.3 (MFMC Theorem [FF56; EFS56])**.** *Let f be a maximum flow and (S, T) be a minimum cut. Then $|f| = c(S, T)$.*

Proof. The LPs (2.8)-(2.14) and (2.15)-(2.18) are always feasible and dual to one another. By the Strong Duality Theorem (Theorem 2.11.5) the optimum values of the LPs are equal. By combining Lemma 2.12.1 and Lemma 2.12.2, the optimum value of the LP (2.16)-(2.18) is the capacity of a minimum cut. □

Without loss of generality, we can work with *net flow*: Given a feasible flow f, for each arc $(u, v) \in E$ we define $\tilde{f}(u, v) := f(u, v) - f(v, u)$. This implies the skew symmetry property

$$\tilde{f}(u, v) = -\tilde{f}(v, u), \qquad\qquad (u, v) \in E,$$

and the capacity constraints and flow conservation are still satisfied. The only difference is that the flow value on an arc may be negative.

Lemma 2.12.4. *For every flow f and any cut (S, T) we have*

$$\tilde{f}(S, T) = |f|.$$

Proof. We use an inductive argument. The claim holds for the cut $(\{s\}, V \setminus \{s\})$ by definition of $|f|$. Let the claim hold for a cut (S, T), and let $x \in T \setminus \{t\}$. If we move x from T to S to obtain (S', T'), the value $\tilde{f}(S, T)$ decreases by $\tilde{f}(S, x)$ and increases by $\tilde{f}(x, T)$.

By skew symmetry and flow conservation, the net increase is thus

$$\tilde{f}(x, T) - \tilde{f}(S, x) = \tilde{f}(x, T) + \tilde{f}(x, S) = 0.$$

Hence, $\tilde{f}(S', T') = |f|$. We can obtain any cut in the network by moving the appropriate nodes from T to S one by one. □

It is possible to determine a minimum cut from any given maximum flow.

Theorem 2.12.5. *Given a maximum flow, a minimum cut in the same network can be determined in linear time.*

This theorem will be proved in the next subsection.

2.12.3 *Maximum Flows by Augmenting Paths*

Theorem 2.12.5 is useful because the maximum flow problem is often not solved with LP solvers that can construct an optimal dual solution simultaneously, but by 'combinatorial' algorithms. The first such method was given by Ford and Fulkerson [FF57]. Imagine we start with the *zero flow*, i.e., $f(u,v) = 0$ for every $(u,v) \in E$. Determine a path of non-saturated arcs $(e_1, e_2, \dots, e_k)$ from s to t, if one exists. Such a path can be found with a depth-first search, where an arc may be traversed unless saturated. Then, let $\Delta = \min_{i=1}^{k} c(e_i) - f(e_i)$. It is now possible to increase the flow on each arc on the path by Δ. Flow conservation and the capacity constraints are maintained, and the value of the flow increases by Δ. Repeating the step until t cannot be reached from s yields an increasingly greater flow value. However, this need not lead to a maximum flow even if the process terminates. Consider Figure 2.1a on page 34: There is no such path from s to t, yet the flow is not a maximum flow. We have to be able to revert previous choices by sending flow in the opposite direction, which causes flow to be 'cancelled out'.

In order to formalize this idea, it is convenient to operate on the residual network. An example is given in Figure 2.1b on page 34.

Definition 2.12.6 (Residual Network)**.** Given a flow network $N = (V, E, c, s, t)$ and a flow f in N, the *residual network* is defined as $N_f := (V, E_r, c_f, s, t)$, where $(u,v) \in E_r$ if $f(v,u) > 0$ or $c(u,v) - f(u,v) > 0$. We set $c_f(u,v) := f(v,u)$ in the former case and $c_f(u,v) := c(u,v) - f(u,v)$ in the latter.

The residual network is quite intuitive to understand. If we find a path $p = (e_1, \dots, e_k)$ in it from s to t, we can *augment* the flow f (as a function on $V \times V$) by the minimum value $\Delta = \min_{i=1}^{k} c_f(e_i)$ on this path. Such a path is thus called an *augmenting path*. The following lemma generalizes this idea.

Lemma 2.12.7. *Let $N = (V, E, c, s, t)$ be a flow network and f be a flow in N (as a function on $V \times V$). Let f_r be a flow in N_f. Then*

$$f'(u,v) = f(u,v) + f'(u,v) - f(v,u)$$

is a flow in N with value $|f'| = |f| + |f_r|$.

The proof is quite technical and can be found in [Cor+01]. The basic Ford–Fulkerson algorithm is now easily formulated: Start with a feasible flow f, for example the zero flow. Construct the residual network N_f. Find an augmenting path in N_f, and augment f with it. Construct the residual network for the augmented flow, and repeat.

The partial correctness of the Ford–Fulkerson method hinges on the following fact, which we shall now prove rigorously.

Lemma 2.12.8. *Let f be a feasible flow in a flow network N. Then f is maximum if and only if there is no f-augmenting path in N_f.*

Proof. '$\Rightarrow$': By contraposition. If there is an f-augmenting path, then f is clearly not maximum, for otherwise it could be augmented by Lemma 2.12.7.

'$\Leftarrow$': If there is no augmenting path, then t cannot be reached from s in N_f. Let $S \subseteq V$ be the set of nodes reachable from s in N_f, and let $T = V \setminus S$. Clearly, (S, T) is a cut. If for $u \in S$ and $v \in T$ there is an arc $(u, v) \in E$, then we have $f(u, v) = c(u, v)$, for otherwise v would be in S. If there is an arc $(v, u) \in E$, then $f(v, u) = 0$, for otherwise v would also be in S. If neither (u, v) nor (v, u) are in E, we have $f(u, v) = 0 = f(v, u)$. Thus,

$$\tilde{f}(S, T) = \sum_{u \in S} f(u, v) - \sum_{v \in T} f(v, u) = \sum_{u \in S} c(u, v) = c(S, T).$$

By Lemma 2.12.4 we can conclude $c(S, T) = |f|$. Assume for the sake of contradiction that f is not a maximum flow. Then there is a maximum flow f' with

$$|f'| > |f| = c(S, T). \tag{2.20}$$

By the MFMC Theorem, $|f'|$ equals the capacity of the minimum cut, which contradicts (2.20).

□

We now return to the question of finding a minimum cut.

Proof of Theorem 2.12.5. If f is maximum, then there is no augmenting path in N_f by Lemma 2.12.8. We can construct the cut (S, T) used in its proof of capacity $|f|$ (i.e., a minimum cut) in linear time with breadth-first search. □

The following lemma will be useful in the runtime analysis of Dinitz's algorithm in Section 2.15.

Lemma 2.12.9. *Given a flow network N with maximum flow value M and some flow f, the maximum flow value M_r in the residual network N_f is equal to $M - |f|$.*

Proof. For any cut (S, T), we have by Lemma 2.12.4

$$|f| = \tilde{f}(S, T) = \sum_{u \in S, v \in T} f(u, v) - \sum_{u \in S, v \in T} f(v, u)$$

and

$$c_f(S, T) = \sum_{u \in S, v \in T} (c(u, v) - f(u, v)) + \sum_{u \in S, v \in T} f(v, u).$$

Combining the two equations yields

$$c_f(S, T) = \sum_{u \in S, v \in T} c(u, v) - |f|. \tag{2.21}$$

If we minimize $c_f(S, T)$, we also minimize the right-hand side of (2.21). As $|f|$ is constant, this implies that a minimum cut in N_f is also a minimum cut in N. Let (S, T) now be a minimum cut in N_f.

By applying the MFMC Theorem twice on (2.21), we have

$$M_r = c_f(S, T) = M - |f|$$

as desired. □

We will now prove total correctness of the Ford–Fulkerson algorithm in case the capacities are integers.

Theorem 2.12.10. *If all capacities are integers, the Ford–Fulkerson algorithm terminates with an integral maximum flow of value M in time $\mathcal{O}(M|E|)$.*

Proof. Clearly, if the capacities are integers, then the increment value Δ in the first iteration is an integer, and the capacities in the residual network of the Δ-augmented flow are integers as well. Inductively, this means that if the algorithm terminates, it finds an integral flow.

As the maximum flow value is an integer, and every augmentation increases the value of the current flow by at least one, the algorithm must terminate after at most M augmentations, each of which takes $\mathcal{O}(|E|)$ time. □

Ford and Fulkerson show that the algorithm need not terminate for irrational capacities [FF10]. If the capacities are rational, we can multiply with the least common multiple of their denominators to obtain integral capacities. After this integral problem has been solved, we can divide again to obtain a maximum flow in the original problem: As multiplication and division by positive numbers correspond to scaling the flow LP, this preserves feasibility and optimality.

2.12.4 *Flow Algorithms and Their Runtimes*

If a flow is decomposed into several flows, of which each only carries flow along a single path, the sum of these pathlengths can be as large as $\Theta(|V||E|)$ [GR98; GT14]. This is called the *flow decomposition barrier* because it implies that algorithms which augment the flow one path at a time, and paths arc-by-arc, have a runtime of $\Omega(|V||E|)$. This does not mean, however, that $\Omega(|V||E|)$ is a lower bound for the maximum flow problem: It does, for example, not apply to solving the linear program for flows, algorithms based on the push-relabel paradigm (e.g., [GT88b]) or algorithms that use special data structures to manipulate entire paths (such as link-cut trees [ST81]).

The first algorithm to break this bound for dense graphs was given by Cheriyan et al. [CHM96], it runs in time $\mathcal{O}(|V|^3/\log|V|)$. It partitions the adjacency matrix of the network into submatrices of size $1 \times \lfloor \log(|V|) \rfloor$, which can then be processed in constant time by table look-ups of $\log(|V|)$-sized integers. After decades of research and many intermediate results, Orlin [Orl13] was able to achieve the runtime $\mathcal{O}(|V||E|)$ in general.

If all capacities are equal to one ('unit capacities'), Dinitz's algorithm runs in

$$\mathcal{O}\left(|E| \min\left(\sqrt{|E|}, |V|^{2/3}\right)\right)$$

time [ET75; Kar73], as we will see in the following sections. These bounds were recently beaten after decades of research in two breakthrough papers by Mądry [Mąd13] and Lee and Sidford [LS14].

Mądry's algorithm uses a primal-dual technique for a matching problem and electrical flow computations. The maximum flow problem is reduced to the matching problem. The algorithm runs in time $\tilde{\mathcal{O}}(|E|^{10/7})$. The Lee–Sidford algorithm, which is an interior-point method, has runtime $\tilde{\mathcal{O}}(|E|\sqrt{|V|})$ on unit capacity networks.

An application of these algorithms (Theorem 4.1.4) renders some of our results in Chapters 6 and 11 (published in [Blu16]) obsolete, although we think they still have some merits.

If the capacities are integers bounded by U, the celebrated binary blocking flow algorithm of Goldberg and Rao [GR98], which generalizes ideas of Dinitz's algorithm, runs in time

$$\mathcal{O}\left(|E| \min\left(\sqrt{|E|}, |V|^{2/3}\right) \log\left(\frac{|V|^2}{|E|}\right) \log U\right).$$

It uses link-cut trees and additional steps such as determining and contracting strongly connected components in order to obtain an acyclic network for the blocking flow algorithm. After the flow computation, the components are de-contracted and the flow is routed inside them.

Mądry [Mąd16] was able to further develop the electrical-flow techniques to obtain a runtime of $\tilde{\mathcal{O}}(|E|^{10/7}U^{1/7})$ in the integer capacity setting, where the Lee–Sidford algorithm has runtime $\tilde{\mathcal{O}}(|E|\sqrt{|V|}\log^2 U)$.

2.13 DINITZ'S ALGORITHM

Dinitz [Din70] and independently, Edmonds and Karp [EK72], introduced the idea of finding shortest augmenting paths, which leads to strongly polynomial algorithms for the maximum flow problem.

Dinitz's algorithm can be formulated in terms of blocking flows, a term coined by Karzanov [Kar74], which allows for simplifications and improvements. For a historical account of these developments, see [Din06]. The following variant of the algorithm has been attributed to Even and Itai by Dinitz [Din06], and it can be found in Even's textbook [Eve11].

A *blocking flow* is a flow where every path from s to t has at least one saturated arc. Note that a maximum flow is always a blocking flow, but the converse does not hold in general. An example is the blocking flow in Figure 2.1a on page 34, a maximum flow of greater value is shown in Figure 2.1c.

Dinitz's algorithm works in phases: It computes a blocking flow in the residual network (starting from the zero flow) in each phase and then updates the residual network for the next phase.

Lemma 2.13.1. *The length of the shortest augmenting path increases*[10] *with every blocking flow phase.*

The number of phases until termination is thus bounded by $|V|$. The total runtime is $\mathcal{O}(|V|B(|V|, |E|))$ where $B(|V|, |E|)$ is the runtime for computing a blocking flow.

Finding a blocking flow is possible in the following way: First, one computes a level graph of the residual network in a breadth-first-search from s in $\mathcal{O}(|E|)$. The i-th level consists of the nodes at distance i from s. Arcs that do not lead to the next level are discarded, while arcs between nodes of the same level are present. In particular, the level network is acyclic. Then, one computes a blocking flow on this level network with depth-first-searches: With each DFS, we try to establish a path from s to t, along which we push flow such that the path becomes blocked, i.e., no more flow can be sent through it. This is done by pushing the minimum of the residual capacities on the path. A move from a node to another in DFS is called an *advance* move.

If at some point, a node v does not have an outgoing arc, delete it and the arc (u, v) through which we entered it. Continue the DFS in u. This is called *retreat*. Any previous advance that we retreat back to is called an *unsuccessful advance* in hindsight. If t is found, push as much flow through the path as possible, i.e., the minimum capacity of its arcs. (The advances made on these arcs are called *successful* advances in retrospect.) All arcs on the path that become saturated (at least one) can be ignored in following searches.

By considering the arcs of a node in the order they appear in its adjacency list, this amounts to traversing the list not more than once. Start the next DFS. If t cannot be found in a search, i.e., all paths are blocked and we backtrack to s, we have established a blocking flow.

Lemma 2.13.2. *A blocking flow can be found in time $\mathcal{O}(|V||E|)$.*

Proof sketch. There can be $|E|$ unsuccessful advances and retreats in the searches at most: if we cannot reach t from a node v, then the arc through which we entered v will be ignored after the retreat. Since at least one arc is saturated with every augmenting path found, there can be at most $|E|$ augmentations. Each augmentation along a path costs $\mathcal{O}(|V|)$, and we can include the cost of successful advances in this estimate. □

Lemmata 2.13.1 and 2.13.2 imply the following theorem.

Theorem 2.13.3 ([Din70]). *Dinitz's algorithm runs in time $\mathcal{O}(|V|^2|E|)$.*

10 The final phase is considered to increase the length to ∞.

Sleator and Tarjan [ST81] introduced the link-cut trees data structure, which can be used to solve the blocking flow problem with a runtime of $\mathcal{O}(|E| \log |V|)$. A variant of Goldberg and Tarjan [GT90] improves this to $\mathcal{O}(|E| \log(|V|^2/|E|))$ time. If the capacities in the network are all equal to one, Dinitz's algorithm can be shown to be even faster. This analysis, which will be used in Chapters 3 to 5, is the subject of the next section.

2.14 ALMOST UNIT CAPACITY NETWORKS

In some applications, all capacities in a flow network are equal to one except for the arcs emanating from the source and the arcs leading to the sink. We now choose to keep $s, t \notin V$.

Definition 2.14.1 (AUC, AUC-2)**.** Let $G = (V, E)$ be a directed graph and let $N = (V \dot\cup \{s, t\}, E \cup E_s \cup E_t, c)$ be a flow network ('G-network') with

$$E_s \subseteq \{s\} \times V, \qquad E_t \subseteq V \times \{t\}, \qquad c : E \cup E_s \cup E_t \longrightarrow \mathbb{N}.$$

N is called an almost unit capacity (AUC) G-network if $c(u, v) \leq 1$ for all $(u, v) \in E$. If merely $c(u, v) \leq 2$ holds, N is called an AUC-2 G-network.

The definition of AUC-2 networks will be needed in the following section because residual networks of AUC networks can have capacities of two per arc[11]. We now prepare the reduction of large source and sink arc capacities. Intuitively speaking, one should send as much flow as possible on the length-2 paths $s \to v \to t$. In the following, if an arc is not present, we assume its capacity is zero.

Lemma 2.14.2. *Let N be a flow network. There exists a maximum flow f in N with*

$$f(s, v), f(v, t) \geq F_v := \min(c(s, v), c(v, t))$$

for all $v \in V$. Let N' denote a copy of N with capacities $c'(s, v), c'(v, t)$ reduced by F_v for $v \in V$. A maximum flow in N' plus the flow F_v on the paths $s \to v \to t$ is one such flow f.

Proof. For every $v \in V$, define $F_v := \min(c(s, v), c(v, t))$. Consider the feasible flow f^- in N where $f^-(s, v) = F_v = f^-(v, t)$ for all $v \in V$ and $f^-(u, v) = 0$ for $(u, v) \in E$. We will now reduce the capacities by these flow values to obtain the flow network N': Define $c' := c - f^-$.

The crucial idea is that any cut in N has exactly the capacity of the corresponding cut in N' plus the values F_v for $v \in V$.

11 This does not happen in the bipartite network (Section 3.6) and the re-orientation network (Section 4.1), where for an arc (u, v) of the original network there never is a reverse arc (v, u). However, for Goldberg's network (Subsection 3.3.3), we need this definition.

Let M denote the value of the maximum flow in N. This is also the capacity of some minimum cut (S,T) in N. In N', the cut (S,T) has capacity

$$\begin{aligned} C'_{(S,T)} &= \sum_{v\in T} c'(s,v) + \sum_{v\in S} c'(v,t) + \sum_{\substack{u\in S\setminus\{s\} \\ v\in T\setminus\{t\}}} c'(u,v) \\ &= M - \sum_{v\in V} F_v. \end{aligned}$$

If (S,T) is also a minimum cut in N', its capacity must equal the maximum flow value of N'. We can then add the flow f^- to a maximum flow of N' to obtain a feasible flow for N with a value of $(M - \sum_{v\in V} F_v) + \sum_{v\in V} F_v = M$. It is thus a maximum flow in N.

It remains to show that (S,T) is indeed a minimum cut of N'. Assume otherwise, i.e., there exists a cut (S^*,T^*) with

$$C'_{(S^*,T^*)} < C'_{(S,T)} = C_{(S,T)} - \sum_{v\in V} F_v. \tag{2.22}$$

Thus,

$$C_{(S^*,T^*)} = C'_{(S^*,T^*)} + \sum_{v\in V} F_v \overset{(2.22)}{<} C_{(S,T)},$$

so (S,T) is not a minimum cut in N, which contradicts the assumption.

□

Proposition 2.14.3. *Let N be an AUC-2 G-network for a directed graph $G = (V,E)$. A maximum flow in N can be found by running linear-time pre- and postprocessing algorithms and invoking any maximum flow algorithm on the obtained AUC-2 G-network $\tilde{N}$ with bounded total source and sink arc capacities $C_{s,t}(\tilde{N}) \in \mathcal{O}(|E|)$.*

Proof. We subtract the value F_v as defined in Lemma 2.14.2 from $c(s,v), c(v,t)$ for every $v \in V$ to obtain c', other capacities are adopted without change.

Since for every v, at least one of these two arcs now has zero capacity, every flow-carrying path $s \rightsquigarrow v \rightsquigarrow t$ must pass through vertices in $V \setminus \{v\}$. Thus, the flow $s \to v$ is bounded by $2\text{outdeg}(v)$ and the flow $v \to t$ is bounded by $2\text{indeg}(v)$.

For any vertex v with $c'(s,v) > 2\text{outdeg}(v)$ we can now safely set $\tilde{c}(s,v) = 2\text{outdeg}(v)$ for all $v \in V$. Likewise, for v with $c'(v,t) > 2\text{indeg}(v)$ we can set $\tilde{c}(v,t) = 2\text{indeg}(v)$ for all $v \in V$. Other capacities are adopted. Call this AUC-2 G-network $\tilde{N}$. Its total source and sink arc capacities are

$$\begin{aligned} C_{s,t}(\tilde{N}) &= \sum_{v\in V} (\tilde{c}(s,v) + \tilde{c}(v,t)) \\ &\le \sum_{v\in V} (2\text{outdeg}(v) + 2\text{indeg}(v)) = 2|E|. \end{aligned}$$

The capacity reduction takes $\mathcal{O}(|V| + |E|)$ time.

□

2.15 DINITZ'S ALGORITHM ON AUC NETWORKS

We now investigate how Dinitz's algorithm [Din70] performs on AUC-2 G-networks. Recall that Dinitz's algorithm works in phases. In each phase, a blocking flow is found. First, we give a generalization of a proposition by Kowalik [Kow06, Proposition 3] (see also [ET75; Kar73; Eve11]).

Proposition 2.15.1. *Each phase of Dinitz's algorithm runs in $\mathcal{O}(|E|)$ on an AUC-2 G-network N if the total source and sink arc capacities $C_{s,t}(N)$ are at most $c|E|$ for some constant $c > 0$.*

Proof. There are at most $2|V| + |E|$ arcs in the network, so the number D of deletions is at most $D \leq 2|V| + |E|$. The total number B of backtrack steps is at most the number of advance steps A, which is also an upper bound on the number of pushes P performed on the arcs. The number of advance steps on an arc is bounded by its capacity. Thus, we have

$$\begin{aligned} A &\leq \sum_{v\in V} c(s,v) + \sum_{(u,v)\in E} c(u,v) + \sum_{v\in V} c(v,t) \\ &= C_{s,t}(N) + \sum_{(u,v)\in E} c(u,v) \leq c|E| + 2|E| = (c+2)|E|. \end{aligned}$$

Therefore, a blocking flow can be found in $A + B + D + P \in \mathcal{O}(|E|)$ steps. $\square$

We next generalize theorems of Even and Tarjan for unit-capacity networks [ET75] (they are independently described by Karzanov [Kar73]) to determine runtime bounds for AUC-2 networks with bounded total source and sink arc capacities. The proofs are analogous. The theorem we desire is the following.

Theorem 2.15.2. *Dinitz's algorithm runs in $\mathcal{O}(|E| \min(\sqrt{|E|}, |V|^{2/3}))$ time on an AUC-2 G-network N with bounded total source and sink arc capacities $C_{s,t}(N) \in \mathcal{O}(|E|)$.*

Lemma 2.15.3. *Let N be an AUC-2 G-network with maximum flow value $M \neq 0$. For the zero flow, the distance from s to t is at most $\frac{2|E|}{M} + 2$.*

Proof. Define

$$V_i := \{v \in V \cup \{s,t\} \mid v \text{ is at distance } i \text{ from } s\},$$

where unreachable vertices ($i = \infty$) are not of interest, and let l denote the distance of t from s. Let E_i denote the set of arcs from V_{i-1} to V_i. Every E_i defines a cut in the network. For $i = 2, \ldots, l-1$, we have

$$2|E_i| \geq M \tag{2.23}$$

because these arcs have a capacity of two at most and M is the value of the minimum cut. However, $2|E_1|$ and $2|E_l|$ may be significantly smaller than M because the arcs may have a capacity larger than two. We account for them separately to arrive at

$$\begin{aligned}2|E| &\geq \sum_{i=1}^{l} 2|E_i| \\ &\geq 2|E_1| + (l-2)M + 2|E_l| \geq (l-2)M,\end{aligned}$$

so we have $l \leq \frac{2|E|}{M} + 2$. □

Theorem 2.15.4. *Dinitz's algorithm runs in $\mathcal{O}(|E|^{3/2})$ time on an AUC G-network N with bounded total source and sink arc capacities $C_{s,t}(N) \leq c|E|$.*

Proof. If $M \leq \sqrt{|E|}$, the result follows from Proposition 2.15.1 since every phase increases the flow by at least one. Otherwise, consider the phase in which the flow value F reaches the value $M - \sqrt{|E|}$. When this phase begins, we have $F < M - \sqrt{|E|}$. The residual network $\tilde{N}$ is an AUC-2 G-network. By Lemma 2.12.9, its maximum flow value is

$$\tilde{M} = M - F > M - \left(M - \sqrt{|E|}\right) = \sqrt{|E|}.$$

Since the flow in the residual network is initially zero, by Lemma 2.15.3, the length of the shortest augmenting path satisfies

$$\tilde{l} \leq \frac{2|E|}{\tilde{M}} + 2 < \frac{2|E|}{\sqrt{|E|}} + 2 = 2\sqrt{|E|} + 2.$$

The number of phases until this point is thus at most $2\sqrt{|E|} + 2$, and since at most $\sqrt{|E|}$ phases are needed until completion, the total number of phases is at most $3\sqrt{|E|} + 2$. Hence, the algorithm needs $\mathcal{O}(|E|^{3/2})$ steps by Proposition 2.15.1. □

Lemma 2.15.5. *Let N be an AUC-2 G-network with maximum flow value $M \neq 0$. For the zero flow, the distance from s to t is at most*

$$\frac{(2\sqrt{2})|V|}{\sqrt{M}} + 1.$$

Proof. Define

$$V_i := \{v \in V \cup \{s,t\} \mid v \text{ is at distance } i \text{ from } s\},$$

where unreachable vertices ($i = \infty$) are not of interest, and let l denote the distance of t from s. We consider the cuts between V_i and V_{i+1} for $i = 0, \ldots, l$. Since M is the value of the minimum cut, the value $C(i, i+1)$ of the cut between V_i and V_{i+1} must be at least M. Therefore, we have $2(|V_i| \cdot |V_{i+1}|) \geq C(i, i+1) \geq M$ for $1 \leq i \leq l-2$ since every arc from V_i to V_{i+1} has a capacity of two at most. Therefore, for every $1 \leq i \leq l-2$, we have $|V_i| \geq \sqrt{M/2}$ or $|V_{i+1}| \geq \sqrt{M/2}$.

We now intend to sum over the cardinalities $|V_i|$ for $i = 0, \ldots, l$. We would like to argue that for any two successive sets, one set has at least

$\sqrt{M/2}$ vertices. In the worst case, the summation sequence would alternate between values less than and greater or equal to $\sqrt{M/2}$. However, in our case it is possible that $|V_1|, |V_{l-1}|$ are significantly smaller than $\sqrt{M/2}$, since source and sink arc capacities can be larger than two. Thus,

$$\frac{l-1}{2} \cdot \sqrt{M/2} \leq 2 + \left\lfloor \frac{l-1}{2} \right\rfloor \cdot \sqrt{M/2} \leq \sum_{i=0}^{l} |V_i| \leq |V|,$$

which leads to

$$l \leq \frac{2|V|}{\sqrt{M/2}} + 1 = \frac{2\sqrt{2}|V|}{\sqrt{M}} + 1.$$

□

Theorem 2.15.6. *Dinitz's algorithm runs in $\mathcal{O}(|V|^{2/3}|E|)$ time on an AUC-2 G-network where the total source and sink arc capacities are bounded as $C_{s,t}(N) \leq c|E|$.*

Proof. If $M \leq |V|^{2/3}$, the result follows since the flow increases at least by one per phase of Dinitz's algorithm. Otherwise, let F be the flow value of the phase in which the flow reaches the value $M - |V|^{2/3}$. We have $F < M - |V|^{2/3}$. By Lemma 2.12.9, the maximum flow value in the residual network is

$$\tilde{M} = M - F > M - (M - |V|^{2/3}) = |V|^{2/3}.$$

Since the flow in the residual network is initially zero, we can apply Lemma 2.15.5: The length of the shortest path satisfies

$$\tilde{l} \leq \frac{2\sqrt{2}|V|}{\sqrt{\tilde{M}}} + 1 < \frac{2\sqrt{2}|V|}{\sqrt{|V|^{2/3}}} + 1 = 2\sqrt{2}|V|^{2/3} + 1.$$

Thus the number of phases up to this point is at most $2\sqrt{2}|V|^{2/3} + 1$, and at most $|V|^{2/3}$ phases remain, for a total of $\mathcal{O}(|V|^{2/3})$ phases. An application of Proposition 2.15.1 yields the desired runtime bound. □

3 THE DENSEST SUBGRAPH PROBLEM

3.1 DEFINITION AND PROPERTIES

The average density, or simply *density*, of a simple graph $G = (V, E)$ is defined as $|E|/|V|$. The density is an important measure in graph theory. In case $|E| \in \Theta(|V|^2)$ a family of graphs is called *dense*. In the case $|E| \in \mathcal{O}(|V|)$ the family is called *sparse*.

Analogously, the density of a subgraph $H = (V_H, E_H) \subseteq G$ is defined as

$$d(H) := \frac{|E_H|}{|V_H|}.$$

Definition 3.1.1. Let G be a simple graph. Its *maximum density* is

$$d^*(G) := \max_{H \subseteq G} \frac{|E_H|}{|V_H|},$$

and the *densest subgraphs* are the subgraphs for which the maximum is attained.

A densest subgraph is always an induced subgraph. The value $2d^*$ is sometimes called the *maximum average degree*.

The densest subgraph problem is the problem of finding a densest subgraph. As we shall see in Chapter 4, there is an algorithm that approximates d^* without returning a subgraph of approximate maximum density, hence one may study the problem of computing the maximum density in its own right.

We note that there is also a notion of maximum density in directed graphs, and some techniques to compute it are similar to the ones used for the undirected case [Chaooa; KSo9a].

A densest subgraph need not be connected. However, finding a connected densest subgraph is not more difficult than finding a densest subgraph. We shall give some structural results for which we need the following lemma.

Lemma 3.1.2. *Let $x, y, a, b > 0$. Then*

$$\frac{x}{y} < \frac{a}{b} \Leftrightarrow \frac{x+a}{y+b} < \frac{a}{b}$$

and

$$\frac{x}{y} = \frac{a}{b} \Leftrightarrow \frac{x+a}{y+b} = \frac{a}{b}.$$

Proof. For the first equivalence, we have

$$\begin{aligned} && \frac{x}{y} &< \frac{a}{b} \\ \Leftrightarrow && xb &< ya \\ \Leftrightarrow && \frac{xb+ab}{ya+ab} &< 1 \\ \Leftrightarrow && \frac{x+a}{y+b} &< \frac{a}{b}. \end{aligned}$$

The proof of the second equivalence is analogous with '$<$' replaced by '$=$'. □

Bălălău et al. [Băl+15, Corollary 4.1] claim that if $S_1, S_2 \subseteq V$ induce densest subgraphs, then $S_1 \cap S_2$ and $S_1 \cup S_2$ also induce densest subgraphs. The claim for $S_1 \cap S_2$ is obviously false for disjoint sets, which is corrected in the following proposition. Our proof is elementary and does not use LP theory. We later found out that a generalization to matroids had been proved earlier by Catlin et al. [Cat+92], they attribute it Tomizawa [Tom76]. The connection to matroids will become apparent in Chapter 8.

Proposition 3.1.3 (Essentially [Tom76; Cat+92]). *If $H_1, H_2 \subseteq G$ are densest subgraphs, then so is $H_1 \cup H_2$. If in addition, H_1 and H_2 are not disjoint, then $H_1 \cap H_2$ is a densest subgraph.*

Proof. Both claims hold if $d^* = 0$. In the following, let $d^* > 0$, and denote $H' = H_1 \cap H_2$.

The first claim is obvious if $H_1 \subseteq H_2$ or $H_2 \subseteq H_1$. Otherwise, let $s_e := |E_{H'}|$, $s_v := |V_{H'}|$ denote the shared edges and vertices of H_1, H_2, neither of which is a subgraph of the other. Let

$$\begin{aligned} a &:= |E_{H_1}|, & x &:= |E_{H_2}|, \\ b &:= |V_{H_1}|, & y &:= |V_{H_2}|, \\ a' &:= a - s_e, & x' &:= x - s_e, \\ b' &:= b - s_v, & y' &:= y - s_v. \end{aligned}$$

If H_1, H_2 are disjoint, we have by Lemma 3.1.2

$$d(H_1 \cup H_2) = \frac{x+a}{y+b} = \frac{x}{y},$$

hence $H_1 \cup H_2$ is a densest subgraph. If the two shared only vertices, but not edges, then the denominator would be smaller than $y + b$. Thus, $H_1 \cup H_2$ would be denser than the densest subgraph, a contradiction.

Therefore, we can assume in the following that the two are not disjoint, and that all quantities are greater than zero. Clearly, we have

$$\frac{s_e + a'}{s_v + b'} = \frac{a}{b} = \frac{x}{y} = \frac{s_e + x'}{s_v + y'}.$$

Assume for the sake of contradiction that

$$\frac{a'}{b'} < \frac{s_e}{s_v}.$$

Thus, by Lemma 3.1.2

$$\frac{a}{b} = \frac{a' + s_e}{b' + s_v} < \frac{s_e}{s_v} = d(H'),$$

a contradiction to the fact that a/b equals the maximum density. Hence,

$$\frac{a'}{b'} \geq \frac{s_e}{s_v}. \tag{3.1}$$

If this holds with equality, we can apply Lemma 3.1.2 to obtain

$$\frac{x}{y} = \frac{a}{b} = \frac{s_e + a'}{s_v + b'} = \frac{a'}{b'},$$

and by another application

$$d(H_1 \cup H_2) = \frac{x + a'}{y + b'} = \frac{x}{y}.$$

Hence $H_1 \cup H_2$ is a densest subgraph. Likewise, if (3.1) is strict, we use Lemma 3.1.2 to get

$$\frac{x}{y} = \frac{a}{b} = \frac{s_e + a'}{s_v + b'} < \frac{a'}{b'}.$$

From this, we derive

$$\begin{aligned}
& & b' &< \frac{ya'}{x} \\
&\Rightarrow & y + b' &< y + \frac{ya'}{x} = \frac{y(x + a')}{x} \\
&\Rightarrow & \frac{x}{y} &< \frac{x + a'}{y + b'} = d(H_1 \cup H_2).
\end{aligned}$$

This is a contradiction to H_2 being a densest subgraph.

We now prove the claim for intersection, where H_1 and H_2 are not disjoint. Again, it is immediate if $H_1 \subseteq H_2$ or $H_2 \subseteq H_1$. We can again assume that all quantities are greater than zero, because if $H_1 \cap H_2$ contained only vertices, its density would be zero.

We saw in the proof of the first claim that (3.1) holds with equality in this situation. Hence, by applying Lemma 3.1.2,

$$d(H_1) = \frac{s_e + a'}{s_v + b'} = \frac{s_e}{s_v} = d(H').$$

This concludes the proof. □

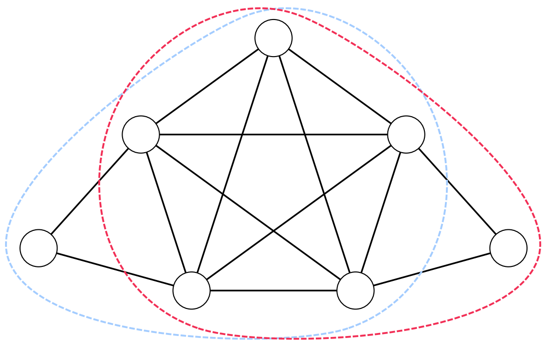


Figure 3.1: The maximum density of the graph shown is two, and it is a densest subgraph itself. The two subgraphs indicated by ovals and their intersection graph are also densest subgraphs.

Proposition 3.1.3 is illustrated in Figure 3.1 on the current page. Valari et al. [VKP12, Lemma 1] claim that for two induced subgraphs H_1, H_2 that share at least one vertex and have the same density, the union $H_1 \cup H_2$ has strictly higher density. This is false, Figure 3.1 gives counterexamples with equal densities. In fact, there are examples of (non-densest) induced subgraphs where the density of the union is strictly smaller although they share vertices (for an example, substitute the subgraph K_5 in Figure 3.1 by K_7). The preceding proof, however, shows that the statement is true if the subgraphs are edge-disjoint. We can now address the matter of finding connected densest subgraphs.

Proposition 3.1.4. *Given a densest subgraph H of G, its connected components are themselves densest subgraphs. In particular, there is always a connected densest subgraph in G, and it can be recovered from a given densest subgraph in $\mathcal{O}(|V| + |E|)$ time.*

Proof. The intuitive idea behind the proof is that if a connected component C of a densest subgraph H had density less than d^*, then removing it from H would result in a subgraph of greater density, which is a contradiction to d^* being maximum.

Let $H \subseteq G$ be a densest subgraph, and let $C_1, \ldots, C_k$ denote its connected components. If all $C_1, \ldots, C_k$ have the same density d, then $d^* = d(\bigcup_{i=1}^k C_i) = d$ by an argument similar to the union case of Proposition 3.1.3. Thus, each C_i is a connected densest subgraph.

In case there are at least two different densities among the C_i, we will deduce a contradiction. Components with the same density are grouped into a single subgraph of the same density. Consider these pairwise disjoint subgraphs $S_1, \ldots, S_l$ in ascending order of density, i.e., $d(S_i) < d(S_{i+1})$ for $i = 1, \ldots, l-1$. Note that $d(\bigcup_{i=1}^l S_l) = d^*$.

Let H_1, H_2, H_3 be disjoint subgraphs with $d(H_1) < d(H_2) < d(H_3)$. By a twofold application of Lemma 3.1.2, we have

$$\begin{aligned} & & d(H_1) < d(H_2) < d(H_3) \\ \Rightarrow & & d(H_1 \cup H_2) < d(H_2) < d(H_3) \\ \Rightarrow & & d((H_1 \cup H_2) \cup H_3) < d(H_3). \end{aligned}$$

By considering the subgraphs $S_1 \cup S_2, S_1 \cup S_2 \cup S_3, \ldots$ and applying this argument inductively, we obtain

$$d^* = d\left(\bigcup_{i=1}^{l} S_l\right) < d(S_l),$$

but this is a contradiction.

The connected components of a densest subgraph can be determined in time $\mathcal{O}(|V| + |E|)$ with depth-first search. □

An easy, yet important consequence is the following lemma, which will be used in Chapter 14 for an MILP formulation.

Lemma 3.1.5. *A simple graph G is a forest if and only if $d^*(G) < 1$. Moreover, the maximum density of a forest equals the average density of the largest connected component (tree).*

Proof. '$\Leftarrow$': If G is cyclic, then its maximum density is at least one by considering the cycle as a subgraph. The claim follows by contraposition.

'$\Rightarrow$' and 'Moreover': First note that a tree of n vertices has an average density of $(n-1)/n$. We can prove that this is indeed its maximum density by induction over the number of vertices in the tree:

The claim is true by inspection for $n = 1$. Consider a tree with $n \geq 2$ vertices, and let the claim hold for all trees with up to $n-1$ vertices. By Proposition 3.1.4, there is a connected densest subgraph, which must be a tree. If it had less than n vertices, its maximum density would be less than $(n-1)/n$ by the induction hypothesis, a contradiction.

If G is a forest, then by Proposition 3.1.4, one densest subgraph is a tree. By the above consideration, the largest connected component (tree) of G has the highest density among all subtrees of G. In particular, the maximum density of G is less than one.

□

Listing all densest subgraphs is, in general, not possible in polynomial time. A simple way of constructing a graph with an exponential number of densest subgraphs is to create n components, each consisting of, say, a 3-cycle. Then, there are $2^n = 2^{|V|/3}$ subgraphs with (maximum) density 1. What can we say about connected graphs?

Proposition 3.1.6. *Consider families of simple connected graphs $G = (V, E)$ with a fixed density $d^*(G)$.*

If $d^*(G) < 1$, there is exactly one connected densest subgraph in the graph.

There is a family with $d^*(G) = 1$ that has $\Omega(2^{|V|})$ connected densest subgraphs, and this is asymptotically maximal.

For every half-integral[1] $d^*(G) > 1$, there is a family whose number of connected densest subgraphs is in $\Omega(2^{|V|/(2d^*+1)})$.

Proof. The case $d^*(G) < 1$ follows from Lemma 3.1.5.

For the case $d^*(G) = 1$, consider a cycle of three vertices and select one vertex v on it. Attach n vertices directly to v via n edges. The cycle itself has a density of one. Any subgraph not containing the cycle has a density of less than one by Lemma 3.1.5, as it is a forest.

Thus, every densest subgraph must contain the cycle. There are 2^n possible ways of selecting vertices attached to the cycle. Selecting a single vertex with its edge raises both the numerator and the denominator of the density ratio by one, thus the density remains one. The subgraphs created by selecting the vertices are connected, their number is in $\Omega(2^{|V|})$. Since there are only $2^{|V|} - 1$ induced subgraphs in any graph, this is asymptotically maximal.

Now consider a half-integral $d^*(G) > 1$. We sketch the argument, it could be made more formal by using ideas from the preceding theorems.

Let $n \geq 4$ be such that $d^* = (n-1)/2 = n(n-1)/(2n)$. Consider a complete graph C on n vertices with density $(n-1)/2$. Further add m copies C'_i of the complete graph on n vertices, each with one arbitrary edge removed. Each of the C'_i has density $(n-1)/2 - 1/n < (n-1)/2$. Link each of these graphs to C via a single edge, the endpoints can be chosen arbitrarily. Call the resulting graph $G(m) = (V, E)$ with $|V| = n(m+1)$. If one vertex from C is chosen, then choosing C in its entirety increases the average density. The same holds for every C'_i. Furthermore, any connected subgraph that contains not only vertices from one of the C'_i must contain at least one vertex from C.

If one chooses C and an arbitrary subset of the C'_i, together with the respective connecting edges, the density is also $(n-1)/2$. It follows from the above discussion that $(n-1)/2$ is indeed the maximum density. As there are 2^m possible ways of adding subsets of the C'_i to C, there are

$$2^{\frac{|V|}{n}-1} = 2^{\frac{|V|}{2d^*+1}-1}$$

densest connected subgraphs. $\square$

1 A number $x \in \mathbb{R}^+$ is half-integral if $x = \frac{k}{2}$ for some $k \in \mathbb{N}$.

3.2 BOUNDS ON THE MAXIMUM DENSITY

An obvious lower bound on d^* is $|E|/|V|$ by considering the whole graph as a subgraph. We will now give better upper bounds than the trivial $|E|$.

Lemma 3.2.1. *For a simple graph $G = (V, E)$ and a densest subgraph $H = (V_H, E_H) \subseteq G$, we have*

$$|V_H| \geq \sqrt{2|E_H| + \frac{1}{4}} + \frac{1}{2}.$$

Proof. A subgraph with $|V_H|$ vertices cannot have more edges than the complete graph on $|V_H|$ vertices, i.e., $|E_H| \leq (|V_H|^2 - |V_H|)/2$. The claim follows by solving for $|V_H|$. □

We are now able to prove that $d^*(G) \in \mathcal{O}(\sqrt{|E|})$, a crucial fact we will exploit in the proof of Theorem 6.0.1.

Proposition 3.2.2. *For a simple graph $G = (V, E)$, we have*

$$d^*(G) \leq \frac{1}{4}\left(\sqrt{8|E|+1} - 1\right) = \sqrt{\frac{|E|}{2} + \frac{1}{16}} - \frac{1}{4},$$

in particular, $d^*(G) \leq \sqrt{|E|/2}$.

Proof. It is easy to verify that

$$\frac{x}{\sqrt{2x + 1/4} + 1/2} = \frac{\sqrt{8x+1} - 1}{4} \tag{3.2}$$

holds for every $x \geq 0$. For a densest subgraph (V_H, E_H) of G, we have by Lemma 3.2.1

$$\begin{aligned} d^*(G) = \frac{|E_H|}{|V_H|} \leq \frac{|E_H|}{\sqrt{2|E_H| + \frac{1}{4}} + \frac{1}{2}} &\overset{(3.2)}{=} \frac{1}{4}\left(\sqrt{8|E_H|+1} - 1\right) \\ &\leq \frac{1}{4}\left(\sqrt{8|E|+1} - 1\right), \\ &= \sqrt{\frac{|E|}{2} + \frac{1}{16}} - \frac{1}{4}. \end{aligned}$$

It is readily verified that $\sqrt{a+b} \leq \sqrt{a} + \sqrt{b}$ for $a, b \geq 0$. Applied to above inequality we obtain $d^*(G) \leq \sqrt{|E|/2}$. □

The bound of Proposition 3.2.2 holds with equality for complete graphs. Asymptotically equivalent bounds have been known before for the arboricity $\Gamma(G)$ and pseudoarboricity $p(G)$ of the graph, which are both upper bounds on $d^*(G)$. We conclude our structural investigations with another bound, which improves over a bound previously known. All these previous bounds will be discussed in Section 8.4.

Proposition 3.2.3. *For a simple graph $G = (V, E)$ we have*

$$d^*(G) \le \Delta(G)/2.$$

Proof. Consider a densest subgraph $H = (V_H, E_H)$ of G. Assume that $d^* = \frac{|E_H|}{|V_H|} > \frac{\Delta}{2}$. We obtain

$$\begin{aligned} |E_H| > \frac{|V_H|\,\Delta(G)}{2} &\ge \frac{\sum_{v\in V_H} \deg_G(v)}{2} \\ &\ge \frac{\sum_{v\in V_H} \deg_H(v)}{2} \overset{(2.1)}{=} |E_H|, \end{aligned}$$

a contradiction. □

An alternative proof via LP theory will be given in Section 3.5.

3.3 ALGORITHMS FOR THE DENSEST SUBGRAPH PROBLEM

Despite the exponential number of (induced) subgraphs of a graph, and possibly an exponential number of densest subgraphs, a densest subgraph can be found in polynomial time. In this section, we will review several existing approaches.

3.3.1 *0-1 Fractional Programming*

Picard and Queyranne [PQ82] formulated the problem as a 0-1 fractional programming problem, which is defined as follows. Let $A, B \subseteq \mathcal{P}(\{1, \ldots, n\})$ and $a_S, b_T \in \mathbb{R}$ for all $S \in A$ and $T \in B$, respectively, with $a_S \ge 0$ if $|S| \ge 2$ and $b_T \le 0$ if $|T| \ge 2$. Consider the problem

$$\begin{aligned} \max \quad & \frac{f(x_1, \ldots, x_n)}{g(x_1, \ldots, x_n)} = \frac{\sum_{S\in A} a_S \prod_{i\in S} x_i}{\sum_{T\in B} b_T \prod_{j\in T} x_j} \\ \text{s.t.} \quad & (x_1, \ldots, x_n) \neq 0, \\ & x_i \in \{0, 1\}, \qquad\qquad i = 1, \ldots, n, \end{aligned}$$

with the promise that for all $x = (x_1, \ldots, x_n) \neq 0$, $g(x) > 0$ and $f(x)/g(x) \ge 0$. The densest subgraph problem is easily seen to be the special case

$$\frac{f(x)}{g(x)} = \frac{\sum_{uv\in E} a_{uv} x_u x_v}{\sum_{v\in V} x_v},$$

where $a_{uv} = 1$ if $uv \in E$ and zero otherwise, and the other variables are defined in a straightforward manner. Note that $f(x)/g(x)$ is actually the maximum average degree, twice the maximum density.

Picard and Queyranne show how the 0-1 fractional programming problem can be solved with a sequence of up to $|V|$ maximum flow computations on a sequence of nested subgraphs of G. We will not review their approach because we will next see algorithms that use $\mathcal{O}(\log |V|)$ flow computations.

3.3.2 *The Provisioning Problem*

As noted (but not explicitly proved) by Kortsarz and Peleg [KP94], the densest subgraph problem can be solved by a (Turing-)reduction to the provisioning problem discussed by Lawler [Law76] (see also [Rhy70; Bal70]).

Let there be items $\{1,\dots,n\}$ of cost c_i for $i = 1,\dots,n$, and sets $S_j \subseteq \{1,\dots,n\}$ for $j = 1,\dots,m$. A set S_j has value v_j if all items contained in it are purchased. What is the optimal choice of items? We can formulate an integer linear program by introducing indicator variables x_i for selecting item i and s_j for selecting item set j as follows:

$$\max \quad \sum_{j=1}^{m} s_j - \sum_{i=1}^{n} c_i \tag{3.3}$$

$$\text{s. t.} \quad x_i \geq s_j, \qquad i \in S_j,\ j = 1,\dots,n, \tag{3.4}$$

$$x_i \in \{0,1\}, \qquad i = 1,\dots,n, \tag{3.5}$$

$$s_j \in \{0,1\}, \qquad j = 1,\dots,m. \tag{3.6}$$

This ILP can be reduced to a flow problem on a bipartite network [Bal70; Law76], which enables us to solve the ILP in polynomial time. In fact, we will see this bipartite flow network in Section 3.6 via a different approach. For this reason, we omit the details and postpone the runtime analysis to that section. We next show that the maximum density problem can be solved via the provisioning problem, which is not immediately obvious.

For some numberings of V and E, let V be the set of items, and let the edges $e \in E$ be the sets S_e. We set the value of an edge to be $v_e = 1$, and the cost of each vertex $c_v = d$ for some $d \in \mathbb{R}_0^+$.

Lemma 3.3.1. *If $c(v) = d$ for all $v \in V$, the ILP (3.3)-(3.6) has a feasible solution of value at least zero if and only if a subgraph of density at least d exists.*

Proof. If there is a subgraph $H = (V_H, E_H)$ of density $|E_H|/|V_H| \geq d$, then setting $x_v = 1$ for all $v \in V_H$ and $s_e = 1$ for all $e \in E_H$ and all other variables to zero clearly gives us a feasible solution. Its objective value is $|E_H| - d|V_H| \geq 0$.

If there is a feasible integral solution (x, s) of value at least zero, then select exactly the vertices and edges whose variable assignments equal one. These form a subgraph $H = (V_H, E_H)$. The objective value of (x, s) is $|E_H| - d|V_H| \geq 0$, hence $|E_H|/|V_H| \geq d$. □

In order to determine d^*, one can use Lemma 3.3.1 and a binary search to find the smallest feasible d. That the binary search terminates after a polynomial number of steps is perhaps not immediately obvious because the maximum density is a rational number. This will be addressed in the review of the next algorithm.

3.3.3 *Goldberg's Method*

Goldberg [Gol84] proposed an approach that solves only $\mathcal{O}(\log |V|)$ flow problems. We will call the cut $(\{s\}, V \cup \{t\})$ in a G-network the *singleton cut*. As we shall see, it requires special attention. The following theorem is illustrated in Figure 3.2.

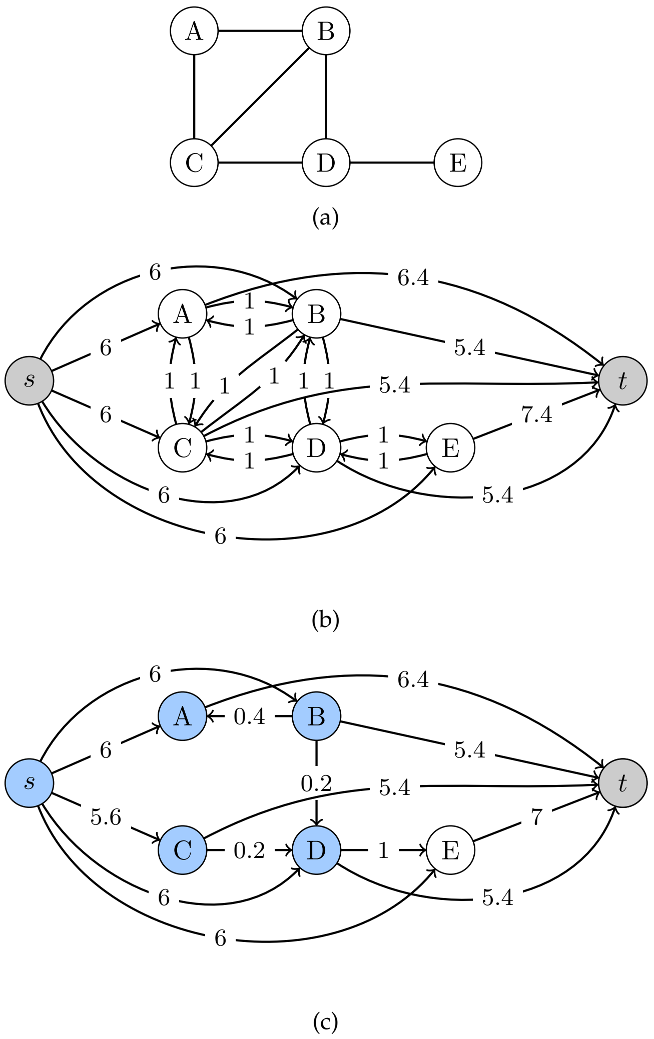


Figure 3.2: (a) A simple graph with $d^* = 1.25$. The vertices $\{A, B, C, D\}$ induce the unique densest subgraph. (b) Goldberg's flow network for test value $d = 1.2$. (c) A maximum flow in the network for $d = 1.2$ (edges that carry zero flow are not shown). A non-singleton minimum cut is indicated by blue vertices. The cut corresponds to a set of vertices that induce a subgraph of density greater 1.2 (here, the densest subgraph.)

Theorem 3.3.2 ([Gol84])**.** *Let $G = (V, E)$ be a simple graph. Define a parameterized flow network $N = (V', E', c, d)$ by*

$$\begin{aligned}
V' &:= V \dot\cup \{s,t\}, \\
E' &:= E \cup \{(s,v) \mid v \in V\} \dot\cup \{(v,t) \mid v \in V\}, \\
c(s,v) &:= |E|, && v \in V, \\
c(u,v) &:= 1, && uv \in E, \\
c(v,t) &:= |E| + 2d - \deg(v), && v \in V.
\end{aligned}$$

A flow of $|V||E|$ is feasible in N if and only if $d \geq d^*(G)$. Moreover, if $d < d^*$, every minimum cut corresponds to a subgraph of density greater d, and if $d = d^*$, every minimum cut except the singleton cut corresponds to a densest subgraph.

Proof. For test value $d \geq 0$, consider an arbitrary cut (S, T) of N and denote $S' = S \setminus \{s\}$ and $T' = T \setminus \{t\}$. If $S' = \emptyset$, then the cut capacity is $|V||E|$, otherwise it is given by

$$\begin{aligned}
c(S,T) &= \sum_{u\in S, v\in T} c(u,v) \\
&= \sum_{v\in T} c(s,v) + \sum_{v\in S} c(v,t) + \sum_{u\in S, v\in T} c(u,v) \\
&= |E||T'| + \left(|E||S'| + 2d|S'| - \sum_{v\in S'} \deg(v)\right) \\
&\quad + \sum_{u\in S', v\in T'} c(u,v) \\
&= |E||V| + 2|S'| \left(d - \frac{\sum_{v\in S'} \deg(v) - \sum_{u\in S', v\in T'} 1}{2|S'|}\right) \\
&= |E||V| + 2|S'|(d - d_{S'}),
\end{aligned} \tag{3.7}$$

where

$$d_{S'} := \frac{\sum_{v\in S'} \deg(v) - \sum_{u\in S', v\in T'} 1}{2|S'|}$$

is exactly the density of the graph $G[S']$.

Consider the case $d < d^*$ and a set $S' \neq \emptyset$ of vertices. If $d_{S'} \leq d$, the cut's capacity is at least the capacity $|V||E|$ by (3.7), which equals the capacity of the singleton cut. If $d_{S'} > d$, then by (3.7) the cut (S, T) has smaller capacity than the singleton cut $(\{s\}, V \cup \{t\})$. There is such a set S', for example the set of vertices of a densest subgraph. Therefore, the singleton cut is not a minimum cut, and a flow of value $|V||E|$ is not feasible by the MFMC Theorem. Moreover, every minimum cut corresponds to a subgraph of density greater than d. (It need not be a densest subgraph because of the factor $|S'|$ in (3.7).)

If $d \geq d^*$, then (3.7) implies that the singleton cut $(\{s\}, V \cup \{t\})$ is a minimum cut. By the MFMC Theorem, the maximum flow value is $|V||E|$. If $d > d^*$, the singleton cut is the only minimum cut. If $d = d^*$, the densest subgraphs are minimum cuts as well, and all other cuts are not minimum. (This last case was not covered by Goldberg.) □

Note that we can, similarly to Lemma 2.14.2, subtract $|E| - \deg(v)$ from the source and sink arcs in order to get the capacities $c(s,v) = \deg(v)$ and $c(v,t) = 2d$. We call this the *modified Goldberg network*. The capacity reduction is not as strong as the one in Lemma 2.14.2, but it can be explicitly stated irrespective of d.

If the graph has nonnegative integer weights on the edges or vertices, or both, we can use essentially the same approach [Gol84]:

Let $w : E \cup V \to \mathbb{N}_0$ be the weight function. If the weights are rational, one can scale them in order to obtain nonnegative integers. For a subgraph $H = (V_H, E_H)$, its density is then defined as

$$d(H) := \frac{\sum_{e \in E_H} w(e) + \sum_{v \in V_H} w(v)}{|V_H|}, \tag{3.8}$$

and we further define the weighted degree $\deg^w(v) := \sum_{uv \in E} w(uv)$. Goldberg's network can be adapted in the following way:

$$\begin{aligned}
V' &:= V \,\dot\cup\, \{s,t\}, && \\
E' &:= E \cup \{(s,v) \mid v \in V\} \,\dot\cup\, \{(v,t) \mid v \in V\}, && \\
c(s,v) &:= m, && v \in V, \\
c(u,v) &:= w(uv), && uv \in E, \\
c(v,t) &:= m + 2d - \deg^w(v) - 2w(v), && v \in V,
\end{aligned}$$

where m is chosen large enough such that no capacity is negative. The correctness proof of this method is analogous to the proof of Theorem 3.3.2. We will, however, only consider the unweighted problem in this thesis.

Equipped with Theorem 3.3.2, one can employ a binary search to find the maximum density by constructing the flow network parameterized with the test value, and finding a non-singleton minimum cut (if one exists). Fortunately, the maximum density is a rational number.

Lemma 3.3.3 ([Gol84]). *If H is a subgraph of G of density d, and no subgraph has a density greater or equal to $d + 1/(|V|(|V|-1))$, then H is a densest subgraph.*

Proof. We have

$$d^* \in \{m/n \mid 0 \le m \le |E|, 1 \le n \le |V|\}.$$

The difference D between two different possible values in this set is

$$D = \frac{m_1}{n_1} - \frac{m_2}{n_2} = \frac{m_1 n_2 - m_2 n_1}{n_1 n_2}.$$

If $n_1 = n_2$, then $m_1 \neq m_2$, and thus

$$|D| \ge \frac{1}{n_1} \ge \frac{1}{|V|} \ge \frac{1}{|V|(|V|-1)}.$$

Otherwise, $n_1 n_2 \leq n(n-1)$, and as the numerator cannot be zero, $|D| \geq 1/(|V|(|V|-1))$. □

Lemma 3.3.3 implies that the binary search can be stopped once the difference between upper and lower bound is at most $1/(|V|(|V|-1))$, as then the subgraph returned from the largest test value $d \leq d^*$ has density d^*.

In the proof of the following theorem, Goldberg makes a subtle mistake: For test value $d = d^*$, the singleton cut $(\{s\}, V \cup \{t\})$ is a minimum cut that does not correspond to a densest subgraph. One has to adapt the minimum cut algorithm in the proof of Theorem 2.12.5 such that a minimum cut *other than* the singleton cut is returned, provided one exists.

Theorem 3.3.4 ([Gol84]). *A densest subgraph of a graph $G = (V, E)$ can be determined in time $\mathcal{O}(M(|V|+2, |E|+2|V|) \log |V|)$, where $M(n, m)$ is the time to determine a maximum flow on a network of n nodes and m arcs.*

Proof. Perform the binary search while the difference between upper and lower bound is greater than $1/(|V|(|V|-1))$. The set of potential test values is thus bounded by $\mathcal{O}(|V|^3)$, and the binary search performs $\mathcal{O}(\log |V|)$ maximum flow computations.

After a maximum flow f has been determined, we need to find a minimum cut other than the singleton cut (if one exists). To this end, we 'mirror' the flow network: t becomes the source and s becomes the sink, and all arcs are directed in the opposite direction. We mirror f analogously, which is feasible because both capacity constraints and flow conservation are satisfied. The cut capacities of the mirrored network are identical to those of the original network, so by an application of the MFMC Theorem, the mirrored flow is maximum. We now determine a minimum cut (T, S) starting from the new source t with BFS as in the proof of Theorem 2.12.5. Unless $(\{s\}, V \cup \{t\})$ is the only minimum cut in the original network, (T, S) must be different from it. Again, by the MFMC theorem, this minimum cut corresponds to a minimum cut in the original network, and it is not the singleton cut $(\{s\}, V \cup \{t\})$. The runtime for mirroring and determining the cut is linear. □

By using a primal-dual technique, Georgakopoulos and Politopoulos [GP07] show how a subgraph of the graph can be removed when a test in the binary search fails. However, they were not able to prove a better runtime estimate based on this. It would be interesting to investigate if this approach translates to the flow-based approaches in Chapter 4, and if the reduced graphs can be aligned with the 'nested graphs' that Picard and Queyranne [PQ82] describe. We do not review their modification here, which has a quite complicated analysis. However, it can be easily implemented, and we will report experimental results for it in Chapter 12.

When Goldberg's paper was written, the fastest known[2] maximum flow algorithm, due to Sleator and Tarjan [ST81], had runtime $\mathcal{O}(|V||E|\log|V|)$, so the total runtime was $\mathcal{O}(|V||E|(\log|V|)^2)$.

Gallo, Grigoriadis and Tarjan [GGT89] describe an algorithm ('GGT') based on the push-relabel paradigm for parametric flow problems where the source and sink arc capacities are nonincreasing and nondecreasing functions of the parameter, respectively. When the parameter is increased, the flow from the previous maximum flow computation can be re-used. This can be applied to Goldberg's network (leading to a 'linear[3] search' over parameter d) for a total runtime of $\mathcal{O}(|V||E|\log(|V|^2/|E|)$. This even holds in the weighted setting (see also [Che95]). Goldberg's runtime estimates for the weighted versions of the problem were worse, yet still polynomial.

However, the GGT algorithm does not generally benefit from the development of faster flow algorithms, as it is based on the push-relabel paradigm. Gusfield and Tardos [GT94] gave an improvement with runtime $\mathcal{O}(|V|^2\sqrt{|E|})$ if the number of flow problems is in $\mathcal{O}(|V|)$. However, the number of possible test values and thus the number of flow problems is much larger in our case (see also Section 6.1).

If we use Goldberg's binary search, the numerators and denominators of capacities can be polynomially bounded, and we can scale the capacities to obtain integral capacities bounded by a polynomially large U. Then, it is possible to use an algorithm tailored to integral capacities (see Subsection 2.12.4).

Theorem 3.3.5. *The densest subgraph problem can be solved in time*

$$\mathcal{O}\left(|E|\min\left(|V|^{2/3}, \sqrt{|E|}\right)\log\left(\frac{|V|^2}{|E|}\right)\log^2|V|\right)$$

with the Goldberg–Rao algorithm and in time

$$\tilde{\mathcal{O}}\left(|E|\sqrt{|V|}\cdot\log^3|V|\right) = \tilde{\mathcal{O}}\left(|E|\sqrt{|V|}\right)$$

with the Lee–Sidford algorithm.

We do not find Mądry's flow algorithm for integer capacities to be useful because its runtime includes a factor of $U^{1/7}$, which is unreasonably large after scaling.

3.4 INTEGRAL TEST VALUES AND SMALLER SEARCH INTERVALS

If we only use integral test values for d^*, Goldberg's algorithm determines $\lceil d^*\rceil$ and an 'almost-densest' subgraph of density greater

2 For dense graphs, several algorithms with runtime $\mathcal{O}(|V|^3)$ were available, e.g., [Kar74].

3 Of course, we can re-use existing flows in a binary search as well by storing the flow found for the last unsuccessful test.

$\lceil d^* \rceil - 1$. In this case, we are able to give slightly better runtime estimates. Interestingly, $\lceil d^* \rceil$ equals the smallest maximum indegree (Section 3.5) and the pseudoarboricity (Chapter 8).

Let us take a look at the factor $\log |V|$ incurred by the binary search. Recall that for fractional test values, there are $\Omega(|V|^2)$ potential values for the search. If we restrict ourselves to integral test values, $\mathcal{O}(\log d^*)$ tests suffice. One way of achieving this is to perform an exponential search [Kow06]: Start testing with $d = 1$ and double d until the first successful test. Then $\lceil d^* \rceil \leq d \leq \lfloor 2d^* \rfloor$ after $\mathcal{O}(\log d^*)$ tests, and the following binary search performs $\mathcal{O}(\log d^*)$ tests when using d as an upper bound.

Another method is to use an algorithm specifically developed for approximation to obtain an upper bound. We will see that a 2-approximation of d^* can be determined in time $\mathcal{O}(|E|)$ in Chapter 7, which reduces the runtime to find a 2-approximating upper bound considerably. However, we will apply a more general technique in Chapter 6 that uses a $(1+\epsilon)$-approximation to obtain an even smaller search interval. As we learned after [Blu16] was accepted for publication, Bezáková had hinted at such a technique for the pseudoarboricity problem in her master's thesis [Bez00] at a time when no approximation scheme was known.

The idea behind the following technical lemma is as follows. If we use a trivial lower bound and a $(1+\epsilon)$-approximation as an upper bound, the search interval size is in $\mathcal{O}(d^*)$. However, if we use the approximation to set a lower bound as well, we can reduce the interval size to $\mathcal{O}(\epsilon d^*)$. Thus, a binary search performs $\mathcal{O}(\log(\epsilon d^*))$ tests.

Lemma 3.4.1. *Let I be an interval of integers, and let $x \in I$ be the minimum value for which a (decidable) property holds. Furthermore, let the property hold for all $x' \in I$ with $x' > x$. Given an approximation $d \in I$ for x with $x \leq d \leq \lceil (1+\epsilon)x \rceil$ for some $\epsilon > 0$, a binary search for x can be realized with $\mathcal{O}(\log(\epsilon x))$ tests.*

Proof. We have $d - 1 \leq (1+\epsilon)x$ and can obtain the lower bound $(d-1)/(1+\epsilon) \leq x$. Let L denote the size of the interval

$$I \cap \{\lceil (d-1)/(1+\epsilon) \rceil, \dots, d-1\}$$

of integral values for x that remain to be checked. We have

$$L \leq (d-1) - \left\lceil \frac{d-1}{1+\epsilon} \right\rceil + 1 \leq (1+\epsilon)x - \frac{d-1}{1+\epsilon} + 1.$$

Now, we apply the bounds $x \leq d$ and $\epsilon > 0$ and obtain

$$\begin{aligned} L &\leq (1+\epsilon)x - \frac{x-1}{1+\epsilon} + 1 \\ &< \frac{(1+\epsilon)^2 x - x}{1+\epsilon} + 2 \\ &= \frac{(\epsilon^2 + 2\epsilon)x}{1+\epsilon} + 2 \\ &= \left(\epsilon + \frac{\epsilon}{1+\epsilon}\right) x + 2 < 2\epsilon x + 2. \end{aligned}$$

Thus, we need to perform $\mathcal{O}(\log(\epsilon x))$ tests in the binary search on I at most. $\square$

Theorem 3.4.2. *To compute $\lceil d^* \rceil$, along with a subgraph of density greater than $\lceil d^* \rceil - 1$, Goldberg's method can be made to run in time*

$$\mathcal{O}\left(|E| \min\left(\sqrt{|E|}, |V|^{2/3}\right) \log d^*\right)$$

with Dinitz's algorithm, and in time

$$\tilde{\mathcal{O}}\left(|E|^{10/7} \log d^*\right) = \tilde{\mathcal{O}}\left(|E|^{10/7}\right)$$

with Mądry's algorithm for unit capacities, and in time

$$\tilde{\mathcal{O}}\left(|E|\sqrt{|V|} \log d^*\right) = \tilde{\mathcal{O}}\left(|E|\sqrt{|V|}\right)$$

with the Lee–Sidford algorithm.

Proof. Obtain a 2-approximation d with the greedy algorithm in Chapter 7. Since d is integral, we have $\lceil d^* \rceil \leq d \leq 2d^* \leq 2\lceil d^* \rceil = \lceil 2\lceil d^* \rceil \rceil$. By applying Lemma 3.4.1 with $x = \lceil d^* \rceil$ and $\epsilon = 1$, performing the binary search is possible with $\mathcal{O}(\log d^*)$ tests (instead of $\log |V|$).

For every integral test value g, we construct Goldberg's network and reduce capacities either by Proposition 2.14.3 or use 'modified network' we described in the previous section. For every test value d in the search, we compute the maximum flow and check whether it saturates all source arcs (Theorem 3.3.2).

For Dinitz's algorithm, Theorem 2.15.2 can be applied for each test. For the Lee–Sidford algorithm, the runtime is immediate. Mądry's algorithm allows parallel arcs in the flow network [Mąd13], i.e., the flow network is a multigraph. After the capacity reduction we can split every source arc (s, v) into $c(s, v)$ parallel unit capacity arcs from s to v. We proceed analogously for the sink arcs. The number of arcs in this modified network is in $\mathcal{O}(|E|)$.

Once we have determined $\lceil d^* \rceil$, a test for $d = \lceil d^* \rceil - 1$ allows us to return a minimum cut as a subgraph of density greater $\lceil d^* \rceil - 1$.[4] $\square$

4 Note that since $\lceil d^* \rceil - 1 < d^*$, we need not exclude the singleton cut here.

With the above theorem, an 'almost-densest subgraph' can be determined faster than a densest subgraph with Theorem 3.3.5. This improvement is not purely theoretical: Dinitz's algorithm on almost unit capacity networks is quite simple and can be expected to be faster in practice than the rather involved Goldberg–Rao flow algorithm (see Subsection 2.12.4) used in Theorem 3.3.5.

The runtime for integral test values can be further reduced (Chapters 5 and 6). To this end, we will establish a relationship of the densest subgraph problem with graph orientations in the following section.

3.5 LINEAR PROGRAMS FOR THE DENSEST SUBGRAPH PROBLEM

Charikar [Chaooa] proposes the following linear programming formulation for computing the maximum density:

$$\begin{aligned}
&\max && \sum_{uv\in E} x_{uv} && && (3.9)\\
&\text{s.t.} && \sum_{v\in V} y_v \le 1, && && (3.10)\\
& && x_{uv} \le y_u, y_v, && uv \in E, && (3.11)\\
& && x_{uv} \ge 0, && uv \in E, && (3.12)\\
& && y_v \ge 0, && v \in V. && (3.13)
\end{aligned}$$

While the LP objective is to select as much of the edges as possible while putting little mass on the vertex variables, it is not immediately clear that the LP's optimum value is equal to d^*. In order to show this, Charikar uses the following lemmata.

Lemma 3.5.1 ([Chaooa]). *For any subgraph H of G, the optimum value of (3.9)-(3.13) is at least $d(H)$.*

Proof. Let $H = (V_H, E_H) \subseteq G$. For every $v \in V_H$, set $y_v = 1/|V_H|$ and for every $uv \in E_H$, set $x_{uv} = 1/|V_H|$. This is a feasible solution to the LP (3.9)-(3.13) with value $|E_H|/|V_H| = d(H)$. □

Charikar next proves that for an LP solution of value v, there exists a subgraph of this density. We add a trivial runtime analysis for the construction of such a subgraph from the LP solution.

Lemma 3.5.2 ([Chaooa]). *Let $G = (V, E)$ be the input graph. Given a feasible solution to (3.9)-(3.13) with value v, a subgraph H with $d(H) \ge v$ can be constructed in time $\mathcal{O}(|E| + |V| \log |V|)$.*

Proof. Consider a feasible solution (x, y) with $\sum_{uv\in E} x_{uv} = v$. Without loss of generality, assume that $x_{uv} = \min(y_u, y_v)$.

For a parameter $r \in [0, 1]$, let

$$\begin{aligned}
V(r) &= \{v \in V \mid y_v \ge r\},\\
E(r) &= \{uv \in E \mid x_{uv} \ge r\}.
\end{aligned}$$

Note that if $uv \in E(r)$, then $u, v \in V(r)$. Moreover, if $u, v \in V(r)$ then $uv \in E(r)$ by the assumption $x_{uv} = \min(y_u, y_v)$. Thus, $E(r)$ is exactly the set of edges induced by $V(r)$. We have

$$\int_0^1 |V(r)|\, dr \leq 1 \tag{3.14}$$

$$\int_0^1 |E(r)|\, dr = \sum_{uv \in E} x_{uv}. \tag{3.15}$$

This is due to the fact that a vertex v raises $V(r)$ by one from 0 to y_v. Therefore, the total area of the integral is the sum $\sum_{v \in V} y_v$, which is bounded by one according to (3.10). Analogously, one shows (3.15).

We now show there exists an $r \in [0, 1]$ such that $|E(r)|/|V(r)| \geq v$, then the subgraph $(V(r), E(r))$ has density at least v. To see this, assume there is no such r. We obtain

$$v = \sum_{uv \in E} x_{uv} \overset{(3.15)}{=} \int_0^1 |E(r)|\, dr < v \int_0^1 |V(r)|\, dr \overset{(3.14)}{\leq} v,$$

a contradiction.

To obtain a set S such that $d(G[S]) \geq v$, sort the values y_v in descending order in time $\mathcal{O}(|V| \log |V|)$. Then, starting from the largest y-value and going down to the smallest, gradually add the vertices v with $y_v \geq r$ to S and grow the set $E[S] = E(r)$ of induced edges simultaneously. This is possible in time $\mathcal{O}(|E|)$. Record the maximum of $|E(r)|/|S(r)|$ during this computation. Then identify a set S that obtained the maximum in a second run of the algorithm, and return it. □

Theorem 3.5.3 ([Chaooa]). *The optimum value of the LP* (3.9)-(3.13) *is* d^*.

Proof. Let OPT denote the optimum value of the LP. By Lemma 3.5.1, $OPT \geq d^*$, and by Lemma 3.5.2, $d^* \geq OPT$. Therefore, $OPT = d^*$. □

In an unpublished extended version of [KS09a], Khuller and Saha consider the LP where (3.10) is required with equality, which affects neither feasibility nor the optimum value.

Theorem 3.5.4 ([KS09b]). *The LP* (3.9)-(3.13), *where* (3.10) *is required with equality, has an optimal solution where all values y_v greater than zero are equal. Furthermore, for any optimal solution, the vertices v with $y_v > 0$ induce a densest subgraph.*

Bălălău et al. [Băl+15] independently show the furthermore-part. A direct consequence is that, given an optimal solution to the modified LP, a densest subgraph can be constructed in time $\mathcal{O}(|E|)$ instead of $\mathcal{O}(|E| + |V| \log |V|)$. Theorem 3.6.2 will show how to achieve this from an optimum solution to the dual LP.

The dual of the LP (3.9)-(3.13) is the following:

$$\begin{aligned}
&\min && d && && (3.16)\\
&\text{s. t.} && d - \sum_{uv\in E} f_{uv,v}, \geq 0, && v \in V, && (3.17)\\
& && f_{uv,u} + f_{uv,v} = 1, && uv \in E, && (3.18)\\
& && f_{uv,u}, f_{uv,v} \geq 0, && uv \in E, && (3.19)\\
& && d \geq 0. && && (3.20)
\end{aligned}$$

Here, we modified the actual dual constraint $f_{uv,u} + f_{uv,v} \geq 1$ to hold with equality, which affects neither feasibility nor the optimum value. It is then possible to reduce the number of variables from $2|E| + 1$ to $|E| + 1$, since we can substitute $f_{uv,u} = 1 - f_{uv,v}$. This is used in our LP implementation in Chapter 12.

The Strong Duality Theorem implies that the dual LP has the same optimum value as (3.9)-(3.13). The dual LP has been independently examined by Cohen [Coh10] and Venkateswaran [Ven04], and has a simple interpretation: every edge uv of the graph can be *fractionally* oriented to its two endpoints with $f_{uv,u}$ and $f_{uv,v}$, and the maximum fractional indegree d is to be minimized.

We can now state an alternative proof to Proposition 3.2.3.

Alternative proof of Proposition 3.2.3. Set $f_{uv,u} = 1/2 = f_{uv,v}$ for every edge and $d = \Delta/2$. The total value of fractional assignments for a vertex $v \in V$ is

$$\sum_{uv\in E} f_{uv,v} = \deg(v)/2 \leq \Delta/2 = d.$$

Hence, (f_u, f_v, d) is a feasible solution and thus, $d^* \leq d = \Delta/2$. □

3.6 THE BIPARTITE ORIENTATION NETWORK

The fractional orientation variables in the previous section were suggestively denoted by f because they correspond to a flow in a certain network. We give an alternative proof of Theorem 3.5.3, which uses the following lemma. A proof sketch is due to Cohen [Coh10]. We will refer to the network defined in the lemma as 'the bipartite network' throughout the thesis. As we noted for the provisioning problem, a generalization was previously known [Bal70; Law76] (see also the remarks in [GGT89]), a weighted version is used by Chen [Che95]. It is also independently described by Aichholzer et al. [AAR95]. An example of the bipartite network can be seen in Figure 3.3 on the following page.

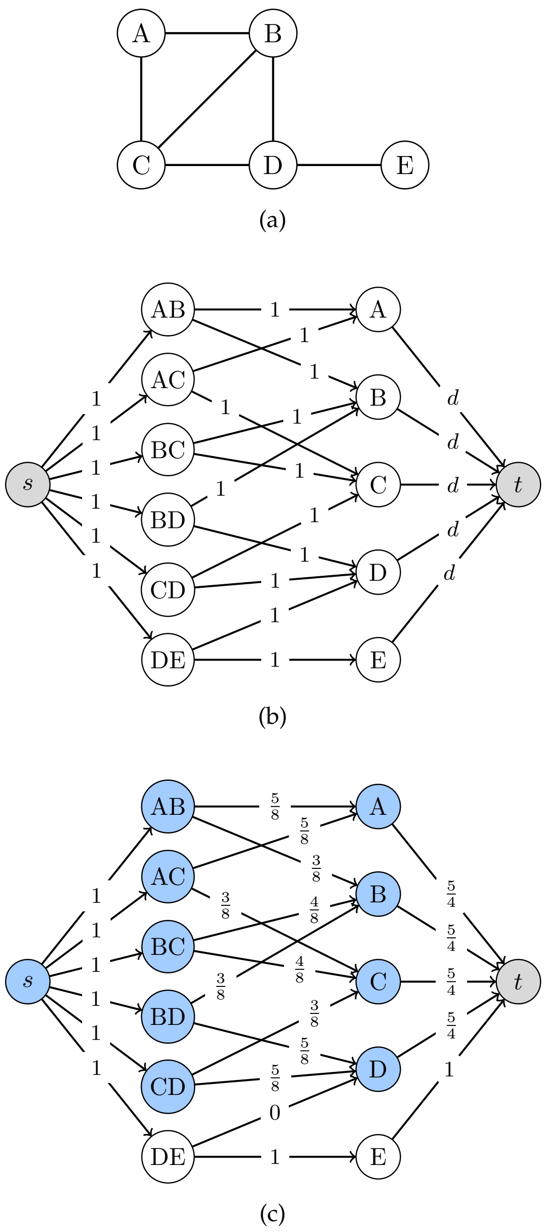


Figure 3.3: (a) A simple graph with $d^* = 5/4$. (b) The corresponding bipartite flow network for a test value d. (c) A maximum flow in the network for $d = 5/4$. A non-singleton minimum cut is indicated with blue vertices. It is returned when mirroring the network and searching in the residual network from t.

Lemma 3.6.1 ([Coh10]). *Let $G = (V, E)$ be a simple graph. For $d \geq 0$, consider the following parameterized flow network $N_d = (V', E', c, d)$, a bipartite graph augmented with a source and sink:*

$$\begin{aligned}
V' &:= E \mathbin{\dot\cup} V \mathbin{\dot\cup} \{s,t\},\\
E' &:= \{(e,u) \mid uv = e \in E\} \mathbin{\dot\cup} \{(s,e) \mid e \in E\} \mathbin{\dot\cup} \{(v,t) \mid v \in V\},
\end{aligned}$$

with capacities

$$\begin{aligned}
c(s,e) &:= 1, && e \in E,\\
c(e,u) &:= 1, && e \in E, u \in e,\\
c(v,t) &:= d, && v \in V.
\end{aligned}$$

There is a feasible (and maximum) flow of value $|E|$ in N_d if and only if $d \geq d^*(G)$.

Proof. Let $H = (V_H, E_H)$ be a densest subgraph of G.

'$\Rightarrow$': If a flow of value $|E|$ is feasible, then $|E_H|$ will be sent from the sink to the edge nodes corresponding to E_H. It is then propagated to the nodes corresponding to V_H. Since each vertex node can pass at most d to the sink, we must have $d|V_H| \geq |E_H| \Leftrightarrow d \geq \frac{|E_H|}{|V_H|} = d^*$.

'$\Leftarrow$': If $d \geq d^* = \frac{|E_H|}{|V_H|}$, we show that the cut through the arcs from s to the edge nodes is a minimum cut with value $|E|$. By the MFMC Theorem, the value of the minimum cut is equal to the value of the maximum flow, so the flow of value $|E|$ is feasible.

Consider an arbitrary cut of the network. Let T be the set of edge nodes on the t-side of the cut. These contribute $|T|$ to the cut capacity by their incoming arcs. Let $S \subseteq V$ be the set of vertices whose corresponding vertex nodes are on the s-side of the cut. These contribute $|S|d$ to the cut. Note that for the set of edges induced by S in G, $E[S]$, we have $d \geq |E_H|/|V_H| \geq |E[S]|/|S|$. The total number of outgoing edges of the edge nodes is $2|E|$. To obtain the true value they contribute to the cut with their outgoing arcs, we have to subtract from this number as follows. Subtract $2|T|$ for the edge nodes on the t-side. Subtract at most $2E[S]$ for the edges induced by S: if such an edge is on the s-side, its outgoing arcs do not leave the s-side, if it is on the t-side, it is part of T and has already been accounted for. Let $\overline{E[S]}$ denote the set of edges not induced by S, but whose edge nodes are on the s-side of the cut. Such an non-induced edge has either one or two end vertices on the t-side. The latter each contribute two to the cut, so we only have to subtract one for the edges that have exactly one end in S. Their number is at most $\overline{E[S]}$. Thus, the cut capacity C is at least

$$\begin{aligned}
C &\geq |S|d + |T| + (2|E| - 2|T| - 2|E[S]| - |\overline{E[S]}|)\\
&\geq |E[S]| + 2|E| - |T| - 2|E[S]| - |\overline{E[S]}| \qquad (3.21)\\
&= 2|E| - (|T| + |E[S]| + |\overline{E[S]}|) \geq |E|.
\end{aligned}$$

□

If $d \geq d^*$, d and a maximum flow in the network N_d correspond a feasible solution to the LP (3.17)-(3.20): The flow sent from the source to an edge node is exactly one and must be split to the two endpoints of the corresponding edge, modelling (3.18). The vertices can absorb d at most, which is modeled in (3.17).

Likewise, if (f_u, f_v, d) is a solution to the LP, then it corresponds to a maximum flow of value $|E|$ in the network N_d. Using a maximum flow algorithm in a binary search for the minimum feasible d allows for better runtimes than solving the dual LP with linear program solvers. Can we construct a densest subgraph from an optimal dual solution?

Theorem 3.6.2. *Let a maximum flow in the bipartite network be given for test value $d = d^*$. For any densest subgraph $H = (V_H, E_H)$, the cut $C_H = \{s\} \cup E_H \cup V_H$ is a minimum cut in the bipartite network. Apart from these cuts and the singleton cut through the source arcs, there are no minimum cuts for this test value.*

Proof. The arcs going from s to the edge nodes corresponding to $E \setminus E_H$ contribute $|E| - |E_H|$ to the cut C_H. The arcs going from the vertex nodes corresponding to V_H to t each have capacity $d^* = |E_H|/|V_H|$, so they contribute $|E_H|$ to the cut in total. Thus, the cut capacity is $|E|$. By Theorem 3.6.1, this is the minimum capacity.

To see that all other cuts except the singleton cut have a higher capacity, let $S \subseteq V$ be the set of vertex nodes on the s-side and $T \subseteq E$ be the set of edge nodes on the t-side.

If $|S| = 0$, every edge node contributes one to the cut if it is on the t-side, otherwise it contributes two. Therefore, only the cut $\{s\}$ has capacity $|E|$. (We do not consider $G[\varnothing]$ to be a subgraph.)

If $|S| \geq 1$, and it is not the set of vertices of a densest subgraph, then $|E[S]|/|S| < d^* = d$, so $|S|d > |E[S]|$. Thus, we see strict inequality in (3.21), so the cut's capacity exceeds $|E|$ and it is thus not minimum.

If S is a set of vertices and $E \setminus T \neq E[S]$, i.e., the cut does not represent an induced subgraph, we can show that its capacity also exceeds $|E|$: The nodes corresponding to S contribute $|S|d$. Every edge node not on the s-side of the cut contributes one to the cut. If it is in $E[S]$, this means an excess compared to the situation of the cut corresponding to an induced subgraph (where the edge node contributes zero to the cut), regardless of it being densest or not. If it is not in $E[S]$, there is no change with respect to the situations discussed above. If an edge node not in $E[S]$ is on the s-side of the cut, it adds an additional two to the cut capacity. Whether $(S, E[S])$ is a densest subgraph or not, in both cases the cut capacity exceeds $|E|$ by above considerations. We can determine a non-singleton cut by 'mirroring' the flow network as we did in the proof of Theorem 3.3.4. $\square$

In contrast to Goldberg's method for test values $d \leq d^*$, it is unclear how to obtain a subgraph of density at least d unless we hit $d = d^*$

exactly. On the other hand, we can determine a fractional d-orientation for any $d \geq d^*$.

By restricting the variables in the LP to integral values, we obtain an ILP for the problem of finding the smallest integer d such that an integral d-orientation exists.

Definition 3.6.3. Let G be a simple graph. The smallest integer d such that a d-orientation $\vec{G}$ of G exists is called the *orientation number* $p(G)$, and $\vec{G}$ is an *optimal orientation*. The problem of determining an optimal orientation is called the *orientation problem*.

We can now see that an optimal orientation can be found efficiently despite the NP-completeness of ILPs (with only binary variables) in general (Theorem 2.11.13).

Corollary 3.6.4. *The orientation problem can be solved in $\mathcal{O}(|E|^2 \log p)$ time.*

Proof. Use an exponential search to obtain a 2-approximation of p, followed by a binary search. In each test, construct the bipartite network and find an integral maximum flow with the Ford–Fulkerson algorithm. Since the maximum flow value is bounded by $|E|$, its runtime is $\mathcal{O}(|E|^2)$. We report a test to be successful if the maximum flow value is exactly $|E|$. Correctness follows from Lemma 3.6.1. □

Aichholzer et al. [AAR95] claim that a runtime of $\mathcal{O}(|E|^{3/2} \log d^*)$ is achieved with the Hopcroft–Karp algorithm [HK73] for bipartite matching.[5] They do not elaborate what is to be done in the analysis about the arcs to the sink – in the straightforward reduction from the bipartite matching problem to the maximum flow problem, all arcs have unit capacity. We will address this issue in Chapter 5 for Dinitz's algorithm with the theory of AUC networks.

Venkateswaran [Ven04] shows that if variable d in the dual LP is transformed into a constant, then the parameterized LP (with then constant objective function) has a totally unimodular constraint matrix. He also gives an example where the original matrix is not TU.

We give a simple alternative proof of total unimodularity based on Lemma 2.11.10 and the relationship between the dual LP and the bipartite network.

Proposition 3.6.5 ([Ven04])**.** *If d is fixed to be a constant in the LP* (3.17)-(3.20)*, the constraint matrix is TU.*

Proof. Note that if d is fixed as a constant, it appears on the right-hand side of the LP, not in the constraint matrix. We will refer to the constraints by their original counterparts. Consider the bipartite

5 The Hopcroft–Karp algorithm, which is generalized by Dinitz's algorithm, runs in time $\mathcal{O}(m\sqrt{n})$ for n nodes and m arcs. Note that $n = |E| + |V|$ and $m \in \mathcal{O}(|E|)$ in our bipartite network.

network (without s and t) as an undirected graph G' with $|V| + |E|$ vertices. Each edge node has two incident edges that correspond to $f_{uv,u}$ and $f_{uv,v}$. These $|E|$ rows in the incidence matrix of G' are exactly the rows for Constraints (3.18).

Every vertex node v has an edge incident to it for every edge incident to v in the original graph; this edge corresponds to $f_{uv,v}$. Thus, these $|V|$ rows are exactly the rows for Constraints (3.17) with the occurrence of d removed. By Lemma 2.11.10 this matrix is TU. □

Proposition 3.6.5 and Theorem 2.11.11 imply the following corollary.

Corollary 3.6.6. *For every integer $d \geq d^*$, there is a (non-fractional) d-orientation.*

The orientation problem can alternatively be solved in polynomial time by employing a binary search and a polynomial-time LP solver that determines an optimal extreme point. Note that Corollary 3.6.6 also follows from the bipartite network for test value $\lceil d^* \rceil$ by invoking Theorem 2.12.10 on integral flows.

By exploiting the duality of the densest subgraph problem and the smallest maximum fractional indegree problem, we obtain a simpler proof than Venkateswaran for the following theorem.

Theorem 3.6.7 ([AAR95; Bez00; Ven04]). *Let $G = (V, E)$ be a simple graph. Then the smallest maximum indegree $p(G)$ equals $\lceil d^*(G) \rceil$.*

Proof. By Theorem 3.5.3 and observing that the dual LP describes the fractional orientation problem, we have $d^* \leq p$. If we use $\lceil d^* \rceil$ as a feasible test value, Corollary 3.6.6 shows that there is a $\lceil d^* \rceil$-orientation, i.e., $p \leq \lceil d^* \rceil$. Thus $p = \lceil d^* \rceil$. □

We will write $\lceil d^* \rceil$ instead of p in the remainder of the thesis except Chapters 8-10 that deal with pseudoforest partitions. There, we will write p because it equals the pseudoarboricity.

Theorem 3.6.7 is independently proved by an application of Hall's theorem [Hal35] by Aichholzer et al. [AAR95] and by induction and path reversals by Bezáková [Bez00]. We will see another proof of Theorem 3.6.7 based on matroid theory in Chapter 8. As Kowalik [Kow06] notes, the following theorem can also be employed. It was proved independently by Hakimi [Hak65] and Frank and Gyárfás [FG78]. We present the proof by the latter, which is simpler and can be easily turned into an efficient algorithm.

Theorem 3.6.8 ([Hak65; FG78]). *Let $G = (V, E)$ be a simple graph and $b : V \to \mathbb{Z}$. G has an orientation $\vec{G}$ with $\mathrm{indeg}_{\vec{G}}(v) \leq b(v)$ for all $v \in V$ if and only if*

$$|E[S]| \leq \sum_{v \in S} b(v) \quad \forall S \subseteq V. \tag{3.22}$$

Proof. '$\Rightarrow$': Let $\vec{G}$ be an orientation with $\operatorname{indeg}_{\vec{G}}(v) \leq b(v)$ for all $v \in V$. Clearly,

$$|E[S]| \leq \sum_{v\in S} \operatorname{indeg}_{\vec{G}}(v) \leq \sum_{v\in S} b(v).$$

'$\Leftarrow$': Let $\vec{G}$ denote an arbitrary orientation of G. Let V^* be the set of vertices $v \in V$ with $\operatorname{indeg}_{\vec{G}}(v) > b(v)$. Define the quality of an orientation $\vec{G}$ to be

$$q(\vec{G}) := \sum_{v\in V^*} \operatorname{indeg}_{\vec{G}}(v) - b(v).$$

We now prove that if (3.22) holds, then we can either modify $\vec{G}$ such that $q(\vec{G}) = 0$, which proves the claim, or find a set S that violates (3.22), which constitutes a contradiction.

Unless $q(\vec{G}) = 0$, let $u \in V$ be a vertex with $\operatorname{indeg}_{\vec{G}}(u) > b(u)$, and let S be the set of all vertices that can be reached from u in the opposite direction of the edges. (For example, if $u \leftarrow v \leftarrow w$, then $v, w \in S$.)

If there is a vertex $v \in S$ with $\operatorname{indeg}_{\vec{G}}(v) < b(v)$, reverse the path from v to u. This improves the quality measure q by one as all vertices on the path except u and v do not change their indegree, u does not enter V^*, and v's indegree drops by one.

If no such vertex exists, then $\operatorname{indeg}_{\vec{G}}(v) \geq b(v)$ for every $v \in S$. As no edge enters S, we have

$$|E[S]| = \sum_{v\in S} \operatorname{indeg}_{\vec{G}}(v) > \sum_{v\in S} b(v).$$

This contradicts (3.22). By applying the argument repeatedly, we can reduce q to zero or arrive at a contradiction. □

Note that the theorem is quite general because it allows different indegree bounds for the vertices. We can compute such an orientation if it exists by altering the sink arc capacities in the bipartite network to the function b. We can do the same for a function $b : V \to \mathbb{R}$ and fractional orientations. It stands to reason that the theorem also holds in this case. However, the above proof does not work because the improvement of the quality measure could be arbitrarily small and thus q may never reach zero. Recall from Section 2.12.4 that the Ford–Fulkerson algorithm suffers from the same problem for irrational capacities. We shall not pursue the matter further. Let us instead give an alternative proof of Theorem 3.6.7.

Alternative proof of Theorem 3.6.7. Set $b(v) = \lceil d^* \rceil$ for every $v \in V$. For nonempty $S \subseteq V$, we can rewrite (3.22) equivalently as

$$d(G[S]) = \frac{|E[S]|}{|S|} \leq \lceil d^* \rceil .$$

This is true by the definition of the maximum density. By Theorem 3.6.8, an orientation $\vec{G}$ with $\text{indeg}_{\vec{G}}(v) \leq \lceil d^* \rceil$ for all $v \in V$ exists.

To see that this upper bound is the smallest possible, set $b(v) = \lceil d^* \rceil - 1 < d^*$ to arrive at a contradiction by considering the densest subgraph and (3.22). $\square$

The proof of Theorem 3.6.8 inspires how $\lceil d^* \rceil$ can be found algorithmically without knowledge of maximum flow algorithms. We will call this the *path reversal algorithm*, which was described independently by Venkateswaran [Ven04] and Asahiro et al. [Asa+07]. Further properties of this algorithm were examined by Borradaile et al. [Bor+17].

Starting from an arbitrary orientation, take a vertex u with maximum indegree d_{max} and search for path $u \leftarrow \cdots \leftarrow v$ to a vertex v with indegree at most $d_{max} - 2$. Reverse the path. The indegree of u and v is then at most $d_{max} - 1$, and the indegrees of the remaining vertices on the path do not change. Repeat this step (updating d_{max} when necessary) until no such reversible path can be found. Reversible paths can be interpreted as augmenting paths in a flow network. Frank and Gyárfás [FG78] already mentioned that network flow theory can be applied, but did not elaborate.

Bezáková [Bez00] describes a similar algorithm that starts with a lower bound to $\lceil d^* \rceil$ (e.g., zero) and all edges unoriented. In order to orient an unoriented edge uv, first arbitrarily orient it to v. If the lower bound becomes violated thereby, search for a reversible path from v as described, and reverse it. If no such path exists, the lower bound is raised by one. Once all edges have been oriented, the lower bound is reported as $\lceil d^* \rceil$.

Theorem 3.6.9 ([Bez00; Ven04; Asa+07])**.** *The orientation problem can be solved in time $\mathcal{O}(|E|^2)$ with algorithms based on path reversals.*

We note that the runtime analysis of Bezáková's algorithm is particularly easy, as $|E|$ edges are oriented, and searches for reversible paths take $\mathcal{O}(|E|)$ with BFS.

A trivial analysis of the path-reversal algorithm starting with an arbitrary orientation yields a runtime bound of $\mathcal{O}(|V|^2|E|)$: The maximum indegree can be reduced at most $|V|$ times, there are at most $|V|$ vertices with current maximum indegree, and reducing the maximum indegree of a vertex is possible in $\mathcal{O}(|E|)$. Instead of proving the $\mathcal{O}(|E|^2)$ runtime as in [Ven04; Asa+07], we note the following interesting application of a constant-factor approximation: If one uses a 2-approximating orientation as the initial orientation, the maximum indegree can drop only $\mathcal{O}(\sqrt{|E|})$ times by Lemma 3.2.2. In this way the runtime can be analyzed to be $\mathcal{O}(|V||E|^{3/2})$.

It is possible to use a flow algorithm for finding reversible paths from all vertices with maximum indegree at the same time, which leads to an algorithm with runtime $\mathcal{O}(|E| \min(\sqrt{|E|}, |V|^{2/3})\sqrt{|E|})$ using

Dinitz's algorithm when starting from a 2-approximating orientation. The flow can be interpreted as an 'overlay of reversed paths', i.e., an edge may be flipped several times by different paths. A similar, but better approach based on binary search will be given in the following chapter, therefore we do not delve into the details here.

3.7 STREAMING ALGORITHMS

A graph stream is a graph that may be subject to change, typically by single edge insertions and deletions. If both are allowed, the problem of maintaining a solution under these changes is called *fully dynamic*. We give a succinct literature review here.

Bahmani et al. [BKV12] show how a $(2+\epsilon)$-approximation to the densest subgraph can be found in $\mathcal{O}(\log |V|\epsilon^{-1})$ passes and $\mathcal{O}(|V|)$ space. They also address the case of directed graphs and test a MapReduce implementation.

Bhattacharya et al. [Bha+15a; Bha+15b] show how to maintain a $(4+\epsilon)$-approximating subgraph with high probability in the fully dynamic setting in $\tilde{\mathcal{O}}(1)$ amortized time, while queries take $\tilde{\mathcal{O}}(1)$ time. It uses $\tilde{\mathcal{O}}(|V|)$ space. An approximation factor of $(2+\epsilon)$ can be traded for a query time of $\tilde{\mathcal{O}}(|V|)$.

Epasto et al. [ELS15] present a fully dynamic algorithm that maintains a $(2+\epsilon)$-approximating subgraph whose amortized runtimes are $\tilde{\mathcal{O}}(\epsilon^{-2})$ for insertion and $\tilde{\mathcal{O}}(\epsilon^{-4})$ for deletion with high probability. It requires $\mathcal{O}(|V|+|E|)$ space.

McGregor et al. [McG+15b; McG+15a] present a single-pass algorithm that returns a $(1+\epsilon)$-approximating subgraph with high probability. An update takes $\tilde{\mathcal{O}}(1)$ time, and it needs $\tilde{\mathcal{O}}(|V|\epsilon^{-2})$ space. The desired subgraph is constructed with a polynomial-time algorithm such as Goldberg's.

Angel et al. [Ang+14] address the problem of changing edge weights. Das Sarma et al. show how to maintain a densest subgraph under a certain condition [Das+12].

4 THE ORIENTATION PROBLEM

4.1 THE RE-ORIENTATION ALGORITHM

In the previous chapter, we saw that the dual (3.16)-(3.20) of the LP for the densest subgraph problem is the relaxation of the smallest maximum indegree orientation problem, and that it can be solved in polynomial time by performing a binary search and solving a bipartite flow problem.

There is yet another flow network defined on the original graph that has several advantages. We call the algorithm that uses it the *re-orientation algorithm*. It has been described independently by Bezáková [Bez00], Kowalik [Kow06], and Asahiro et al. [Asa+07].

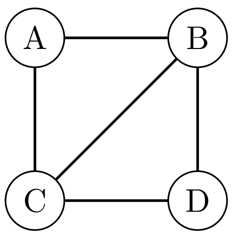
Unlike the bipartite network, where an orientation is computed 'from scratch' (the edges are not oriented initially), we start with an arbitrary orientation $\vec{G}$ of the graph. For some test value d, we try to re-orient $\vec{G}$ such that every vertex has at most d ingoing edges (Figure 4.1 on the next page). In order to do so, we add a source with arcs to every vertex that has more than d ingoing edges: this vertex has to flip at least $\text{indeg}_{\vec{G}}(v) - d$ ingoing edges (more, if outgoing edges flip are flipped well) to lower its indegree to d. Likewise, we add a sink with arcs from every vertex whose indegree is less than d. The indegrees of these vertices may rise up to d by flipping edges.

Definition 4.1.1. For an orientation $\vec{G} = (V, \vec{E})$ of a simple graph $G = (V, E)$ and $d \geq 0$, the *re-orientation network* $(V \,\dot{\cup}\, \{s,t\}, A, c)$ is given as follows:

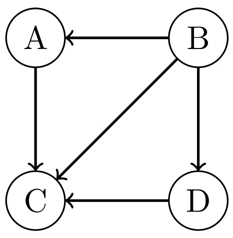
$$\begin{aligned}
(u,v) \in A : c(u,v) &:= 1 && \Leftrightarrow u \leftarrow v \text{ in } \vec{G},\\
(s,v) \in A :\ c(s,v) &:= \text{indeg}_{\vec{G}}(v) - d && \Leftrightarrow \text{indeg}_{\vec{G}}(v) \geq d,\\
(v,t) \in A :\ c(v,t) &:= d - \text{indeg}_{\vec{G}}(v) && \Leftrightarrow \text{indeg}_{\vec{G}}(v) < d.
\end{aligned}$$

Recall Theorem 2.12.10: If the capacities are all integral, which is the case if d is an integer, there is an integral maximum flow. Let us run an algorithm that determines an integral maximum flow. A flow of 1 through an arc other than the source and sink arcs means that the corresponding edge should be reversed, while a flow of 0 means it should point in the original direction of $\vec{G}$. In the relaxation of the orientation problem, we have fractional test values d, and the (possibly non-integral) flow determines a fractional re-orientation.[1]

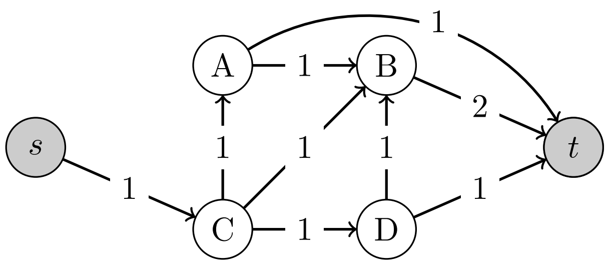
[1] Here we could also start from an initial fractional orientation $\vec{G}_f$. It appears that this provides no advantage.



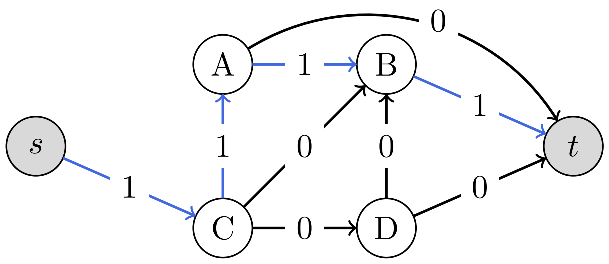
(a)



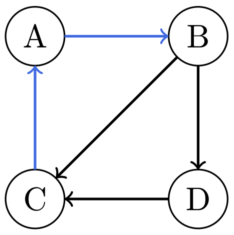
(b)



(c)



(d)



(e)

Figure 4.1: (a) A simple graph with $\lceil d^* \rceil = 2$. (b) A 3-orientation of the graph that serves as an initial orientation here. (c) The re-orientation network for the initial orientation and test parameter $d = 2$. (d) An integral maximum flow that saturates all source arcs, which means that the test for $d = 2$ is successful. (e) A 2-orientation that is obtained by reversing the edges in the initial orientation whose corresponding arcs carry nonzero flow (blue).

A test is successful if all source arcs are saturated by the maximum flow, i.e., every vertex whose indegree was greater than d could lower it to at most d, and the indegrees of the other vertices did not rise to more than d.

While this is quite intuitive, the fact that the test performed in this way correctly answers the question whether a d-orientation exists was proven rigorously and independently by Bezáková [Bez00], Kowalik [Kow06], and Asahiro et al. [Asa+07]. The reader may want to rely on intuition and skip the following two lemmata that use results from the previous chapter for an alternative proof. We first show that a flow in the re-orientation network can be converted into a flow in the bipartite network.

Lemma 4.1.2. *Let $\vec{G}$ be an orientation. Consider the re-orientation network for $\vec{G}$ and a (possibly non-integral) test value $d \geq 0$. Let*

$$R := c(\{s\}, V \cup \{t\}) = \sum_{\substack{v \in V \\ \operatorname{indeg}_{\vec{G}}(v) > d}} (\operatorname{indeg}_{\vec{G}}(v) - d)$$

denote the total indegree to be re-oriented away.

For $0 \leq x \leq R$, a flow f of value $R - x$ in the re-orientation network can be converted into a flow $\tilde{f}$ of value $|E| - x$ in the bipartite network for test value d in linear time. Moreover, if f and d are integral, so is $\tilde{f}$.

Proof. We denote the orientation of each edge in $\vec{G}$ by $f_{uv,u}, f_{uv,v} \in \{0, 1\}$, which sum to one. Let R and x be defined as above.

Consider the (possibly fractional) orientation $\vec{G}'$ that arises when a edges $(u, v) \in \vec{E}$ are re-oriented according to f: Set $f'_{uv,v} = f_{uv,v} - f(v, u)$, and set $f'_{uv,u} = f_{uv,u} + f(v, u)$. Clearly $f'_{uv,v} + f'_{uv,u} = 1$.

Let $\operatorname{in}(v) := \sum_{uv} f_{uv,v}$. Note that if f is integral, so are all the values $f'_{uv,v}$ because we started from a non-fractional orientation.

For a vertex $v \in V$, define the *deficiency*

$$\operatorname{def}(v) := \begin{cases} (\operatorname{indeg}_{\vec{G}}(v) - d) - f(s, v), & \text{if } \operatorname{indeg}_{\vec{G}}(v) > d, \\ 0, & \text{otherwise.} \end{cases} \tag{4.1}$$

Intuitively speaking, this is the amount of indegree that f fails to re-orient at v in order to lower the indegree to d. Note that the deficiencies sum to x. By flow conservation, for a vertex v with $\operatorname{indeg}_{\vec{G}}(v) > d$ (and thus $f(v, t) = 0$) we have

$$\begin{aligned} \operatorname{def}(v) &= \operatorname{indeg}_{\vec{G}}(v) - d - (f(v, V) - f(V, v)) \\ &= \operatorname{indeg}_{\vec{G}'}(v) - d. \end{aligned} \tag{4.2}$$

We now modify f' in order to obtain the desired flow $\tilde{f}$ in the bipartite network. For each $v \in V$ with $\operatorname{indeg}_{\vec{G}}(v) > d$, subtract a total of $\operatorname{def}(v)$

from the values $f'_{uv,v}$ such that they stay nonnegative in an arbitrary fashion, this is feasible because

$$\sum_{uv \in E} f'_{uv,v} = \operatorname{indeg}_{\vec{G}'}(v) \overset{(4.2)}{=} \operatorname{def}(v) + d \tag{4.3}$$
$$\geq \operatorname{def}(v).$$

If f and therefore the orientation f' are integral, we should only subtract zero or one for each edge in order to maintain integer values. The other values $f'_{uv,v}$ are adopted without change. Call the modified values $\tilde{f}_{uv,u}, \tilde{f}_{uv,v}$. Note that $\tilde{f}$ defines a (fractional) orientation only if $\tilde{f}_{uv,v} + \tilde{f}_{uv,u} = 1$ for all $uv \in E$. We show that $(\tilde{f}_u, \tilde{f}_v)$ extends to a feasible flow $\tilde{f}$ with $\tilde{f}(u,v) = \tilde{f}_{uv,v}$ in the bipartite network. We set the flow from the source to a node for edge uv to $\tilde{f}_{uv,v} + \tilde{f}_{uv,u} \leq 1$, this ensures flow conservation in this node and a flow value of $|E| - x$. The flow on each sink arc (v,t) is set to

$$\tilde{f}(v,t) = \sum_{uv \in E} \tilde{f}_{uv,v}.$$

For a vertex v with $\operatorname{indeg}_{\vec{G}}(v) > d$ we have

$$\tilde{f}(v,t) = \sum_{uv \in E} f'_{uv,v} - \operatorname{def}(v) \overset{(4.3)}{=} d.$$

For the other vertices, define the *opportunity*

$$\operatorname{opp}(v) := \begin{cases} (d - \operatorname{indeg}_{\vec{G}}(v)) - f(v,t), & \text{if } d > \operatorname{indeg}_{\vec{G}}(v), \\ 0, & \text{otherwise,} \end{cases}$$

which is, intuitively speaking, the amount of indegree that v could still take. For each $v \in V$ with $d > \operatorname{indeg}_{\vec{G}}(v)$ we have by flow convervation

$$\operatorname{opp}(v) = (d - \operatorname{indeg}_{\vec{G}}(v)) - (f(V,v) - f(v,V)) \tag{4.4}$$

and therefore

$$\begin{aligned} \tilde{f}(v,t) = \sum_{uv \in E} f'_{uv,v} &= \sum_{uv \in E} f_{uv,v} + f(V,v) - f(v,V) \\ &= \operatorname{indeg}_{\vec{G}}(v) + f(V,v) - f(v,V) \\ &\overset{(4.4)}{=} d - \operatorname{opp}(v) \\ &\leq d. \end{aligned}$$

Therefore, the flow $\tilde{f}$ is feasible for test value d. If the flow f is integral, so is $\tilde{f}$. □

It is not surprising that we can perform the conversion in the opposite direction.

Lemma 4.1.3. *A flow (f_u, f_v) of value $|E| - x$ in the bipartite flow network for test value $d \geq 0$ can be converted into a flow f of value $R - x$ in the re-orientation network for test value d in linear time, where R is defined as in Lemma 4.1.2. If (f_u, f_v) and d are integral, so is f.*

The proof is similar to Lemma 4.1.2 and omitted. Although we already know the runtimes in the following theorem for finding $\lceil d^* \rceil$ via Goldberg's method (Theorem 3.4.2), we can now determine an optimal orientation simultaneously. Bezáková [Bez00] and Asahiro et al. [Asa+07] only show the runtime of $\mathcal{O}(|E|^{3/2} \log d^*)$ on the re-orientation network with Dinitz's algorithm.

Theorem 4.1.4. *A smallest maximum indegree orientation can be computed with the re-orientation network in time*

$$\mathcal{O}\left(|E| \min\left(\sqrt{|E|}, |V|^{2/3}\right) \log d^*\right)$$

with Dinitz's algorithm, and in time

$$\tilde{\mathcal{O}}\left(|E|^{10/7} \log d^*\right) = \tilde{\mathcal{O}}\left(|E|^{10/7}\right)$$

with Mądry's algorithm, and in time

$$\tilde{\mathcal{O}}\left(|E|\sqrt{|V|} \log d^*\right) = \tilde{\mathcal{O}}\left(|E|\sqrt{|V|}\right)$$

with the Lee–Sidford algorithm.

Proof. Lemma 4.1.2 and Lemma 4.1.3 show in conjunction with the bipartite network that a maximum flow saturates all source arcs if and only if $d \geq d^*$. Hence, a test for a d-orientation can be made by computing the maximum flow in the re-orientation network for test value d. The runtime analysis is analogous[2] to the proof of Theorem 3.4.2. □

While binary-search-based approaches seem to be the fastest one could come up with, we will see in Chapters 5 and 6 that even the logarithmic factor incurred by the binary search can be successfully attacked for Dinitz's algorithm.

We will next show that if $d \geq d^*$, a maximum flow in the bipartite network can be converted into a maximum flow in Goldberg's network, and vice versa.

Lemma 4.1.5. *A maximum flow in the bipartite network for a test value $d \geq d^*$ can be converted into a maximum flow f in Goldberg's network for test value d in linear time. If f is integral, so is the constructed flow.*

2 In fact, it is slightly easier, because there are no vertices with both a source and a sink arc. Therefore, a variant of Proposition 2.14.3 without Lemma 2.14.2 would suffice.

Proof. Let (f_u, f_v) denote the maximum flow in the bipartite network as before. Define $\mathrm{in}(v) := \sum_{uv} f_{uv,v}$, which is at most d for every $v \in V$, and $\mathrm{out}(v) = \sum_{uv} f_{uv,u}$. Note that since $d \geq d^*$, the flow value is $|E|$ and we have $\mathrm{in}(v) + \mathrm{out}(v) = \deg(v)$. We set the flow in Goldberg's network to be

$$\begin{aligned} f(u,v) &= f_{uv,v}, && uv \in E, \\ f(v,u) &= f_{uv,u}, && uv \in E, \\ f(s,v) &= |E|, && v \in V, \\ f(v,t) &= |E| + 2\,\mathrm{in}(v) - \deg(v), && v \in V. \end{aligned}$$

We first show that the flow is feasible. The sink arc capacity constraints are fulfilled because we have $\mathrm{in}(v) \leq d$:

$$\begin{aligned} f(v,t) &= |E| + 2\,\mathrm{in}(v) - \deg(v) \\ &\leq |E| + 2d - \deg(v) \\ &= c(v,t). \end{aligned}$$

All other capacity constraints are trivially satisfied.

Every vertex v receives $\mathrm{in}(v)$ from its neighbors and sends $\mathrm{out}(v)$ to its neighbors. Furthermore, it receives $|E|$ from the source and sends $|E| + 2\,\mathrm{in}(v) - \deg(v)$ to the sink. Flow conservation is fulfilled in every vertex v since

$$\begin{aligned} |E| + \mathrm{in}(v) &= |E| + \mathrm{in}(v) - (\deg(v) - \mathrm{in}(v) - \mathrm{out}(v)) \\ &= \mathrm{out}(v) + (|E| + 2\,\mathrm{in}(v) - \deg(v)). \end{aligned}$$

Thus the flow f is feasible and its value is $|V||E|$, so it is maximum. Clearly f is integral if (f_u, f_v) is. □

The opposite direction is a little harder to show, because the flow values $f(u,v) + f(v,u)$ in Goldberg's network need not sum to one.

Lemma 4.1.6. *A maximum flow in Goldberg's network for test value $d \geq d^*$ can be converted into a maximum flow in the bipartite network for test value d in linear time.*

Proof. Consider a maximum flow f in Goldberg's network for test value $d \geq d^*$. We will first modify f while maintaining feasibility and maximality. For $uv \in E$, consider the sum $s(uv) = f(u,v) + f(v,u)$. If $s(uv) = 1$, we do not alter the flow values. If $s(uv) < 1$, then we increase both $f(u,v)$ and $f(v,u)$ by $(1 - s(uv))/2$. Clearly, this increases the sum to exactly one and does not affect flow conservation in u and v. It also does not violate capacity constraints because

$$\begin{aligned} \frac{1 - f(u,v) - f(v,u)}{2} &\leq 1 - f(u,v) - f(v,u) \\ &\leq 1 - \max(f(u,v), f(v,u)). \end{aligned}$$

If $s(uv) > 1$, we decrease both $f(u,v)$ and $f(v,u)$ by $(s(uv)-1)/2$. Again, the sum becomes one and flow conservation remains fulfilled. The flow values stay nonnegative because

$$\frac{f(u,v)+f(v,u)-1}{2} \leq f(u,v)+f(v,u)-1 \leq \min(f(u,v),f(v,u)).$$

Let $\mathrm{in}(v) = \sum_{(u,v)} f'(u,v)$ and $\mathrm{out}(v) = \sum_{(u,v)} f'(v,u)$ with $\mathrm{in}(v) + \mathrm{out}(v) = \deg(v)$. We now set $f'_{uv,v} = f'(u,v)$ and $f'_{uv,u} = f'(v,u)$ in the bipartite network, both being in the interval $[0,1]$ and satisfying $f'_{uv,u} + f'_{uv,v} = 1$. It remains to show that $\mathrm{in}(v) \leq d$. The amount of flow that enters a vertex v in f' is $|E| + \mathrm{in}(v)$, the amount that leaves the vertex is at most $\mathrm{out}(v) + |E| + 2d - \deg(v)$. Therefore, for every $v \in V$,

$$\begin{aligned} |E| + \mathrm{in}(v) &\leq \mathrm{out}(v) + |E| + 2d - \deg(v) \\ &= (\deg(v) - \mathrm{in}(v)) + |E| + 2d - \deg(v) \\ &= |E| + 2d - \mathrm{in}(v). \end{aligned}$$

By rearranging and dividing by two, we obtain the desired

$$\sum_{uv\in E} f'_{uv,v} = \mathrm{in}(v) \leq d.$$

□

Unfortunately, extending the proofs to non-maximal flows and to the case $d < d^*$ seems difficult.

The re-orientation network has an advantage over the bipartite network (at least on the surface), as we shall see in the next section.

4.2 KOWALIK'S APPROXIMATION SCHEME

Kowalik [Kow06] uses the re-orientation network to obtain an approximation scheme for the smallest maximum indegree problem. We will use this approximation scheme for new results in Chapter 6 and Chapters 9 and 10.

A rough overview of the algorithm is as follows: Perform the re-orientation algorithm from the previous chapter using Dinitz's algorithm in a binary search with integral test values. However, if only a $(1+\epsilon)$-approximation is required, we can stop Dinitz's algorithm after $\mathcal{O}(\log_{1+\epsilon}|V|)$ phases. The usage of Dinitz's algorithm is essential, as we shall see.

In order to show this, Kowalik proves the following lemma [Kow06], which is a variant of a lemma by Brodal and Fagerberg [BF99, Lemma 2]. However, in its stated form, we found that it is not sufficient for its purposes[3], and this mistake was copied to a generalized version in [Blu16].

3 This was confirmed by Łukasz Kowalik, personal communication, September 2016.

Lemma 4.2.1 ([Kow06, Lemma 2]). *Let $\vec{G}$ be a d-orientation of a graph G, and let $d > d^*$. Then for every vertex v, there is a path $v \leftarrow \cdots \leftarrow u$ of length at most $\log_{d/d^*} |V|$ in $\vec{G}$ from a vertex u whose indegree is smaller than d.*

An *arbitrary* orientation is used in each test of the re-orientation algorithm. However, the lemma only makes a statement about testing for d when a d-orientation is already given, which is pointless.[4] Dropping the requirement of a d-orientation does not render the proof in [Kow06] invalid, only the word 'exactly' must be replaced with 'at least'. The required lemma is as follows.

Lemma 4.2.2. *Let $\vec{G}$ be an arbitrary orientation of a graph G, and let $d > d^*$. Then for every vertex v, there is a path $v \leftarrow \cdots \leftarrow u$ of length at most $\log_{d/d^*} |V|$ in $\vec{G}$ from a vertex u whose indegree is smaller than d.*

We defer the proof to Section 6.1, where we will prove a generalization to fractional orientations. Let us accept Lemma 4.2.2 as given for the moment.

We now review Kowalik's algorithm and its analysis (Algorithm 4.1 on the facing page) in detail. We note that the binary search in the original [Kow06, Algorithm 4.2] is broken. (The reader may want to skip to Theorem 4.2.3 and assume the binary search works correctly.) Kowalik's search maintains a lower bound l and an (initially feasible) upper bound u, the test value is chosen as $t = \lceil (l+u)/2 \rceil$. If the test is successful, the upper bound is set to t. If for example $l = 2, u = 3$, this means that the next test value is three, and since this is feasible, an infinite loop is entered. Thus in the case of success, an update $u \leftarrow u - 1$ should be performed. The minimum feasible value encountered so far can be stored separately. Now, however, we may not stop when $l = u$, as this value may not have been tested, even if we update $l \leftarrow t + 1$ upon every failure. We should only stop once $l > u$. Alternatively, one can use the test value $t = \lfloor (l+u)/2 \rfloor$, keep the upper bound feasible by setting $u = t$ upon success, and update $l \leftarrow t + 1$ in the case of failure. We will use this latter binary search in Section 7.3.

Theorem 4.2.3 ([Kow06]). *Given $\epsilon > 0$, an orientation with a maximum indegree of at most $\lceil (1+\epsilon) d^* \rceil$ can be found in time*

$$\mathcal{O}(|E| \log |V| \epsilon^{-1} \log d^*).$$

Proof. Obtain a 2-approximating upper bound by exponential search, this needs $\mathcal{O}(\log d^*)$ tests. Alternatively, we can use a linear-time 2-approximation algorithm (see Chapter 7).

4 One might wonder if using a good initial orientation might yield some advantage. Although using a 2-approximating orientation in an algorithm in Section 3.6 was useful, we do not find this to be the case for the re-orientation algorithm.

Algorithm 4.1: Kowalik's approximation scheme (with a corrected binary search).

```
Input: A simple graph $G = (V, E)$, an arbitrary orientation $\vec{G}$ of
       $G$, and a parameter $\epsilon > 0$.
Output: An integer $d$ such that $\lceil d^* \rceil \leq d \leq \lceil (1+\epsilon)d^* \rceil$.
function test($t$,$k$):
    $N_t \leftarrow$ the re-orientation network for parameter $t$
    /* Compute integral flow                                   */
    Run Dinitz's algorithm on $N_t$ for $k$ phases
    if all source arcs in $N_t$ are saturated then
        return true
    return false
function approxKowalik($\epsilon$):
    $l = 0$
    $u = 1$
    $k = 2 + \log_{1+\epsilon} |V|$                  /* Pathlength bound */
    /* Compute 2-approximation by exponential search         */
    while test($u$,$k$) = false do
        $l = u + 1$          /* asymptotically not necessary */
        $u \leftarrow 2u$
    $d = u$                /* minimum feasible found so far */
    while $l < u$ do
        $t = \left\lfloor \frac{u+l}{2} \right\rfloor$
        if test($t$,$k$) = true then
            $d = t$
            $u \leftarrow t$
        else
            $l \leftarrow t + 1$
    return $d$
    /* The algorithm can be modified to return a
       $d$-orientation                                       */
```

By Lemma 2.15.1, a blocking flow can be computed in time $\mathcal{O}(|E|)$. By a Taylor expansion of $\log_{1+\epsilon}(x)$, the runtime of Algorithm 4.1 is bounded by $\mathcal{O}(|E| \log |V| \epsilon^{-1} \log d^*)$. We now prove its correctness.

If $d < d^*$, the test will always (and correctly) return a failure.

If $d^* \leq d < (1+\epsilon)d^*$, the test may return a failure although d is feasible because the flow algorithm is terminated early.

If $d \geq (1+\epsilon)d^*$, then $d/d^* \geq 1+\epsilon$. Every residual network for an integral flow corresponds to an orientation. By Lemma 4.2.2, if there is an augmenting path in the residual network, then there is one of length at most $2 + \log_{d/d^*} |V|$. By Lemma 2.13.1, the length of the shortest augmenting path increases with every blocking flow phase. Thus after $2 + \log_{1+\epsilon} |V|$ phases, there can be no augmenting path, and by Lemma 2.12.8 the flow is maximum.

If a test is successful, the corresponding d-orientation is obtained by reversing the edges whose corresponding arcs carry a flow of one. We can store the d-orientation for the smallest d that is reported feasible in the binary search, and by the above analysis this d satisfies $\lceil d^* \rceil \leq d \leq \lceil (1+\epsilon)d^* \rceil$. □

Note that Kowalik's scheme never stops the binary search early as it does not detect when the error ϵ is undershot; it solely relies on the stopping criterion derived from Lemma 4.2.2. Thus, if this criterion is never met during the execution, Kowalik's scheme is in fact the re-orientation algorithm and outputs the optimum solution. We will, however, additionally stop the binary search for fixed ϵ in Section 7.3.

While the output d approximates d^*, we do not know how to obtain a subgraph of density at least $d^*/(1+\epsilon)$ within the same runtime: Goldberg's network can be used to find such a subgraph with $d/(1+\epsilon)$ as the test value and a single maximum flow computation. However, in the re-orientation network, terminating after $2 + \log_{1+\epsilon} |V|$ phases may prevent us from finding a maximum flow for test values below $(1+\epsilon)d^*$. Even if we could convert a *maximum* flow in the re-orientation network for $d < d^*$ into a maximum flow in Goldberg's network for the same test value (which we did not achieve in the previous section), this would not help in finding a suitable subgraph.

However, the approximation scheme will help us in determining $\lceil d^* \rceil$ faster in Chapter 6, which eliminates the binary search for finding an 'almost-densest subgraph' of density greater $\lceil d^* \rceil - 1$.

4.3 APPLICATIONS AND GENERALIZATIONS

If a graph is given as an adjacency matrix, the question whether $(u, v) \in E$ can be answered in constant time. However, the representation needs $\Theta(|V|^2)$ bits. If we use adjacency lists, we can answer in $\mathcal{O}(|V|)$ time by scanning u's adjacency list, and $\mathcal{O}(\log |V|)$ time with a binary search if the adjacency lists are sorted. As we shall soon see, all adjacency lists can be sorted in total linear time.

Forest partitions were used to represent graphs by Kannan et al. [KNR92] and Arikati et al. [AMZ97]. Chrobak and Eppstein [CE91] and Aichholzer et al. [AAR95] apply the integral orientation problem[5] to the adjacency-list representation of a graph. These techniques can speed up some graph algorithms.

Theorem 4.3.1 (Essentially [CE91; AAR95; AMZ97]). *Let $G = (V, E)$ be a simple graph. Given a $\lceil d^* \rceil$-orientation $\vec{G}$ of G, a data structure that uses $\mathcal{O}(|E|)$ space and answers edge queries correctly in $\mathcal{O}(\log d^*)$ time can be constructed in $\mathcal{O}(|E|)$ time.*

Proof. We first sort all adjacency lists of the simple graph G in $\mathcal{O}(|V| + |E|)$. (We assume that $V = \{1, \dots, |V|\}$.) Create $|V|$ linked lists (buckets) and go through the adjacency lists in ascending order $u = 1, \dots, |V|$. For every v with (u, v) in u's list, add u to bucket v. After this has been done for all lists, bucket v contains a sorted adjacency list of vertex v: Clearly, every neighbor of v is present, and the vertices were inserted in ascending order.

Let $\vec{G}$ be a $\lceil d^* \rceil$-orientation of G. We create a new adjacency list representation of G: An edge uv is stored in the adjacency list of v if $u \to v$ in $\vec{G}$. Then, every adjacency list has length at most $\lceil d^* \rceil$, which is optimal. The E-membership of $uv \in V \times V$ can be queried in time $\mathcal{O}(\log d^*)$ by performing a binary search on u's list and, if it is not present there, another search on v's list. □

Chrobak and Eppstein [CE91] show that a planar graph can be 3-oriented in time $\mathcal{O}(|V|)$. By Theorem 4.3.1, edge queries can then be performed in constant time.

For general graphs, Eppstein [Epp94] and Aichholzer et al. [AAR95] use a 2-approximation algorithm (Chapter 7) to obtain an orientation in linear time that guarantees the same asymptotic bounds as Theorem 4.3.1. Brodal and Fagerberg [BF99] consider a dynamic data structure for supporting adjacency queries that uses orientations of bounded indegree. They achieve $\mathcal{O}(d^*)$ time for an adjacency query. Edge insertion is possible in constant amortized time, edge deletion is possible in $\mathcal{O}(d^* + \log |V|)$ amortized time. A variant of this data structure supports queries in $\mathcal{O}(\log d^*)$ time, insertions in $\mathcal{O}(\log d^*)$ amortized time, and deletions in $\mathcal{O}(\log |V|)$ amortized time. We will apply this data structure in our arboricity approximation scheme in Section 10.3.

For further applications of orientations and the equivalent pseudoforest partitions, see [Wes88; GW92; Kow06]. We can generalize the orientation problem to edge-weighted graphs $G = (V, E, w)$, and ask to minimize the maximum weighted indegree. We list two results showing that it is unlikely to be solvable in polynomial time as well.

5 We will see the equivalence to pseudoforest partitions and their relationship with forest partitions in Chapter 8.

Theorem 4.3.2 ([Asa+07]). *Let $G = (V, E, w)$ be a simple graph with edge-weights $w : E \to \mathbb{N}$. The decision problem whether an orientation of G exists whose maximum weighted indegree is at most k is NP-complete, even for planar bipartite graphs.*

The proof uses a reduction from the partition problem.

Theorem 4.3.3 ([Asa+07]). *There is a $3/2$-approximation algorithm for the smallest maximum weighted indegree problem if the edge weights are from the set $\{1, 2\}$, and this factor is optimal unless P=NP.*

The proof of the latter uses a reduction from a variant of the 3-SAT problem.

5 BALANCED BINARY SEARCH

Knuth points out that while the first binary search was published in 1946, the first published binary search without bugs did not appear until 1962.

— Jon Bentley, Programming Pearls (1986)

In the following we will show that the runtime claim of $\mathcal{O}(|E|^{3/2}\log d^*)$ by Aichholzer et al. [AAR95] for the bipartite network is correct for Dinitz's algorithm. Furthermore, we shall recast the Gabow–Westermann algorithm, which is a matroid partitioning algorithm (see Chapter 8) with runtime

$$\mathcal{O}\left(|E|\min\left(\sqrt{|E|\log d^*}, (|V|\log d^*)^{2/3}\right)\right)$$

in the language of flows entirely.

We first modify the bipartite flow network as follows. The only arcs whose capacity exceeds one are those from the vertex nodes to the sink, which have the test value d as their capacity. However, as a node for vertex $v \in V$ has $\deg_G(v)$ ingoing arcs, we can safely set $c(v,t) = \min(d, \deg_G(v))$. By the degree sum formula, we now have an AUC network with total sink and source arc capacities bounded by $3|E|$.

Theorem 5.0.1 (sketch by [AAR95]). *Let $G = (V, E)$ be a simple graph. A smallest maximum indegree orientation can be determined with Dinitz's algorithm on the bipartite network in time $\mathcal{O}(|E|^{3/2}\log d^*)$.*

Proof. While the number of nodes is $|V| + |E|$, Theorem 2.15.4 nonetheless yields a runtime of $\mathcal{O}(|E|^{3/2})$ for testing a d-orientation when the modified bipartite network is used. Note that the runtime is in fact the same for the original bipartite network.

We perform the binary search for integral test values in the usual way to obtain a total runtime of $\mathcal{O}(|E|^{3/2}\log d^*)$. □

By stopping the flow algorithm early, we can find an optimal orientation for a restricted number of edges.

Lemma 5.0.2 (Re-phrased from [GW92]). *Given $\Delta \in \mathbb{N}$, it is possible to determine a d-orientation for a subgraph of G with at least $|E| - \Delta$ edges satisfying $d \leq \lceil d^*(G) \rceil$ in time $\mathcal{O}(|E|^2/\Delta \log d^*)$.*

Proof. We use the modified bipartite flow network again. Let M denote the maximum flow value and use Dinitz's algorithm. Consider the phase in which the flow value F reaches the value $M - \Delta$. When this phase begins, we have $F < M - \Delta$. The residual network $\tilde{N}$ is an AUC-2 G-network with total source and sink arc capacities $C_{s,t}(\tilde{N}) \leq c|E|$ for a $c > 0$. By Lemma 2.12.9, its maximum flow is

$$\tilde{M} = M - F > M - (M - \Delta) = \Delta.$$

Since the flow in the residual network is initially zero, by Lemma 2.15.3, the length of the shortest augmenting path satisfies

$$\tilde{l} \leq \frac{2|E|}{\tilde{M}} + 2 < \frac{2|E|}{\Delta} + 2.$$

The number of phases until this point is thus at most $\frac{2|E|}{\Delta} + 2$, and the runtime is $\mathcal{O}(|E|^2/\Delta)$ by Proposition 2.15.1.

To obtain the algorithm, perform the usual binary search, and at each test, run $2 + 2|E|/\Delta$ phases of Dinitz's algorithm. Find the smallest integral test value d such that $F \geq |E| - \Delta$. For every test value $d \geq \lceil d^*(G) \rceil$, we have $M = |E|$, hence a $d \leq \lceil d^*(G) \rceil$ will be returned. The flow in this test can be interpreted as an orientation on a subgraph with at least $|E| - \Delta$ edges: discard the edges corresponding to edge nodes whose incoming arcs are not saturated. The remaining edge nodes, along with their outgoing arcs, determine the partial orientation. $\square$

Now we show that a partial orientation can be easily updated upon an edge insertion.

Proposition 5.0.3. *Let $G = (V, E)$ and $(V_H, E_H) = H \subseteq G$, and let a $\lceil d^*(G) \rceil$-orientation $\vec{H}$ of H be given. An edge $e \in E \setminus E_H$ can be inserted into $\vec{H}$ such that it stays a $\lceil d^*(G) \rceil$-orientation in time $\mathcal{O}(|E|)$.*

Proof. Note that the maximum indegree in $\vec{H}$ may be smaller than $\lceil d^*(G) \rceil$. Insert the edge $e = uv$ into H and as $u \to v$ into $\vec{H}$.

If the maximum indegree does not increase thereby, the claim holds. Otherwise, v is the only vertex with the (new) maximum indegree. We perform a single step of the path reversal algorithm in time $\mathcal{O}(|E|)$ (see Section 3.6 from Theorem 3.6.8 onward). If the algorithm successfully lowers the indegree by one, the modified orientation is optimal again. Otherwise, there is no orientation with smaller indegree, hence it already is optimal. $\square$

Now we balance the runtimes of the two algorithms, which gives rise to the name *balanced binary search*. This technique was first described for the bottleneck maximum cardinality matching problem by Gabow and Tarjan [GT88a] (see Section 6.2). To the best of our knowledge, this and the matroid partitioning problem are the only problems this technique has been applied to.

We will set Δ depending on d^*. As the latter is unknown, we will use an approximation instead: For all claims, we initially compute a

2-approximation d_G in linear time with the greedy algorithm in Chapter 7. The values d_G and $d_G/2$ will be used to set parameters. However, we will suggestively call these values d^* as this does not change the runtime estimates asymptotically and simplifies the presentation. We use $\simeq$ to signify the constant factors when setting parameters. We further assume that numeric computations are performed to arbitrary precision, and $d^* > 1$.

Theorem 5.0.4 (Re-phrased from [GW92]). *An optimal orientation can be found in time $\mathcal{O}(|E|^{3/2}\sqrt{\log d^*})$.*

Proof. Set

$$\Delta \simeq \left\lceil \sqrt{|E| \log d^*} \right\rceil .$$

Use Lemma 5.0.2 to obtain a subgraph H with at least $|E| - \Delta$ edges and a $\lceil d^*(G) \rceil$-orientation $\vec{H}$ of it in time $\mathcal{O}(|E|^{3/2}\sqrt{\log d^*})$.

Insert each of the at most Δ remaining edges optimally into $\vec{H}$ in linear time according to Proposition 5.0.3. In total, this takes $\mathcal{O}(|E|^{3/2}\sqrt{\log d^*})$ time. □

Gabow and Westermann achieve the runtime $\mathcal{O}(|E|(|V| \log d^*)^{2/3})$ on the same bipartite network, although they do not call it a flow network. However, they only sketch the argument and require reversing the bipartite network for their analysis: They claim that starting from the edges nodes uses too much time. In the following, we will use the re-orientation network instead to achieve the desired runtime. This is maybe a less intuitive approach – in contrast to the bipartite network, there are no unassigned edges! However, as we saw in the previous chapter, the two networks are equivalent.

Lemma 5.0.5 (Re-phrased from [GW92]). *Given $\Delta \in \mathbb{N}$, it is possible to determine a d-orientation for a subgraph of $G = (V, E)$ with at least $|E| - \Delta$ edges satisfying $d \leq \lceil d^*(G) \rceil$ in time $\mathcal{O}(|E||V|/\sqrt{\Delta} \log d^*)$.*

Proof. We use the re-orientation network with an arbitrary initial (non-fractional) orientation $\vec{G}$, and Dinitz's algorithm. Let

$$R = c(\{s\}, V \cup \{t\}) = \sum_{\substack{v \in V \\ \text{indeg}(v) > d}} (\text{indeg}_{\vec{G}}(v) - d)$$

denote the total number of edges to be oriented away from vertices, and let M denote the maximum flow value.

Consider the phase in which the flow value F reaches the value $M - \Delta$. When this phase begins, we have $F < M - \Delta$. The residual network $\tilde{N}$ is an AUC-2 G-network with total source and sink arc capacities $C_{s,t}(\tilde{N}) \leq c|E|$ for some $c > 0$. By Lemma 2.12.9, its maximum flow is

$$\tilde{M} = M - F > M - (M - \Delta) = \Delta.$$

Since the flow in the residual network is initially zero, by Lemma 2.15.5, the length of the shortest augmenting path satisfies

$$\tilde{l} \leq \frac{(2\sqrt{2})|V|}{\sqrt{\tilde{M}}} + 1 < \frac{(2\sqrt{2})|V|}{\sqrt{\Delta}} + 1.$$

This is an upper bound on the number of phases until this point, and the runtime is $\mathcal{O}(|E||V|/\sqrt{\Delta})$ by Proposition 2.15.1.

To obtain the algorithm, perform the usual binary search, and at each test, run $1 + 2\sqrt{2}|V|/\sqrt{\Delta}$ phases of Dinitz's algorithm. Find the smallest test value d such that $F \geq R - \Delta$. For every test value $d \geq \lceil d^*(G) \rceil$ we have $M = R$, hence a $d \leq \lceil d^* \rceil$ will be found.

In order to obtain the subgraph, look at the vertices whose source arcs are not saturated (if any): Such a vertex v still has, after re-orienting according to the integral flow, a deficiency $\operatorname{def}(v) := c(s,v) - f(s,v)$ of arcs that remain to be re-oriented with respect to the indegree goal d. We proceed similarly to Lemma 4.1.3: Arbitrarily pick $\operatorname{def}(v)$ edges from the new orientation that are oriented towards v, and remove them from the graph. If this is done for every $v \in V$, we are left with a d-orientation of a subgraph that has at most Δ edges less than the original graph. □

We can now again perform a balanced binary search.

Theorem 5.0.6 (Re-phrased from [GW92])**.** *An optimal orientation can be found in time $\mathcal{O}(|E|(|V|\log d^*)^{2/3})$.*

Proof. Compute a 2-approximation to estimate $\lceil d^* \rceil$. Set

$$\Delta \simeq \left\lceil (|V| \log d^*)^{2/3} \right\rceil .$$

Use Lemma 5.0.5 to obtain a subgraph H with at least $|E| - \Delta$ edges and a $\lceil d^* \rceil$-orientation $\vec{H}$ of it in time $\mathcal{O}(|E|(|V|\log d^*)^{2/3})$.

Insert each of the at most Δ remaining edges into $\vec{H}$ in linear time according to Proposition 5.0.3. This takes time $\mathcal{O}(|E|(|V|\log d^*)^{2/3})$ in total.

□

Using Theorems 5.0.4 and 5.0.6, we can eliminate the binary search in Goldberg's method for integral test values and compute an 'almost densest' subgraph with a single maximum flow computation.

Corollary 5.0.7. *A subgraph of density greater than $\lceil d^* \rceil - 1$ can be found in time*

$$\mathcal{O}\left(|E| \min\left(\sqrt{|E|\log d^*}, (|V|\log d^*)^{2/3}\right)\right).$$

6 ACCELERATED BINARY SEARCH

I was shocked to learn that the binary search program that Bentley proved correct and subsequently tested in Chapter 5 of Programming Pearls *contains a bug. [...] [L]et me tell you how I discovered the bug: The version of binary search that I wrote for the JDK contained the same bug. It was reported to Sun recently when it broke someone's program, after lying in wait for nine years or so.*

— Joshua Bloch, Google Research Blog (2006)

In this chapter, we will improve upon the runtime of the Gabow–Westermann algorithm unconditionally for the $\sqrt{|E|}$-branch, and more strongly if certain asymptotic conditions are met for d^*. Analogous conditional bounds can be proved for the $|V|^{2/3}$-branch with our technique. The only flow algorithm we employ is the one of Dinitz. Conditional bounds can also be proved when Mądry's algorithm or the Lee–Sidford algorithm are employed.

We introduce a new technique that we call *accelerated binary search*: A $(1+\epsilon)$-approximation is obtained with an approximation scheme, which is then used to bound the search interval more strongly for the exact algorithm with Dinitz's algorithm on the re-orientation network. With this technique, we can obtain an algorithm with runtime $\mathcal{O}(|E|^{3/2} \log\log d^*)$.

We can even combine this with the balanced binary search: We obtain a smaller search interval and then use the Gabow–Westermann algorithm. This allows for a runtime of $\mathcal{O}(|E|^{3/2}\sqrt{\log\log d^*})$. We call this *accelerated balanced binary search*.

In certain cases, it is worth performing an *iteratively accelerated binary search*. As Kowalik's approximation scheme uses a binary search itself, we can repeatedly approximate with ever smaller values of ϵ to make the search interval smaller and smaller. The use of the balanced binary search for the final interval does not yield any asymptotic benefit, however.

Theorem 6.0.1. *A smallest maximum indegree orientation can be computed in the runtime bounds (I)-(III) stated in Table 6.1 on the next page with Dinitz's algorithm. Moreover, a subgraph of density greater $\lceil d^* \rceil - 1$ can be found within the same runtimes.*

Proof. We set ϵ using a constant-factor approximation. As in the previous chapter, this will be indicated by the symbol $\simeq$.

Consider claim (I) in Table 6.1. Check if $d^* \leq |V|^{0.49}$. If this is the case, see the proof of claim (III). Otherwise, set $\epsilon \simeq (\log(d^*)\log|V|)/d^*$

Table 6.1: New runtime bounds for the orientation problem, depending on d^*. Here, $\log^*$ denotes the iterated logarithm to the base 2.

Bound for d^*	Runtime	
$\Omega\left(\sqrt{\lvert E\rvert}\right)$	$\mathcal{O}\left(\lvert E\rvert^{3/2}\right)$	Chapter 11
–	$\mathcal{O}\left(\lvert E\rvert^{3/2}\sqrt{\log\log d^*}\right)$	(I)
$\mathcal{O}\left(\frac{\sqrt{\lvert E\rvert}}{\log\lvert V\rvert}\right)$	$\mathcal{O}\left(\lvert E\rvert^{3/2}\log^* d^*\right)$	(II)
$\mathcal{O}\left(\frac{\sqrt{\lvert E\rvert}}{\log^2\lvert V\rvert}\right)$	$\mathcal{O}\left(\lvert E\rvert^{3/2}\right)$	(III)

and thus the first phase runs in $\mathcal{O}(|E|d^*)$ time. By Proposition 3.2.2, $d^* \in \mathcal{O}(\sqrt{|E|})$ and hence this runtime can be bounded as $\mathcal{O}(|E|^{3/2})$.

A binary search on the shrunk search interval now needs

$$\mathcal{O}(\log(\epsilon d^*)) \subseteq \mathcal{O}(\log\log d^*)$$

tests by Lemma 3.4.1. The re-orientation algorithm (or Goldberg's or the bipartite network algorithm) can thus perform the second phase in time $\mathcal{O}(|E|^{3/2}\log\log d^*)$. The Gabow–Westermann algorithm can perform the second phase in $\mathcal{O}(|E|^{3/2}\sqrt{\log\log d^*})$ time by changing the parameter for the balanced binary search to $\Delta \simeq \sqrt{|E|\log\log d^*}$ in the proof of Theorem 5.0.4.

For claim (II), assume that $d^* \in \mathcal{O}(\sqrt{|E|}/\log|V|)$. We run the approximation scheme in $i = 1, \ldots, \log^* d^* - 1$ phases[1], each time on the search interval left after the previous phase, with parameters

$$\epsilon_1 \simeq \frac{\log d^*}{d^*}, \epsilon_2 \simeq \frac{\log\log d^*}{d^*}, \epsilon_3 \simeq \frac{\log\log\log d^*}{d^*}, \ldots$$

Recall the functional iteration defined in Section 2.3. We prove inductively that for the interval I_i leftover after phase i, we have $|I_i| \in \mathcal{O}(\log^{(i)} d^*)$ and thus phase i runs in $\mathcal{O}(|E|^{3/2})$ time. The induction basis holds as for the initial search interval I_0, we have $|I_0| \in \mathcal{O}(d^*)$ with the 2-approximation.

Let the induction hypothesis hold for $i-1$. In phase i, we perform Kowalik's scheme on I_{i-1} with ϵ_i in time

$$\mathcal{O}\left(\frac{|E|\log|V|\,d^*}{\log^{(i)} d^*}\log\log^{(i-1)} d^*\right).$$

Thus the phase runs in $\mathcal{O}(|E|^{3/2})$ time and leaves an interval of size $|I_i| \in \mathcal{O}(\log|I_{i-1}|) = \mathcal{O}(\log^{(i)} d^*)$ by Lemma 3.4.1. However, one might fear that the hidden constants accumulate during the iterated

1 Recall that $\log^*$ denotes the iterated logarithm to the base two.

process. In the proof of Lemma 3.4.1, we saw that a multiplicative constant of two and an additive constant of two may be introduced when shrinking the interval. We can divide ϵ_i by a constant $c > 2$ to avoid accumulating ever larger constants.

After $\log^* d^* - 1$ phases, we have shrunk the search interval to a size of $\mathcal{O}(\log^{(\log^* d^*)} d^*) = \mathcal{O}(1)$. The final phase consists of a constant number of tests and runs in $\mathcal{O}(|E|^{3/2})$ time with, e.g., the re-orientation algorithm. Thus claim (II) is proven.

For claim (III), assume that $d^* \in \mathcal{O}(\sqrt{|E|}/(\log |V|)^2)$. Set $\epsilon \simeq 1/d^*$. The first phase clearly runs in time $\mathcal{O}(|E|^{3/2})$. As we get a d-orientation with $\lceil d^* \rceil \leq d \leq \lceil d^* + 1 \rceil = \lceil d^* \rceil + 1$, the second phase consists of a single re-orientation test with Dinitz's algorithm, which takes $\mathcal{O}(|E|^{3/2})$.

A subgraph of density greater $\lceil d^* \rceil - 1$ can be found with a single maximum flow computation on Goldberg's network for test parameter $\lceil d^* \rceil - 1$ in $\mathcal{O}(|E|^{3/2})$ time, see Subsection 3.3.3. □

Similarly, we can speed up the binary search for the flow algorithms of Mądry and Lee and Sidford when d^* is appropriately bounded. However, we are not able to obtain a new unconditional bound as in the case of Dinitz's algorithm.

We can also obtain a near-exact algorithm for the orientation problem whose runtime scales in d^* using the iterated accelerated binary search. The proof is analogous.

Theorem 6.0.2. *A $(\lceil d^* \rceil + 1)$-orientation can be computed within a runtime of $\mathcal{O}(|E| \log |V| \, d^* \log^* d^*)$.*

We conclude this section by asking whether the graph compression technique of Feder and Motwani, which they applied to the bipartite matching problem with Dinitz's algorithm [FM95] (see also Section 6.2), can be used for our purposes.

6.1 FRACTIONAL ORIENTATIONS

We now give a generalization of Lemma 4.2.2 to fractional orientations.

Lemma 6.1.1. *Consider an arbitrary fractional orientation $\vec{G}_f$ of a graph G and $d > d^*(G)$. Then for every vertex v, there is a path $v \leftarrow_f \cdots \leftarrow_f u$ of length at most $\log_{d/d^*} |V|$ in $\vec{G}_f$ from a vertex u whose indegree is smaller than d, where $y \leftarrow_f x$ if $f_{xy,y} > 0$ in $\vec{G}_f$.*

The condition $f_{xy,y} > 0$ is important because in this case, we can send a flow value greater zero from y to x when re-orienting with augmenting-path algorithms.

Proof. Let v be an arbitrary vertex. Let k denote the minimum unit distance in $\vec{G}_f$ with respect to $\leftarrow_f$ as defined above from v to a vertex whose fractional indegree is smaller than d.

Let $|V_i|$ denote the set of vertices which are at distance i at most from v. We show by induction that $|V_i| \geq (\frac{d}{d^*})^i$ for $i = 0, \ldots, k$. The claim holds for $i = 0$. Assume the induction hypothesis holds for some $i < k$. Let E_{i+1} denote the set of edges where both ends are in V_{i+1}.

Every vertex in V_i has a fractional indegree of at least d (otherwise $i \geq k$, which contradicts the assumption). Thus

$$\begin{aligned}
|E_{i+1}| &= \sum_{uw \in E_{i+1}} (f_{uw,u} + f_{uw,w}) \\
&\geq \sum_{u \in V_i} \sum_{\substack{uw \in E \\ f_{uw,u} > 0}} f_{uw,u} = \sum_{u \in V_i} \sum_{uw \in E} f_{uw,u} \geq d|V_i|.
\end{aligned}$$

Since $|E_{i+1}|/|V_{i+1}| \leq d^*$, we obtain $|V_{i+1}| \geq \frac{d}{d^*}|V_i|$. By applying the induction hypothesis, the claim is shown for all $i = 0, \ldots, k$.

Because $|V_k| \leq |V|$, we have $(\frac{d}{d^*})^k \leq |V|$, which concludes the proof. □

Unfortunately, runtime analyses of flow algorithms such as Dinitz's often require integer capacities. For rational test values d for d^*, we can scale the capacities to be integers and apply an algorithm tailored for integral capacities. The Goldberg–Rao method (Section 2.12.4) generalizes ideas of Dinitz's algorithm, and Lemma 6.1.1 might be applied similarly. However, we do not think that this application improves the runtime asymptotically because Goldberg's binary search for rational test values (see Lemma 3.3.3 and thereafter) stops once the search interval size is less than $1/(|V|(|V|-1))$, i.e., there are $\Omega(|V|^2)$ potential test values between two integers. One would have to set a much smaller ϵ in order to accelerate the binary search considerably. Even $\Omega(|V|)$ tests between two integers would be prohibitively large.

However, the stopping criterion of Goldberg's binary search is rather conservative. One may wonder whether the following three points allow for an improvement.

1. The densest subgraph is always an induced subgraph. Thus, several fractions are not candidates, for example $2/3$ in the complete graph K_3.

2. We can restrict the discussion to connected subgraphs because the densest subgraph is attained on a connected subgraph (Proposition 3.1.4).

3. Several densities may coincide (e.g., $12/6 = 10/5$ in Figure 3.1 on page 50).

Let us consider a star of n vertices. The set of all densities of induced connected subgraphs is $\{0/1, 1/2, 2/3, \ldots, (n-1)/n\} \subseteq [0, 1)$ with cardinality n. On the other hand, the complete graph K_n on n vertices has the connected induced subgraphs $K_1, \ldots, K_n$ (all having different densities). Since $d^* = (n-1)/2$, this is not $\Omega(n)$ values between any two integers. (If we drop the connectivity requirement, we can combine several vertex-disjoint subgraphs to obtain more distinct densities.)

We note that coprimality among pairs of integers is not rare, and hence the third bullet point may have an insignificant effect. The following is a well-known result from number theory.

Lemma 6.1.2 (e.g., [HW08]). *Let $n \in \mathbb{N}$, and let integers $1 \leq x, y \leq n$ be chosen uniformly at random, independently of each other. Then*

$$\Pr[x \text{ and } y \text{ are coprime}] \overset{n\to\infty}{\longrightarrow} \frac{6}{\pi^2} \approx 0.608.$$

6.2 BOTTLENECK MAXIMUM CARDINALITY MATCHING

In this section, we give a short note on the bottleneck maximum cardinality matching. Gabow and Tarjan [GT88a] applied a balanced binary search to this problem for a runtime of $\mathcal{O}(|E|\sqrt{|V|\log|V|})$. Can accelerated binary search be applied here as well?

A maximum cardinality matching can be computed in $\mathcal{O}(|E|\sqrt{|V|})$ time [MV80; GT91]. This was the best runtime available to Gabow and Tarjan in 1988. It was first achieved for the case of bipartite graphs by Hopcroft and Karp [HK73] (see also [Kar73; ET75]). Later, the algorithm of Alt et al. [Alt+91] achieved $\mathcal{O}(|V|^{3/2}\sqrt{|E|/\log|V|})$ in the bipartite case, which is superior if $|E| \in \Omega(|V|^2/\log|V|)$.

Today, the best known runtime in general is $\mathcal{O}(|E|\sqrt{|V|}\frac{\log(|V|^2/|E|)}{\log|V|})$ [GK04] (after [FM95; GK97] for the bipartite case). Mądry's algorithm achieves a better bound of $\tilde{\mathcal{O}}(|E|^{10/7})$ in sparse bipartite graphs.[2]

In the bottleneck maximum cardinality matching problem, each edge $e \in E$ has a weight $w(e)$. The goal is to find a maximum cardinality matching where the maximum edge weight is minimized. Gabow and Tarjan [GT88a] first show the obvious way of computing it: One can order the edges as $e_1, \dots, e_{|E|}$ with

$$w(e_1) \leq w(e_2) \leq \cdots \leq w(e_{|E|})$$

in time $\mathcal{O}(|E|\log|V|)$. Let $E_i = \{e_j \mid j \leq i\}$. First a maximum cardinality matching M^* on (V, E) is computed in time $\mathcal{O}(|E|\sqrt{|V|})$ so we can test against its size $|M^*|$. For $i \in \{1, \dots, |E|\}$ we can determine a maximum cardinality matching of (V, E_i) in time $\mathcal{O}(|E|\sqrt{|V|})$. If it has size $|M^*|$, it is a candidate solution, otherwise it cannot be a bottleneck maximum cardinality matching. By performing a binary search for the smallest index i where (V, E_i) has a matching of size $|M^*|$, we can determine a bottleneck maximum cardinality matching.

2 There are also algorithms in both settings that use fast matrix multiplication and machine word operations, which we do consider here. For an overview see [DP14].

Gabow and Tarjan were able to successfully attack the logarithmic factor with their balanced binary search technique.

Theorem 6.2.1 ([GT88a])**.** *A bottleneck maximum cardinality matching can be determined in time $\mathcal{O}(|E|\sqrt{|V|\log|V|})$.*

It is a standard exercise that bipartite matching reduces to solving a maximum flow problem in a unit-capacity network. In fact, the bipartite orientation network for test value $d = 1$ from Section 3.6 is an instance of this network. Theorem 6.2.1 can be proved for bipartite graphs similarly to the analysis in Chapter 5. We sketch the general case.

Proof sketch of Theorem 6.2.1. Let M^* denote a maximum cardinality matching in G. Perform a binary search for the minimum feasible index i such that (V, E_i) has a matching of size $|M^*| - |V|/\Delta$. This can be done in total time $\mathcal{O}(\Delta|E|\log|V|)$. Augment the matching found for (V, E_i) for the minimum feasible index i by performing at most $|V|/\Delta$ augmentations in (V, E_i). Every time no augmentation is possible, increase i. This second phase takes $\mathcal{O}(|E||V|/\Delta)$. The claim follows for $\Delta = \sqrt{|V|/\log|V|}$. □

Unfortunately, it appears that shrinking the search interval is not possible, as it is an interval of edge weights in ascending order (corresponding to the subsets E_i). The maximum matching size can be the same for an interval of size $\Omega(|V|)$.

Still, conditional improvements are possible. As sketched in the above proof, a $(1-\epsilon)$-approximation to the maximum cardinality matching problem can be obtained in $\mathcal{O}(|E|\epsilon^{-1})$ time, as noted in [DP14] (see [MV80; GT91] and [HK73] for bipartite graphs). If we have $|M^*| \in \mathcal{O}(|V|/\log|V|)$, then we can first compute a 1/2-approximation in linear time and therewith set $\epsilon \simeq \sqrt{|V|}/|M^*|$. Performing the binary search, we find the smallest i such that a matching of size at least $(1-\epsilon)|M^*| = |M^*| - \mathcal{O}(\sqrt{|V|})$ exists in (V, E_i). This takes total time $\mathcal{O}(|E||M^*|/\sqrt{|V|}\log|V|)$, which by our assumption is bounded by $\mathcal{O}(|E|\sqrt{|V|})$. Performing $\mathcal{O}(\sqrt{|V|})$ augmentations is possible in time $\mathcal{O}(|E|\sqrt{|V|})$ as in the previous proof sketch.

Unfortunately, the size $|M|$ of a maximum cardinality matching can be as large as $\lfloor |V|/2 \rfloor$. We note that good lower bounds on the size of matchings are known for several graph classes [Bie+04]: There is a matching of size at least $(|V|+4)/3$ in every 3-connected planar graph with $|V| \geq 10$, at least $(|V|-1)/3$ in every connected graph with maximum degree three, and at least $|E|/(2\Delta-1)$ in every connected graph of maximum degree Δ.

7 CONSTANT-FACTOR APPROXIMATIONS

7.1 THE GREEDY ALGORITHM

The greedy algorithm we shall review in this section has been described independently by various researchers for different problems (degeneracy, maximum density, orientations, pseudoarboricity, and arboricity).

- Matula and Beck [MB83] describe the algorithm for determining a vertex ordering $(v_1, \ldots, v_{|V|})$ such that for every $1 \le i \le |V|$, v_i has minimum degree in the induced subgraph $G[\{v_i, \ldots, v_{|V|}\}]$. In particular, the algorithm computes the degeneracy[1] of a graph exactly. The authors analyze the runtime to be $\mathcal{O}(|E| + |V|)$. (For earlier explicit and implicit uses of the algorithm for coloring problems, see the references in [MB83].)

- Kortsarz and Peleg [KP94] give a $1/2$-approximation algorithm for the maximum density problem. It is essentially the greedy algorithm coupled with an unnecessary binary search. They claim its runtime to be $\mathcal{O}((|E| + |V| \log |V|) \log |V|)$ without giving details. Presumably, the term $|V| \log |V|$ corresponds to extracting the minimum from a priority queue, while the second $\log |V|$ comes from the binary search.

- Eppstein [Epp94] gives the greedy algorithm as a 2-approximation to the orientation problem in linear time, which is used to determine the arboricity approximately. He notes that the produced orientation is acyclic.

- Aichholzer et al. [AAR95] give the greedy algorithm for a 2-approximation of the orientation problem in linear time. They note the connection to the arboricity and maximum density.

- Arikati et al. [AMZ97] describe it as a linear-time 2-approximation algorithm for the arboricity.

- Charikar [Cha00a] (noting a similar algorithm by [Asa+00]) describes it as a $1/2$-approximation algorithm for the densest subgraph problem and notes the connection to the dual orientation problem. He also gives a greedy $(1/2 + \epsilon)$-approximation algorithm for the density in directed graphs with a runtime

1 The *degeneracy* is the smallest $k \in \mathbb{N}_0$ such that every induced subgraph has a vertex of degree at most k. It is closely related to the Szekeres–Wilf number [SW68].

of $\mathcal{O}(|E|/\epsilon \log |V|)$, which was improved by Khuller and Saha [KS09a] to $\mathcal{O}(|E|)$.

- Bezáková [Bez00] describes it as a 2-approximation algorithm for the pseudoarboricity and orientation problems.
- Georgakopoulos and Politopoulos [GP07] describe the algorithm as a $1/c$-approximation algorithm for the densest hypergraph problem where hyperedge sizes are bounded by c. It runs in time $\mathcal{O}(|V| \log |V| + |E|c)$, and in linear time for simple graphs ($c = 2$).
- Sozio and Gionis [SG10] extend the algorithm such that it can be used to find a connected subgraph containing a given set $S \subseteq V$ with certain distance and degree properties.
- Borradaile et al. [Bor+17] state the greedy algorithm without proving an approximation guarantee or a runtime bound, but show that it finds a smallest maximum indegree *acyclic* orientation. In fact, this 'acyclic orientation number' is equal to the degeneracy of the graph.

The greedy algorithm is very simple: It removes a vertex u of minimum degree and orients all (remaining) edges (u, v) towards u. This is repeated until all vertices have been removed. The algorithm can be made to run in $\mathcal{O}(|E|)$ time by storing the vertices in linked lists according to their current degrees. When a vertex with the current minimum degree is removed, it is unlinked from the corresponding list using a pointer to its position. Its hitherto unremoved neighbors drop in their degree and are moved between the linked lists accordingly, each in constant time. To find the minimum vertex degree, one iterates over the degree lists in ascending order until a nonempty list is found or all lists have been considered.

Since the sum of degrees in the original graph is $2|E|$, the number of times we visit lists is bounded by $2|E|$. As the current minimum degree d can drop by at most one, one may start the next search for a nonempty list from $d - 1$ instead of zero. Then, the number of lists that are visited can be even bounded by $\mathcal{O}(|V|)$ [Cha00a].

As each edge in the graph is considered a constant number of times, the algorithm runs in $\mathcal{O}(|E|)$ time. It obviously computes an orientation of the graph. Moreover, if at each iteration one considers the remaining vertices and edges as a subgraph, one can output the subgraph with the highest density among these in the end[2]. We now prove that the greedy algorithm simultaneously computes 1/2- and 2-approximations to the densest subgraph and orientation problems,

2 A trivial way of doing this is to run the algorithm once and record the maximum density encountered, and another time to determine the subgraph for which the maximum is attained.

respectively. We state the bounds slightly tighter than the original authors by rounding d^* appropriately.

Lemma 7.1.1 ([AAR95]). *Every subgraph H of a simple graph G contains a vertex of degree at most $\lfloor 2d^* \rfloor$.*

Proof. Assume there exists a subgraph H where every vertex has a degree greater than $2d^*$. By the degree sum formula, its average density is $|E_H|/|V_H| > (|V_H| 2d^*/2)/|V_H| = d^*$, a contradiction. Thus, there is a vertex of degree at most $2d^*$, and therefore of at most $\lfloor 2d^* \rfloor$. □

Theorem 7.1.2 ([AAR95]+[Chaooa]). *The greedy algorithm returns an acyclic d-orientation with $\lceil d^* \rceil \leq d \leq \lfloor 2d^* \rfloor$. Moreover, $d/2 \leq d(H)$ for the returned subgraph H.*

Proof. By Theorem 3.6.7, d is at least $\lceil d^* \rceil$. In every iteration, the vertex with minimum degree is removed, so at most $\lfloor 2d^* \rfloor$ edges are oriented towards it by Lemma 7.1.1. Thus, no vertex in the orientation produced by the algorithm has an indegree of more than $\lfloor 2d^* \rfloor$.

Clearly, the orientation is acyclic because the vertices form a topological ordering in reverse order of their removal.

For the second claim, assume that a vertex v is assigned more than $2d(H)$ edges to it in the algorithm, and consider the remaining subgraph K right before v is removed. By minimality of v's degree and the degree sum formula, the remaining subgraph K has density $d(K) = |E_K|/|V_K| > (|V_K| 2d(H)/2)/|V_K| = d(H)$. But then, K would have been returned instead of H, this is a contradiction. Therefore $d \leq 2d(H)$. □

Borradaile et al. [Bor+17] show that the orientation is optimal among the acyclic orientations of the graph, this will be proved in Theorem 9.0.1.

Corollary 7.1.3 ([Chaooa], [KS09a]). *The greedy algorithm returns a subgraph H with $\lceil d^* \rceil /2 \leq d(H) \leq d^*$.*

An alternative proof of Corollary 7.1.3 is possible with the help of Lemma 11.0.1 [KS09a]. We note that both $d/2$ and $d(H)$ are $1/2$-approximations to d^*, but $d/2$ is never closer to it than $d(H)$ by Theorem 7.1.2. Thus, we will use the value $d(H)$ as a lower bound for the preprocessing (Chapter 11) in our experiments in Chapter 12. This proves to be more effective than $d/2$, which was used in [Blu16].

7.2 THE ALGORITHM OF ASAHIRO ET AL.

Another approximation algorithm is given by Asahiro et al. [Asa+07] (Algorithm 7.1 on the next page), which achieves a $(2 - 1/\lceil d^* \rceil)$-approximation for the orientation problem. This also holds for graphs with positive edge-weights, where the maximum weighted indegree is to be minimized and the density is defined in the straightforward way (see Equation (3.8)). The authors analyze the runtime to be $\mathcal{O}(|E|^2)$. As we shall see, it is possible to implement the algorithm in linear time for unweighted graphs, and $\mathcal{O}(|E| + |V| \log |V|)$ for weighted graphs. Asahiro et al. do not argue why the algorithm terminates. Let l_i denote

Algorithm 7.1: The $(2 - 1/\lceil d^* \rceil)$-orientation algorithm of Asahiro et al. for unweighted graphs.

```
Input: A simple graph $G = (V, E)$.
Output: A $(2 - 1/\lceil d^* \rceil)$-orientation $\vec{G}$ of $G$ (if $\lceil d^* \rceil \geq 1$).
function orient($V, E$):
    Let $l \leftarrow |E|/|V|$
    Let $V_{orig} \leftarrow V$
    Let $\vec{E} \leftarrow \emptyset$
    repeat
        while $\exists v \in V$ with $\deg_G(v) \leq \lceil 2l \rceil - 1$ do
            foreach $uv \in E$ do
                orient $u \to v$ in $\vec{E}$
                $E \leftarrow E \setminus \{uv\}$
            $V \leftarrow V \setminus \{v\}$
        if $V = \emptyset$ then
            return $(V_{orig}, \vec{E})$
        $l \leftarrow |E|/|V|$    /* current sizes of the sets $V, E$ */
    until $\forall v \in V : \deg_G(v) = \lceil 2l \rceil$
    while there is cycle $(v_1, \ldots, v_k)$ in $G$ do
        orient $v_1 \to \cdots \to v_k \to v_1$ in $\vec{E}$    /* cycle loop */
        $E \leftarrow E \setminus \{v_1 v_2, \ldots, v_k v_1\}$
    /* A forest remains, orient towards the leaves */
    while $\exists v \in V$ with $\deg_G(v) \leq 1$ do
        orient $u \to v$ in $\vec{E}$ for the unique $uv \in E$
        $E \leftarrow E \setminus \{uv\}$
        $V \leftarrow V \setminus \{v\}$
    return $(V_{orig}, \vec{E})$
```

l and let V_i and E_i denote the vertices and edges of the remaining graph after the i-th iteration of the repeat-until loop. The authors claim correctly that[3] $\lceil 2l_{i+1} \rceil \geq \lceil 2l_i \rceil$, but strict inequality is needed to avoid an infinite loop. If we do not advance to the cycle loop, there exists at least one vertex whose degree is at least $\lceil 2l_i \rceil + 1$. Then, the

3 Although it might seem a trivial fact, it can not be deduced from Lemma 11.0.1. As the footnote to the remark after that lemma exhibits, even if the threshold is less than d^*, removing a vertex may result in a smaller average density. Thus, *during an iteration* of the inner loop the average density may well decrease between two vertex removals.

threshold will be higher in the next iteration of the repeat-until loop because of the degree sum formula and the choice of 2 as the constant:

$$\lceil 2l_{i+1} \rceil = \left\lceil 2\frac{|E_{i+1}|}{|V_{i+1}|} \right\rceil \geq \left\lceil 2\frac{|V_{i+1}| \lceil 2l_i \rceil + 1}{2|V_{i+1}|} \right\rceil = \lceil 2l_i \rceil + 1.$$

Theorem 7.2.1 ([Asa+07]). *The algorithm of Asahiro et al. finds a* $(\lceil 2d^* \rceil - 1)$*-orientation if* $d^* > 1/2$*, and a* $\lfloor 2d^* \rfloor$*-orientation otherwise.*

Proof. The claim is true if $|E| = 0$. If $d^* = 1/2$, the graph becomes 1-oriented. In the following, let $d^* > 1/2$.

All vertices removed in the repeat-until loop have an indegree of at most $\lceil 2l_{max} \rceil - 1 \leq \lceil 2d^* \rceil - 1$, where l_{max} is the maximum l attained in the repeat-until loop.

At the beginning of the cycle loop we have a $\lceil 2l \rceil$-regular graph whose maximum density is equal to $\lceil 2l \rceil$. Note that by arbitrarily orienting all remaining edges we would already obtain a 2-approximation. Furthermore, we could achieve our goal for *fractional* orientations by assigning every edge $1/2$ to each of its endpoints.[4]

In the forest loop the remaining forest is 1-oriented. The maximum indegree of the vertices processed in the cycle loop is bounded by $\lceil 2l \rceil - 1$ because a vertex is either contained in at least one cycle (thus it has at least one *outgoing* edge in $\vec{G}$) or it is only contained in the remaining forest (thus its maximum indegree is 1). The maximum indegree d_{max} is thus bounded by

$$d_{max} \leq \lceil 2d^* \rceil - 1 \leq 2\lceil d^* \rceil - 1.$$

□

Put differently, the algorithm finds a $(2 - 1/\lceil d^* \rceil)$-approximation to the orientation problem if $d^* > 1/2$.

One may wonder whether a constant $1 < c < 2$ could be used for a threshold of $\lceil cl \rceil - 1$ in the repeat-until loop in order to obtain smaller indegrees, because the cycle orientation loop essentially halves degrees of about $\lceil 2cl \rceil$. However, the degree sum formula does not guarantee termination of the repeat-until loop for such c.

Asahiro et al. give a straighforward runtime analysis (in the edge-weighted case). The repeat-until loop can be implemented in time $\mathcal{O}(|E||V|)$, the cycle loop in $\mathcal{O}(|E|^2)$ by performing depth-first searches, and the forest loop takes $\mathcal{O}(|V|)$. Let us now turn to an improved runtime analysis of the algorithm.

Proposition 7.2.2. *The algorithm of Asahiro et al. can be implemented in time* $\mathcal{O}(|V| + |E|)$ *for unweighted graphs, and in time* $\mathcal{O}(|V| \log |V| + |E|)$ *for edge-weighted graphs.*

4 In fact, if we are only interested in an approximation of the *value* $\lceil d^* \rceil$, we need not perform the forest loop and can save some time.

Proof. The iterations of the first loop can be performed in $\mathcal{O}(|E|)$ total time (instead of $\mathcal{O}(|V||E|)$ as analyzed by the authors) by essentially performing the greedy algorithm from the previous section with the degree-tracking lists, and the addition that we pause once the minimum degree is $\lceil 2l \rceil$ and reset l or leave the loop. In order to detect whether all remaining vertices have the same degree, we additionally keep track of the current maximum degree. The forest loop can also be seen as an execution of the greedy algorithm and can thus be implemented in linear time.

The cycle loop need not be performed in up to $\mathcal{O}(|E|)$ full depth-first searches to find one cycle per search (as described by the authors). It is possible to perform a single modified depth-first search that, once a cycle is found, orients along it and resumes the search in the first vertex of the cycle.[5] We omit the details for the following reason:

As pointed out by Fischer[6], a simple reduction can be used such that known properties and algorithms can be exploited. The resulting algorithm is essentially identical to the modified depth-first we had originally devised.

It is well known that the number of vertices with odd degree is always even, which is sometimes referred to as the 'handshaking lemma'. It is easily proved from the degree sum formula. Add a special vertex v^* and add an edge (u, v^*) for every vertex u of odd degree. Every vertex in this modified graph has even degree. It follows from Theorem 2.2.1 that every connected component has an Euler tour. Such tours can be found in all connected components with Hierholzer's algorithm [HW73] in $\mathcal{O}(|E|)$ total time. By orienting the edges along the tours, every vertex receives exactly half its modified degree, as the number of times we enter a vertex equals the number of times we leave it. In the end, one removes v^* and the additional edges. (The loop for 1-orienting the forest is now not necessary.) The indegree of a vertex of even degree is now exactly half its degree because no edge was added to it. The indegree of a vertex v of odd degree is either $(\deg(v) + 1)/2$ or $\lfloor \deg(v)/2 \rfloor$, depending on the orientation of the additional edge. Unless v's degree is equal to one, at least one edge must point away from v because its edges were oriented along the Euler tour. Therefore, the approximation factor is achieved in this variant of the algorithm.

In the case of edge weights, the algorithm can be implemented in $\mathcal{O}(|E| + |V| \log |V|)$ time: Using Fibonacci heaps [FT87], the vertex of minimum weighted degree can be extracted with a priority queue in $\mathcal{O}(\log |V|)$ amortized time, and the weighted degree of a vertex can be updated in constant amortized time. Strict Fibonacci heaps [BLT12] can even perform the operations in the same worst-case times. □

5 The algorithm is given in our preprint at `https://arxiv.org/abs/1811.06803v1`.
6 Frank Fischer, personal communication, November 2018.

7.3 KOWALIK'S SCHEME FOR FIXED ϵ

For any fixed $\epsilon > 0$, a $(1+\epsilon)$-approximation can be obtained with Kowalik's approximation scheme [Kow06] in time $\mathcal{O}(|E| \log |V| \log d^*)$ (see Section 4.2). It is moreover possible to stop the binary search once the ratio u_i/l_i is sufficiently small, and thus eliminate the factor $\log d^*$. We will also use this for our arboricity approximation scheme in Chapter 10.

Proposition 7.3.1. *For any constant $\epsilon > 0$, we can compute a d-orientation with $\lceil d^* \rceil \le d \le \lceil (1+\epsilon)d^* \rceil$ in time $\mathcal{O}(|E| \log |V|)$.*

Proof. Compute a 2-approximation d_G in linear time with the greedy algorithm and use $u_1 := d_G \le 2d^*$ as an initial feasible upper bound and $l_1 := d_G/2 \ge d^*/2$ as an initial lower bound. Note that $|E| \ge 1$ implies $l_1 > 0$. We keep the upper bound feasible at all times.

The binary search can be safely stopped once $u_i \le \lceil (1+\epsilon) l_i \rceil$, as then the feasible u_i can be returned as the approximation. Thus assume for test value $t_{i+1} = \lfloor (u_i + l_i)/2 \rfloor$ that $u_i > \lceil (1+\epsilon) l_i \rceil \ge (1+\epsilon) l_i$. Note that each test takes $\mathcal{O}(|E| \log |V|)$ time.

If test t_{i+1} fails, we update the lower bound to $\lfloor (u_i + l_i)/2 \rfloor + 1$ and get

$$\frac{u_{i+1}}{l_{i+1}} = \frac{u_i}{\left\lfloor \frac{u_i+l_i}{2} \right\rfloor + 1} < \frac{u_i}{\frac{(1+\epsilon)l_i + l_i}{2}} = \frac{2}{2+\epsilon} \frac{u_i}{l_i}. \tag{7.1}$$

If test t_{i+1} is successful, we update the upper bound to the feasible value $\lfloor (u_i + l_i)/2 \rfloor$ and obtain

$$\begin{aligned} \frac{u_{i+1}}{l_{i+1}} = \frac{\left\lfloor \frac{u_i+l_i}{2} \right\rfloor}{l_i} \le \frac{\frac{u_i+l_i}{2}}{l_i} &\le \min\left(\frac{1}{2}\frac{u_i}{l_i} + \frac{1}{2}, \frac{2+\epsilon}{2+2\epsilon} \frac{u_i}{l_i} \right) \\ &\le \frac{1}{2}\frac{u_i}{l_i} + \frac{1}{2}. \end{aligned} \tag{7.2}$$

(The last inequality is an observation that is of no consequence here.)

Since ϵ is fixed, the bound ratio decays exponentially by (7.1) and (7.2). A constant number of tests suffice to reduce the initial ratio of $u_1/l_1 \le \frac{2d^*}{d^*/2} = 4$ to $1+\epsilon$ or less. Therefore, we have a runtime of $\mathcal{O}(|E| \log |V|)$. □

Note that the reduction in the success case is worse than in the failure case for $u_i/l_i \le \frac{2+\epsilon}{2-\epsilon}$. Otherwise, the reduction in the failure case is worse.

The constant introduced in the proof can be rather large. In order to reach $1+\epsilon$ from the initial ratio of four, the number of tests is at most 27 for $\epsilon = 0.1$, and at most 2772 for $\epsilon = 0.001$. (In addition, a factor of roughly ϵ^{-1} in the runtime comes from the Taylor expansion of $\log_{1+\epsilon}$ in Kowalik's scheme!)

However, we can first approximate with the modified scheme to obtain an initial ratio smaller than four. For example, by setting $\epsilon = 0.1$, we get a ratio of at most $1.1^2 = 1.21$. The number of tests for $\epsilon = 0.001$ is then at most 380. Therefore, we can save some time by approximating repeatedly.

8 ARBORICITY AND PSEUDOARBORICITY

Nur der Einsame findet den Wald;
wo ihn mehrere suchen, da flieht er,
und nur die Bäume bleiben zurück.

— Peter Rosegger, Schriften des Waldschulmeisters (1875)

In this chapter, we will see the connection of the orientation problem to a covering problem on matroids. Throughout the chapter, we will assume the ground sets of matroids to be finite and nonempty. We also assume that graphs have at least two vertices. We partly follow the presentation in the textbook by Scheinerman and Ullman [SU13], which draws on the work of Payan [Pay86] and Catlin et al. [Cat+92].

8.1 FORESTS AND PSEUDOFORESTS

The following characterization of acyclic simple graphs is well known and will lead to our first example of a matroid.

Proposition 8.1.1 (E.g., [Bap14])**.** *The columns of the incidence matrix $I(G)$ of a simple graph $G = (V, E)$ are linearly independent over $\{0, 1\}$ if and only if the graph is acyclic (i.e., a forest).*

Proof. '$\Rightarrow$': Assume G contains a cycle with edges $e_1, \ldots, e_n$. Then by taking the sum over the corresponding columns, we obtain the zero vector. This is due to the fact that the number of cycle edges incident to a vertex is even and we operate over $\{0, 1\}$.

'$\Leftarrow$': If G is a forest, then for any subset $X \subseteq E$, the graph (V, X) is also a forest. Every tree of at least two vertices has at least one vertex of degree one. The claim holds if $|E| = 0$. Now let $|E| \geq 1$. When adding up the columns corresponding to any $X \neq \emptyset$, there is a one in at least one row. $\square$

Let us consider a definition of 'almost acyclic' graphs.

Definition 8.1.2. A graph G is a *pseudoforest* if every connected component of G has at most one simple cycle. If G is connected, it is called a *pseudotree*. If a pseudotree contains exactly one cycle, it is called a *unicyclic component*. A component of a graph that is not a pseudotree is said to be *bicircular*.

Gabow and Westermann observe that a graph is a pseudoforest if and only if it has a 1-orientation [GW92, Lemma 3.2]. A generalization of this fact with a rigorous proof will be given in Theorem 8.2.4.

It provides an alternative proof to the following characterization of pseudoforests.

Proposition 8.1.3 ([Whi88])**.** *A graph (V,E) is a pseudoforest if and only if $|E[S]| \leq |S|$ for all $S \subseteq V$.*

Proof. By Theorem 3.6.8, a graph has a 1-orientation if and only if $|E[S]| \leq |S|$ for every $S \subseteq V$. A 1-orientation exists if and only if the graph is a pseudoforest (Theorem 8.2.4). □

Definition 8.1.4. For a graph $G = (V,E)$ we define the set systems

$$\begin{aligned}\mathcal{F}(G) &:= \{F \subseteq E \mid (V,F) \text{ is a forest}\},\\ \mathcal{B}(G) &:= \{P \subseteq E \mid (V,P) \text{ is a pseudoforest}\}.\end{aligned}$$

We will soon see that $\mathcal{F}$ and $\mathcal{B}$ are matroids. $\mathcal{F}$ is called the *cycle matroid*, while $\mathcal{B}$ is called the *bicircular matroid*, and the two are closely related.[1] A matroid is called *graphic* if it is (isomorphic to) the cycle matroid of a graph.

While rank functions are defined on matroids, we can assume they are defined on set systems for an easier presentation.

Proposition 8.1.5 ([Tut65],[Zas82])**.** *Let $G = (V,E)$ be a simple graph. Then $\mathcal{F}(G)$ and $\mathcal{B}(G)$ are independence systems.*

For $X \subseteq E$, let $n(X) = |V[X]|$. Let $c(X)$ denote the number of connected components and $a(X)$ the number of acyclic connected components of $(V[X],X)$, respectively (if $X = \emptyset$, we define $c(X) = 0 = a(X)$). The rank functions of $\mathcal{F}$ and $\mathcal{B}$ are

$$\rho_{\mathcal{F}}(X) = n(X) - c(X), \tag{8.1}$$
$$\rho_{\mathcal{B}}(X) = n(X) - a(X). \tag{8.2}$$

Proof. Clearly, $X = \emptyset$ is in both $\mathcal{F}$ and $\mathcal{B}$, and both set systems are closed under the subset relation. Hence, they are independence systems. For the rank functions, first consider sets $X \in \mathcal{M}$, where $\mathcal{M} \in \{\mathcal{F},\mathcal{B}\}$. The claim is seen to be true for $|X| \in \{0,1\}$. In the following, let $|X| \geq 2$.

We prove (8.1) for $X \in \mathcal{F}$ by induction over the number of connected components $c(X)$. If $c(X) = 1$, we have a tree, and the claim is true. Let now $c(X) \geq 2$ and let the claim hold for all forests $\tilde{X}$ with $c(\tilde{X}) < c(X)$. By inserting an artificial edge e between two different connected components of $(V[X],X)$, we obtain a forest $X \cup \{e\}$ of $c(X)-1$ connected components and can use the induction hypothesis to obtain

$$\begin{aligned}\rho_{\mathcal{F}}(X) + 1 = \rho_{\mathcal{F}}(X \cup \{e\}) &= n(X \cup \{e\}) - c(X \cup \{e\})\\ &= n(X) - (c(X) - 1).\end{aligned}$$

1 In fact, $\mathcal{B}$ is the 1-*elongation matroid* of $\mathcal{F}$ [SU13].

The claim follows by subtracting one on both sides.

In order to prove (8.2) for $X \in \mathcal{B}$, let $u(X)$ denote the number of unicyclic components of $(V[X], X)$. Remove one arbitrary edge from the unique cycle in every unicyclic component. A forest $\tilde{X}$ remains with

$$\begin{aligned}\rho_{\mathcal{F}}(\tilde{X}) \overset{(8.1)}{=} n(\tilde{X}) - c(\tilde{X}) &= n(X) - c(X) \\ &= n(X) - (u(X) + a(X)).\end{aligned} \tag{8.3}$$

Thus,

$$\rho_{\mathcal{B}}(X) = \rho_{\mathcal{B}}(\tilde{X}) + u(X) = \rho_{\mathcal{F}}(\tilde{X}) + u(X) \overset{(8.3)}{=} n(X) - a(X),$$

and the claim follows.

If $X \notin \mathcal{M}$, then identify the cyclic connected components. For every such component, while there is more than one cycle, remove an arbitrary edge on a cycle. For $\mathcal{F}$, we continue further until there is no cycle. This process does not disconnect the connected components and in case of $\mathcal{B}$, no cyclic component becomes ayclic. In particular, no nonisolated vertex becomes isolated. Hence, for the remaining edges $Y \subseteq X$, we have $Y \in \mathcal{M}$, $n(Y) = n(X), c(Y) = c(X)$, and $a(Y) = a(X)$ for $\mathcal{M} = \mathcal{B}$. Since the choice of edges was arbitrary, there is no *independent* subset of X with larger cardinality.

□

We will now show the third matroid property for $\mathcal{F}$ and $\mathcal{B}$.

Proposition 8.1.6 ([Whi35],[Sim72]). *Given a simple graph $G = (V, E)$, $\mathcal{F}(G)$ and $\mathcal{B}(G)$ are matroids for ground set E.*

Proof. By Propositions 2.10.1 and 8.1.1, $\mathcal{F}$ is a matroid. Alternatively, this could be proved with the help of Proposition 8.1.5. Let us now turn to $\mathcal{B}$, and define $n(X)$ and $a(X)$ as in Proposition 8.1.5.

We already know that $\mathcal{B}$ is an independence system. For the third matroid property, let X and Y be pseudoforests on V with $|X| > |Y|$. For every $e \in X \setminus Y$, consider $H_e = (V, Y \cup \{e\})$.

If e joins a tree with a pseudotree, or e connects two vertices in the same tree, H_e is a pseudoforest. Thus we have found an $e \in X \setminus Y$ such that $Y \cup \{e\} \in \mathcal{B}$. Otherwise, e joins two unicyclic components or connects two vertices inside a unicyclic component. In these cases, H_e contains a single bicircular component. We assume in the following that adding any $e \in X \setminus Y$ results in a bicircular component.

If a vertex u is isolated in (V, Y), then it is also isolated in (V, X), for otherwise we could have added some $(u, v) \in X \setminus Y$. Hence $n(Y) \geq n(X)$. If all connected components of (V, Y) are (uni-)cyclic, we have by (8.2)

$$\rho_{\mathcal{B}}(Y) = n(Y) \geq n(X) \geq \rho_{\mathcal{B}}(X),$$

a contradiction. Therefore, an acyclic connected component $A \subseteq Y$ exists. Every $e \in X$ incident to a vertex of $V[A]$ must be in Y, for otherwise it could have been added. Therefore, a surjective map from the acyclic components of (V, X) to the acyclic components of (V, Y) exists, and hence $a(Y) \leq a(X)$. By Equation (8.2), we have

$$|Y| = \rho_{\mathcal{B}}(Y) = n(Y) - a(Y) \geq n(X) - a(X) = \rho_{\mathcal{B}}(X) = |X|,$$

a contradiction. □

We note that the bicircular matroid also has a (less obvious) matrix representation [Zas82; CGW91].

8.2 MATROID PARTITIONING AND COVERING NUMBERS

Definition 8.2.1. A *matroid k-partition* over a matroid $\mathcal{M}$ for ground set S is a partition $(S_1, \ldots, S_k)$ of S such that each S_i is an independent set in $\mathcal{M}$.

One can also partition S into independent sets from different matroids $(\mathcal{M}_1, \ldots, \mathcal{M}_k)$ over S. However, here we only consider partitions where all matroids are identical.

Definition 8.2.2. The *covering number* of a loopless[2] matroid $\mathcal{M}$ over S, denoted $k(\mathcal{M})$, is the smallest integer k such that a matroid k-partition of S exists.

Given a simple graph G, the *arboricity* $\Gamma(G)$ is the covering number of the cycle matroid $\mathcal{F}$, and the *pseudoarboricity* $p(G)$ is the covering number of the bicircular matroid $\mathcal{B}$.

Examples of forest and pseudoforest partition can be seen in Figure 1.3 on page 3.

We already defined $p(G)$ in Section 3.6, and it will now become apparent why the letter p was chosen. The following theorem is due to Picard and Queyranne.

Theorem 8.2.3 ([PQ82])**.** *For a simple graph G, $p(G) = \lceil d^*(G) \rceil$.*

An alternative proof of this theorem based on matroid theory, due to [SU13], will be developed in this section (another, also based on matroid theory, is given by Westermann [Wes88]). By Theorem 3.6.7, we then know that the pseudoarboricity equals the smallest maximum indegree. However, this also follows from an accessible algorithmic theorem given independently by Bezáková and Kowalik.

Theorem 8.2.4 ([Bez00; Kow06])**.** *Let G be a simple graph and $d \in \mathbb{N}_0$. There is a partition into d pseudoforests if and only if a d-orientation exists, and one can be computed from the other in time $\mathcal{O}(|E|)$.*

2 $k(\mathcal{M})$ exists if and only if $\mathcal{M}$ is loopless, otherwise we would write $k(\mathcal{M}) = \infty$.

Proof. The case $d = 0$ is obvious.

'$\Rightarrow$': For $d \geq 1$, let $(P_1, \ldots, P_d)$ be a pseudoforest partition of E. Every pseudotree can be 1-oriented: If it is a tree, turn it into an arborescence, which has maximum indegree one. In presence of a cycle pick an edge uv on the cycle. Set aside uv, now the remaining graph can be turned into an arborescence rooted at v. The indegree of any vertex is at most one, and this still holds when we orient the remaining edge as $u \to v$.

It follows that a pseudoforest is 1-orientable. Carry out this procedure for the d pseudotrees to obtain a d-orientation, which is possible in linear time.

'$\Leftarrow$': Let $\vec{G}$ be a d-orientation of G. We prove by induction that G can be partitioned into d pseudoforests. It is not diffult to see that this proof can be turned into a linear-time algorithm.

Consider for the base case $d = 1$ a connected component (in the undirected sense) with n vertices. It has at least $n - 1$ edges because it contains a spanning tree. It has at most n edges, for otherwise a 1-orientation would not be possible by the pigeonhole principle. Therefore, the component is a tree or unicyclic. Hence, each component is a pseudotree and the graph is a pseudoforest.

Let the induction hypothesis hold for all $d' < d$ for some $d \geq 2$. For every vertex in a d-orientation, take away one edge entering it (if any). This is a 1-orientation and thus can be converted into a pseudoforest. The remaining graph is a $d - 1$ orientation and can be converted into $d - 1$ pseudoforests by the induction hypothesis. □

Corollary 8.2.5. *The pseudoarboricity equals the smallest maximum indegree.*

As every forest is a pseudoforest, the pseudoarboricity is a lower bound on the arboricity. However, Picard and Queyranne prove a much closer relationship in Theorem 8.2.7. An alternative proof of it that uses matroid clumps is given by Westermann [Wes88]. We will develop a new and algorithmic proof in Chapter 10.

Theorem 8.2.6 ([PQ82])**.** *Let G be a simple graph. Then*

$$\Gamma(G) \in \{p(G), p(G) + 1\}.$$

Figure 1.3 on page 3 shows an example of a graph with $p = 2$ and $\Gamma = 3$. As we shall see, Theorem 8.2.6 is used in an arboricity algorithm by Gabow and Westermann (Theorem 8.3.2), and we will use it for a new runtime result in Chapter 11.

In order to give a first proof of Theorem 8.2.6, let us look at a classic theorem by Nash-Williams [Nas64] (see also [Nas61] for a related packing problem), which was independently discovered in a more general form by Edmonds [Edm65]. Alternative proofs are given by Chen et al. [Che+94] and Reiher and Sauermann [RS14].

Theorem 8.2.7 ([Nas64; Edm65]). *Let G be a simple graph. Then*

$$\Gamma(G) = \max_{\substack{(V_H,E_H)\subseteq G,\\ |V_H|\geq 2}} \left\lceil \frac{|E_H|}{|V_H|-1} \right\rceil . \tag{8.4}$$

That $\Gamma(G)$ is at least the expression on the right-hand side of (8.4) is not difficult to see, as [SU13] explains: Every acyclic spanning subgraph of G has at most $|V|-1$ edges, so $\Gamma(G) \geq |E|/(|V|-1)$, and we can round up because $\Gamma(G)$ is an integer. As the arboricity of a graph is at least the arboricity of any of its subgraphs, we get the desired inequality. That (8.4) holds with equality is, however, not immediate. Let us first assume it as fact and prove Theorem 8.2.6 with it, as in [PQ82].

Definition 8.2.8. Given a simple graph G, the *fractional arboricity*[3] $\gamma(G)$ is defined as

$$\gamma(G) := \max_{\substack{(V_H,E_H)\subseteq G,\\ |V_H|\geq 2}} \frac{|E_H|}{|V_H|-1}. \tag{8.5}$$

Note that although γ is very close to d^*, a densest subgraph does not necessarily maximize (8.5). An example are all but one densest subgraphs in Figure 3.1 on page 50. Likewise, a subgraph maximizing (8.5) need not be a densest subgraph, an example is given in Figure 9.5 on page 134.

Lemma 8.2.9 ([PQ82]). *A simple graph G satisfies $\gamma(G) \leq d^*(G) + 1/2$.*

Proof. Let $D = (V_D, E_D)$ denote a densest subgraph and $H = (V_H, E_H)$ a subgraph that maximizes (8.5). If we had $\gamma > d^* + 1/2$, this would yield

$$\frac{|E_H|}{|V_H|-1} > \frac{|E_D|}{|V_D|} + 1/2 \geq \frac{|E_H|}{|V_H|} + 1/2.$$

By rearranging, we obtain

$$|E_H| > \frac{|V_H|(|V_H|-1)}{2},$$

i.e., more edges than a complete graph on $|V_H|$ vertices can have. This is a contradiction. □

Proof of Theorem 8.2.6. The claim holds if $|E| = 0$. In the following, let $|E| \geq 1$. Since trivially $p(G) \leq \Gamma(G)$ holds, we have to show that $\Gamma(G) \leq p(G) + 1$. From Lemma 8.2.9 and (8.4) we obtain the desired $\Gamma(G) \leq \lceil d^* \rceil + 1 = p(G) + 1$. □

3 We note that if the $|V_H|$ are required to be odd, the resulting measure for multi-graphs has an interesting relationship with edge colorings [Sey79; CZZ19].

Corollary 8.2.10 ([PQ82]). *If $d^* \neq 0$ is an integer, then $\Gamma = p + 1$, and if $d^* \leq p - 1/2$, then $\Gamma = p$.*

In order to compute the covering number and the corresponding partition, Edmonds [Edm65] proposes the following matroid partitioning algorithm. Given a test value k, the algorithm finds a k-partition of S if one exists, and otherwise returns an independent set Y with $|Y| > k\rho(Y)$, which is a certificate that no k-partition exists. The mapping of elements to the sets of a partition is called a *coloring*, i.e., every set of the partition corresponds to a distinct color. If some elements are unassigned, we have a partial coloring.

It is possible to assign color i to an element $x \notin S_i$ if $S_i \cup \{x\}$ is independent. For short, we write $x \leftarrow \langle i \rangle$.

For elements x, y where y is colored as $y \in S_i$ and $x \notin S_i$ has a different color or is uncolored, we write $x \leftarrow y$ if $S_i \setminus \{y\} \cup \{x\}$ is independent. This means that we can re-color: x gets y's color, and y loses its color. More generally, we can re-color entire paths. The relation $\leftarrow$ defines a directed graph on the set S.

Lemma 8.2.11 ([Edm65]). *Let $\mathcal{M}$ be a matroid and $S_1, \ldots, S_k$ be a partial coloring of S. Suppose there is a directed path*

$$x_0 \leftarrow x_1 \leftarrow \cdots \leftarrow x_n$$

that is minimal in the sense that $x_i \not\leftarrow x_j$ for $i > j + 1$ and $x_i \not\leftarrow \langle a \rangle$ for $i < n$, then modifying $(S_1, \ldots, S_k)$ by re-coloring the $x_0, \ldots, x_n$ according to the path is a proper partial coloring of S.

While the lemma is quite intuitive, it is not trivial because re-coloring a single element changes the $\leftarrow$-relation globally. We omit the lengthy, technical proof and refer the reader to the textbook by Scheinerman and Ullman [SU13, Lemma 5.3.1]. Matroid partitioning is now possible by repeatedly applying Lemma 8.2.11 to uncolored vertices that serve as the end x_0 of the path.

The following theorem, together with its proof, establish the correctness of Algorithm 8.1 on the following page. We note that we saw a special case of Lemma 8.2.11 and Theorem 8.2.12 for $\mathcal{B}$ in disguise in Theorem 3.6.8 and its proof (the path reversal algorithm), which becomes apparent by the equivalence of pseudoforest d-partitions and d-orientations (Theorem 8.2.4).

Theorem 8.2.12 ([Edm65]). *Let $\mathcal{M}$ be a loopless matroid over S and let $k \in \mathbb{N}_0$. Then S has a partition into k independent sets if and only if $|Y| \leq k\rho_{\mathcal{M}}(Y)$ for all $Y \subseteq S$.*

Proof. If Algorithm 8.1 never reaches the else-block, then it stops with a k-partition of S because the number of uncolored vertices decreases in the if-block. As every set $Y \subseteq S$ must be distributed among the k sets of a k-partition, the latter's existence implies that $|Y| \leq k\rho(Y)$.

Algorithm 8.1: Edmonds's matroid partitioning algorithm for test value k.

```
Input: A loopless matroid $M$ over a (finite) ground set $S$, and
       $k \in \mathbb{N}$.
Output: A partition $S = S_1 \dot\cup \ldots \dot\cup S_k$ with $S_i \in M$ for all
        $i = 1, \ldots, k$, or a $Y \subseteq S$ with $|Y| > k\rho(Y)$.
$S_i \leftarrow \emptyset$ for all $i = 1, \ldots, k$
while $S \neq \bigcup_{i=1}^{k} S_i$ do
    // There exist uncolored vertices
    if there is a directed path from a color class to an uncolored
      element $x_0$ then
        re-color along a minimal such directed path
    else
        Let $Y$ be the set of elements from which an uncolored
          element can be reached in the directed graph
        return $Y$
return $(S_1, \ldots, S_k)$
```

We now prove that if the algorithm reaches the else-block, then the returned Y has $|Y| > k\rho(Y)$. Let $U \neq \emptyset$ denote the set of uncolored vertices, and let $V = Y \setminus U$. We have $|Y| > |V|$.

We show that

$$\rho(Y) = |Y \cap S_i| \quad \forall i = 1, \ldots, k. \tag{8.6}$$

As $Y \cap S_i$ is an independent set, we have $\rho(Y) \geq |Y \cap S_i|$. Now assume $\rho(Y) > |Y \cap S_i|$. Thus there exists $x \in Y \setminus S_i$ such that $(Y \cap S_i) \cup \{x\}$ is independent.

Consider $S_i \cup \{x\}$, which is not independent because then there would be a directed path $x \leftarrow \langle i \rangle$, and thus we would not have entered the else-block. So $S_i \cup \{x\}$ contains a (unique) minimal dependent set C that is (by independence of $(Y \cap S_i) \cup \{x\}$) not contained in Y, so there is a $z \in C \setminus Y$. In particular, $z \neq x$ and hence $z \in S_i \setminus Y$. Thus $(S_i \setminus \{z\}) \cup \{x\}$ is independent, so $x \leftarrow z$.

This, however, is a contradiction to the fact that $x \in Y$ but $z \in Y$. This proves (8.6). Finally, we obtain

$$k\rho(Y) = \sum_{i=1}^{k} \rho(Y) \overset{(8.6)}{=} \sum_{i=1}^{k} |Y \cap S_i| = \left| \bigcup_{i=1}^{k} Y \cap S_i \right| = |V| < |Y|.$$

□

Corollary 8.2.13 ([Edm65]). *Let $\mathcal{M}$ be a loopless matroid, and let ρ denote its rank function. Then its covering number is*

$$k(\mathcal{M}) = \max_{\substack{Y \subseteq S \\ Y \neq \emptyset}} \left\lceil \frac{|Y|}{\rho_{\mathcal{M}}(Y)} \right\rceil. \tag{8.7}$$

Theorem 8.2.7 can now be obtained from this corollary with the rank function of $\mathcal{F}$: One can show that the maximum of (8.7) is attained at a set Y such that $(V[Y], Y)$ is connected and not an isolated vertex. For such a set Y, the rank function (8.1) is $\rho_{\mathcal{F}}(Y) = |V[Y]| - 1$. For the full proof (which is somewhat similar to the proofs of Propositions 3.1.3 and 3.1.4), see [SU13].

Analogously, we can proceed with the pseudoarboricity. Similarly to the fractional arboricity (8.5), we can define the fractional covering number.

Definition 8.2.14. The *fractional covering number* of a loopless matroid $\mathcal{M}$ over ground set S is

$$k_f(\mathcal{M}) := \max_{\substack{Y \subseteq S \\ Y \neq \emptyset}} \frac{|Y|}{\rho_{\mathcal{M}}(Y)}.$$

Indeed, the fractional covering number of the cycle matroid $\mathcal{F}$ is the fractional arboricity defined in (8.5). Let us now turn to the fractional pseudoarboricity. The following theorem was stated by Scheinerman and Ullman for connected cyclic graphs. However, the theorem holds for all graphs that contain a cycle.

Theorem 8.2.15 ([SU13])**.** *If G contains a cycle, then $d^*(G) = k_f(\mathcal{B}(G))$.*

Proof. We know by Proposition 3.1.4 that $d^*(G)$ is attained on a connected subgraph $H = (V_H, E_H)$ of G. By Lemma 3.1.5 we can conclude that H must contain a cycle as well. Thus, according to Proposition 8.1.5, the rank of E_H is $\rho_{\mathcal{B}}(E_H) = |V_H|$. Thus,

$$k_f(\mathcal{B}) \geq \frac{|E_H|}{|V_H|} = d^*(G).$$

To show the other inequality, note that for every set $X \subseteq E$ such that (V, X) is acyclic, we have

$$\begin{aligned} \rho_{\mathcal{B}}(X) = n(X) - a(X) &\overset{(8.1)}{=} (\rho_{\mathcal{F}}(X) + a(X)) - a(X) \\ &= (|X| + a(X)) - a(X) = |X|. \end{aligned}$$

Hence $|X|/\rho_{\mathcal{B}}(X) = 1$. Thus, if the fractional covering number is greater than one, the set Y maximizing it must induce a cyclic graph, and we can restrict ourselves to Y such that all connected components of (V, Y) are cyclic. Thus, $\rho_{\mathcal{B}}(Y) = n(Y)$, and

$$k_f(\mathcal{B}) = \frac{|Y|}{\rho_{\mathcal{B}}(Y)} = \frac{|Y|}{n(Y)} \leq d^*.$$

□

Note that the theorem fails for graphs where all components are acyclic, e.g., the graph with two vertices and one edge. As a corollary to Theorem 8.2.15, we obtain Theorem 8.2.3.

Proof of Theorem 8.2.3. If $|E| = 0$, $p = 0$ and $d^* = 0$. Let now $|E| \geq 1$. If the graph is disconnected, decompose it into its connected components. The pseudoarboricity and the maximum density of a graph are exactly the maximum pseudoarboricity and the highest maximum density among its connected components (see Section 3.1). Thus for the remainder we assume G is connected.

If the graph is a tree, then $p = 1$ and $0 < d^* < 1$, hence the claim holds. If the graph is cyclic, apply Theorem 8.2.15 and Corollary 8.2.13 to obtain

$$\lceil d^* \rceil = \lceil k_f(\mathcal{B}) \rceil = k(\mathcal{B}) = p.$$

□

8.3 RUNTIMES FOR COMPUTING THE (FRACTIONAL) ARBORICITY

Let us now examine the runtime of Edmonds's algorithm to determine the covering number. If the matroid is unspecified, we count the number of times we ask for independence of a set. This is called the number of accesses to the *independence oracle*.

Theorem 8.3.1 (Essentially [SU13])**.** *The covering number of a matroid M over S can be determined with $\mathcal{O}(|S|^3 \log |V|)$ queries to the independence oracle.*

Proof. Perform a binary search for the minimum k such that Algorithm 8.1 returns a k-partition. This needs $\mathcal{O}(\log |S|)$ tests. (In [SU13] a linear search is used for up to $|S|$ tests.)

As there are $|S|$ elements to be added to the partition, the 'then'-block is executed at most $|S|$ times. The number of independence tests performed in the block is $\mathcal{O}(|S|^2)$. □

Gabow and Westermann propose partitioning algorithms specifically for arboricity and pseudoarboricity (see also [PQ82; GS85]). (Algorithms for the special case of planar graphs will be treated in Sections 9.2.5 and 9.2.6.) In Westermann's thesis [Wes88], the algorithm for pseudoarboricity is formulated in terms of matroids entirely (though it borrows ideas from the analysis of the algorithms of Dinitz and Hopcroft–Karp), the subsequent paper [GW92] uses the bipartite network (Section 3.6) and 'degree-constrained' subgraphs (in other words, orientations). The analysis, however, is still in terms of matroids, and we re-phrased it in terms of flows entirely in Chapter 5.

Theorem 8.3.2 ([Wes88; GW92])**.** *The pseudoarboricity p, along with a partition into p pseudoforests, can be determined in time*

$$\mathcal{O}\left(|E| \min\left(\sqrt{|E| \log p}, (|V| \log p)^{2/3}\right)\right).$$

The arboricity Γ, along with a partition into Γ forests, can be determined in time

$$\mathcal{O}\left(|E|^{5/3}\log^{1/3}|V| + |E|^{4/3}|V|^{1/3}\log^{2/3}|V|\right),$$

and, if $|E| \in \Omega(|V|^{3/2}\log|V|)$ holds, in time

$$\mathcal{O}(|V||E|\log|V|).$$

We note that in [GW92, Table 1] the condition $|E| \in \Omega(|V|^{3/2}\log|V|)$ is erroneously stated for the first of the two time bounds. This first runtime bound uses a conversion of pseudoforest partitions into forest partitions, which we will review in Section 9.1.

In general, the arboricity can be determined by computing the pseudoarboricity p and then testing whether a partition into p forests exists with a matroid partitioning algorithm for $\mathcal{F}$. If p is not feasible, then $\Gamma = p + 1$ by Theorem 8.2.6.[4] This eliminates the binary search.

In Chapter 11, we will work the other way around in a certain scenario: We compute the arboricity of a subgraph first, which leaves a constant-size interval for the pseudoarboricity to check. The arboricity algorithm that we will use is due to Gabow.

Theorem 8.3.3 ([Gab98])**.** *The arboricity $\Gamma(G)$, along with a partition into Γ forests, can be determined in time $\mathcal{O}(|E|^{3/2}\log(|V|^2/|E|)$.*

The algorithm uses Newton's method on polymatroids and an extension of the Hao–Orlin algorithm for minimum cuts [HO92]. A discussion would carry us too far afield.

Picard and Queyranne [PQ82] reduce the pseudoarboricity and arboricity problems to 0-1 fractional programming problems (see Subsection 3.3.1). We note that in this approach the arboricity and maximum density are determined with $\mathcal{O}(|V|)$ minimum cut computations, while the pseudoarboricity is determined with $\mathcal{O}(\log|V|)$ minimum cut computations in a binary search. This is somewhat peculiar because we saw approaches based on binary search for the maximum density in previous chapters, and the maximum density problem is dual to the relaxation of the orientation problem.

Approximation algorithms for the fractional covering problem are described by Plotkin et al. [PST91]. A recent approximation scheme for the fractional arboricity γ specifically was given by Toko Worou and Galtier.

Theorem 8.3.4 ([TG16])**.** *For a simple graph $G = (V, E)$ and every $\epsilon > 0$, a subgraph (V_H, E_H) of G satisfying*

$$\frac{\gamma(G)}{1+\epsilon} \leq \frac{|E_H|}{|V_H| - 1} \leq \gamma(G)$$

can be found in $\mathcal{O}(|E|\log^2|V|\log(|E|/|V|)\epsilon^{-2})$ time.

4 If the corresponding partition into forests is desired, start another test for $p + 1$.

Note that the dependence on ϵ is inversely quadratic and hence this runtime would not be fast enough for the search interval shrinking in Chapter 6.

The algorithm uses a linear programming formulation of the arboricity problem in the minimization sense and then approximately solves the dual problem. To this end, maximum spanning trees are computed with respect to a weight function that corresponds to a dual LP solution. After a maximum spanning tree T^i has been computed, the weights are modified and a maximum spanning tree T^{i+1} is computed on the modified weights. After $K \simeq \gamma \ln |E|/\epsilon^2$ iterations, the algorithm stops and the desired subgraph can be extracted in time $\mathcal{O}(|E| \log |V|)$ from the dual LP solution. In order to set K, a 2-approximation algorithm is used, just as we did in Chapter 6. In order to remove the runtime dependency on γ, the authors use a fast preprocessing that is very similar to the one we will use in Chapter 11.

Recall that Kowalik's approximation scheme (Section 4.2) approximates d^* as a value via the dual orientation problem, but does not compute approximately densest subgraphs. On the other hand, the approximation scheme of Toko Worou and Galtier does not compute a forest partition, but it approximates the densest subgraph by computing approximately densest subgraphs for the slightly different density measure γ. We will address the covering problem for $\mathcal{F}$ constructively in Chapters 9 and 10.

8.4 BOUNDS FOR ARBORICITY AND PSEUDOARBORICITY

The arboricity was first bounded by Chiba and Nishizeki.

Theorem 8.4.1 ([CN85]).

$$\Gamma(G) \leq \left\lceil \sqrt{\frac{|E|}{2} + \frac{|V|}{4}} \right\rceil \tag{8.8}$$

We can improve over it.

Proposition 8.4.2. *The arboricity bound*

$$\Gamma(G) \leq \left\lceil \sqrt{\frac{|E|}{2} + \frac{1}{16}} + \frac{1}{4} \right\rceil \tag{8.9}$$

derived from Proposition 3.2.2 is tighter than bound (8.8).

Proof. By the proof of Theorem 8.2.6, $\Gamma(G) \leq \lceil d^* + 1/2 \rceil$, we obtain (8.9) with Proposition 3.2.2.

Consider a graph with five vertices and six edges (with no isolated vertex for a fair comparison). Equation (8.8) yields an arboricity bound of 3, while our bound yields 2. Hence, our bound is tighter on this instance.

Let us now show for all graphs that the right-hand side of (8.9) is at most the right-hand side of (8.8).

For any $|E| \geq 0$ we can write $|E| = n(n-1)/2$ for a unique $n \in \mathbb{R}^+$. Then $|V| \geq n$ because no graph can have greater density than a complete graph. This means that

$$\left\lceil \sqrt{\frac{|E|}{2} + \frac{|V|}{4}} \right\rceil \geq \left\lceil \sqrt{\frac{n(n-1)}{4} + \frac{n}{4}} \right\rceil = \left\lceil \frac{n}{2} \right\rceil .$$

Now consider bound (8.9). We have

$$\begin{aligned}
\left\lceil \sqrt{\frac{|E|}{2} + \frac{1}{16}} + \frac{1}{4} \right\rceil &= \left\lceil \sqrt{\frac{n(n-1)}{4} + \frac{1}{16}} + \frac{1}{4} \right\rceil \\
&= \left\lceil \sqrt{\frac{n^2 - n + 1/4}{4}} + \frac{1}{4} \right\rceil \\
&= \left\lceil \sqrt{\frac{(n-1/2)^2}{4}} + \frac{1}{4} \right\rceil \\
&= \left\lceil \frac{n-1/2}{2} + \frac{1}{4} \right\rceil = \left\lceil \frac{n}{2} \right\rceil .
\end{aligned}$$

□

Gabow and Westermann provided bounds for arboricity and pseudoarboricity by considering matroid partitions.

Theorem 8.4.3 ([Wes88; GW92]). *Let $G = (V, E)$ be a simple graph with $|E| \geq 1$. Then*

$$p(G) \leq \left\lfloor \sqrt{|E|/2 - 7/16} + 3/4 \right\rfloor , \tag{8.10}$$

$$\Gamma(G) \leq \left\lfloor \sqrt{|E|/2 - 7/16} + 5/4 \right\rfloor . \tag{8.11}$$

The arboricity bound was further improved by Dean et al.

Theorem 8.4.4 ([DHS91]). *Let $G = (V, E)$ be a simple graph. Then*

$$\Gamma(G) \leq \left\lceil \sqrt{|E|/2} \right\rceil ,$$

and for every $|E| \geq 0$, there is a graph (V, E) of arboricity $\left\lceil \sqrt{|E|/2} \right\rceil$.

In order to compare bounds, we use the following lemma.

Lemma 8.4.5. *Let $x > 0$. Then for every $\delta \leq x$, we have*

$$\sqrt{x - \delta} \leq \sqrt{x} - \frac{\delta}{2\sqrt{x}}, \tag{8.12}$$

$$\sqrt{x + \delta} \geq \sqrt{x} + \frac{\delta}{4\sqrt{x}}. \tag{8.13}$$

For every $\delta \leq x/4$, we have

$$\sqrt{x-\delta} \geq \sqrt{x} - \frac{\delta}{\sqrt{x}}. \tag{8.14}$$

Proof. Inequality (8.12) is obtained with elementary calculus. It is an easy exercise to show that $\sqrt{x+\delta} \leq \sqrt{x}+\sqrt{\delta}$ for $x, \delta \geq 0$ and $\sqrt{x-\delta} \geq \sqrt{x}-\sqrt{\delta}$ for $x \geq \delta \geq 0$.

For (8.13), we have for $\delta \leq x$:

$$\begin{aligned}\sqrt{x+\delta} \geq \sqrt{x} + \frac{\delta}{2\sqrt{x+\delta}} &\geq \sqrt{x} + \frac{\delta}{2(\sqrt{x}+\sqrt{\delta})} \\ &\geq \sqrt{x} + \frac{\delta}{2(\sqrt{x}+\sqrt{x})} \\ &= \sqrt{x} + \frac{\delta}{4\sqrt{x}}.\end{aligned}$$

For (8.14), we have for $\delta \leq x/4$:

$$\begin{aligned}\sqrt{x-\delta} \geq \sqrt{x} - \frac{\delta}{2\sqrt{x-\delta}} &\geq \sqrt{x} - \frac{\delta}{2(\sqrt{x}-\sqrt{\delta})} \\ &\geq \sqrt{x} - \frac{\delta}{2(\sqrt{x}-\sqrt{x}/2)} \\ &= \sqrt{x} - \frac{\delta}{\sqrt{x}}.\end{aligned}$$

□

Proposition 8.4.6. *For a simple graph $G = (V, E)$ with $|E| \geq 1$, the Gabow–Westermann arboricity bound* (8.11) *never exceeds the bound* (8.9), *and both bounds are at least* $\left\lceil \sqrt{|E|/2} \right\rceil$.

Proof. Let Γ_{GW} denote the right-hand side of (8.11), and let Γ_{d^*} denote the right-hand side of (8.9). The claim is true for $|E| = 1$ by inspection. For $|E| \geq 2$, define $\epsilon := 1/(32\sqrt{|E|/2})$. Let $[\![x]\!] := x - \lfloor x \rfloor$ denote the digits after the decimal point of x. We have

$$\begin{aligned}\Gamma_{d^*} &= \left\lceil \sqrt{|E|/2 + 1/16} + 1/4 \right\rceil \\ &\overset{(8.13)}{\geq} \left\lceil \sqrt{|E|/2} + \frac{1}{64\sqrt{|E|/2}} + 1/4 \right\rceil \\ &= \begin{cases} \left\lfloor \sqrt{|E|/2} \right\rfloor + 2, & \text{if } \left[\!\left[\sqrt{|E|/2} \right]\!\right] + \epsilon/2 > 3/4, \\ \left\lfloor \sqrt{|E|/2} \right\rfloor + 1, & \text{else.} \end{cases}\end{aligned}$$

On the other hand, noting $7\epsilon < 1/4$ we have

$$\begin{aligned}
\Gamma_{GW} &= \left\lfloor \sqrt{|E|/2 - 7/16} + 5/4 \right\rfloor \\
&\overset{(8.12)}{\leq} \left\lfloor \sqrt{|E|/2} - 7\epsilon + 5/4 \right\rfloor \\
&= \begin{cases} \left\lfloor \sqrt{|E|/2} \right\rfloor + 2, & \text{if } \left[\!\left[\sqrt{|E|/2} \right]\!\right] - 7\epsilon \geq 3/4, \\ \left\lfloor \sqrt{|E|/2} \right\rfloor + 1, & \text{else.} \end{cases}
\end{aligned}$$

Since $\left[\!\left[\sqrt{|E|/2} \right]\!\right] - 7\epsilon \geq 3/4$ implies $\left[\!\left[\sqrt{|E|/2} \right]\!\right] + \epsilon/2 > 3/4$, we have $\Gamma_{GW} \leq \Gamma_{d^*}$. That both bounds are at least $\left\lceil \sqrt{|E|/2} \right\rceil$ follows from Theorem 8.4.4, but we can prove it easily. It is obvious that $\Gamma_{d^*} \geq \left\lceil \sqrt{|E|/2} \right\rceil$.

For Γ_{GW}, the claim is verified for $|E| \leq 7$ by exhaustion. For $|E| \geq 8$, we have

$$\begin{aligned}
\Gamma_{GW} &= \left\lfloor \sqrt{|E|/2 - 7/16} + 5/4 \right\rfloor \\
&\overset{(8.14)}{\geq} \left\lfloor \sqrt{|E|/2} - \frac{7}{16\sqrt{|E|/2}} + 5/4 \right\rfloor \\
&\geq \left\lfloor \sqrt{|E|/2} - \frac{7}{32} + 40/32 \right\rfloor \\
&= \left\lfloor \sqrt{|E|/2} + 33/32 \right\rfloor \\
&\geq \left\lceil \sqrt{|E|/2} \right\rceil .
\end{aligned}$$

□

We were unable to prove that Γ_{d^*} never exceeds Γ_{GW} with the bounds of Lemma 8.4.5. However, we also did not find a counterexample in computational experiments.

Chiba and Nishizeki [CN85] show that the arboricity is bounded by $\lceil |V|/2 \rceil$. Kannan et al. [KNR92] improve this to $\lfloor \Delta/2 \rfloor + 1$. Picard and Queyranne [PQ82] observe that $p \leq \lceil (|V|-1)/2 \rceil$. We give an analogous improvement.

Proposition 8.4.7. *For a simple graph G we have*

$$p(G) \leq \left\lceil \frac{\Delta(G)}{2} \right\rceil \text{ and } \Gamma(G) \leq \left\lceil \frac{\Delta(G)+1}{2} \right\rceil .$$

Proof. By Proposition 3.2.3 we know that $d^*(G) \leq \Delta(G)/2$. The first bound follows from Theorem 8.2.3. The second bound follows from $\Gamma(G) \leq \lceil d^* + 1/2 \rceil$. □

Note that these bounds can be asymptotically tighter than the $\sqrt{|E|}$-type bounds if the vertices have similar degrees. On the other hand, if the graph has few vertices with large degree, the $\sqrt{|E|}$-type bounds are tighter.

9 CONVERSION OF PSEUDOFORESTS INTO FORESTS

Far better an approximate answer to the right question [. . .]
than an exact answer to the wrong question[.]

— John W. Tukey, The Future of Data Analysis (1962)

We will further investigate the relationship of forest and pseudoforest partitions in this chapter. The proof of Theorem 8.2.4 can be easily altered in order to show that an *acyclic* d-orientation can be converted into a partition of d forests. Using the greedy 2-approximation algorithm from Chapter 7 (but not the one by Asahiro et al.), this yields a *constructive* 2-approximation algorithm for arboricity, i.e., the corresponding forest partition can be constructed within the same asymptotic runtime. The following theorem shows that it is futile to hope for better approximations via acyclic orientations and converting them into forests.

Theorem 9.0.1 ([Bor+17])**.** *The greedy algorithm finds an acyclic orientation that has the smallest maximum indegree among all acyclic orientations.*

The theorem is easily proved with the following lemma.

Lemma 9.0.2 (Implicit in [Bor+17])**.** *An acyclic orientation $\vec{G}$ has at least one* sink*, a vertex v whose incident edges are all oriented towards v.*

Proof. If there is an isolated vertex in $\vec{G}$, the claim holds. Otherwise, start in some nonisolated vertex u. Follow an arbitrary outgoing edge (u, v) and repeat with v. Once a sink is reached, we are done. If we never reach a sink, then we must eventually reach an already visited vertex because there are finitely many vertices. Then, however, we would have gone in a cycle, a contradiction to the acyclicity of the orientation. □

Proof of Theorem 9.0.1. The orientation produced by the greedy algorithm is acyclic. Let d_{max} denote the maximum indegree assigned during the algorithm. Before the first vertex of degree d_{max} is deleted, all remaining vertices have degree at least d_{max}.

Let $\vec{G}^*$ denote the acyclic orientation with smallest maximum indegree. The restriction of $\vec{G}^*$ on the remaining subgraph is also acyclic, hence it must have a sink by Lemma 9.0.2. Therefore, $\vec{G}^*$ has a vertex of indegree at least d_{max}. This proves optimality of the greedy solution. □

Using the approximation scheme for pseudoarboricity (Section 4.2 via Theorem 8.2.4), we can approximate the arboricity as a *value* within a factor of $(1+\epsilon)$, plus a small additive constant, in time $\mathcal{O}(|E| \log |V| \epsilon^{-1} \log \Gamma)$. This is evident from the fact that $p \leq \Gamma \leq p+1$ (Theorem 8.2.6). In order to get a corresponding forest partition, however, we need a conversion algorithm. In the next section, we will see an optimal conversion.

9.1 CONVERSION BY DIVIDE-AND-CONQUER

Gabow and Westermann [Wes88; GW92] describe how an edge e can be inserted into a forest partition $(F_1, \ldots, F_k)$, if possible. In order to insert it, the algorithm must possibly move other edges between the forests to obtain a forest k-partition of $F_1 \cup \cdots \cup F_k \cup \{e\}$. If this is impossible, it outputs a forest $(k+1)$-partition. The runtime of the algorithm, which is called *cyclic scanning*, is $\mathcal{O}(k|V|)$. We will not discuss its exact workings.

The runtime can be improved to $\mathcal{O}(|E|)$ by the following pre- and postprocessing: Remove all vertices of degree at most k including their edges. The number $|V'|$ of remaining vertices is at most $2|E|/k$, for otherwise the number of edges would exceed $|E|$ by the degree sum formula. Insertions are now performed with cyclic scanning on the remainder graph in $\mathcal{O}(k|V|') \subseteq \mathcal{O}(|E|)$ per edge. In the end, for each removed vertex u, let $(u, v_1), \ldots, (u, v_l)$ for $l \leq k$ be its incident edges. Insert (u, v_i) into F_i. This clearly does not create cycles.

In order to convert k pseudoforests into $k+1$ forests, and k if possible, the authors propose Algorithm 9.1 on the next page. In a nutshell, the algorithm divides the k pseudoforests into two groups of $\lfloor k/2 \rfloor$ and $\lceil k/2 \rceil$ pseudoforests, recursively converts them into $\lfloor k/2 \rfloor + 1 + \lceil k/2 \rceil + 1$ forests, and then inserts the edges of smallest forest into the $k+1$ others. This is always feasible by Theorem 8.2.6. Once the recursive procedure has stopped, one tries to insert the edges of the smallest forest into the k others, which may be feasible or infeasible. It is possible to show that the time of the insertions is bounded by $\mathcal{O}(|E|^2/k)$ in both functions of Algorithm 9.1. Gabow and Westermann pick an arbitrary forest for insertion. They use the fact that it has at most $2|E|/k - 1$ edges due to their preprocessing, but it easier to argue with the forest of minimum cardinality[1]:

In a partition of $k+1$ forests, there is at least one forest that has less than $|E|/k$ edges. This proves the total insertion runtime because each insertion takes linear time. Thus, for some $c > 0$, the following recurrence for runtime T holds:

$$T(|E|, k) \leq T(|E_1|, \lfloor k/2 \rfloor) + T(|E_2|, \lceil k/2 \rceil) + c|E|^2/k,$$
$$T(|E|, 1) \leq c|V|.$$

1 This is done in [Wes88] for `Convert`, but not `Divide`.

Algorithm 9.1: Divide-and-conquer conversion from k pseudoforests to $k+1$ forests by Gabow and Westermann.

```
Input: A pseudoforest partition $E = P_1 \dot{\cup} \cdots \dot{\cup} P_k$ of a simple
       graph $G = (V, E)$.
Output: A forest partition $E = F_1 \dot{\cup} \cdots \dot{\cup} F_l$ with $l = k$ if
        possible and $l = k+1$ otherwise.
function Convert($P_1, \ldots, P_k$):
    $(F_1, \ldots, F_{k+1}) \leftarrow$ Divide($P_1, \ldots, P_k$)
    W.l.o.g. let $F_{k+1}$ be the forest of smallest cardinality
    foreach $e \in F_{k+1}$ do
        if e can be inserted into $(F_1, \ldots, F_k)$ using cyclic scanning
         then
            insert $e$ into $(F_1, \ldots, F_k)$
            $F_{k+1} \leftarrow F_{k+1} \setminus \{e\}$
    if $F_{k+1} = \emptyset$ then
        return $(F_1, \ldots, F_k)$
    else
        return $(F_1, \ldots, F_{k+1})$
function Divide($P_1, \ldots, P_k$):
    if $k = 1$ then
        if $P_1$ is a forest then
            return $(P_1, \emptyset)$
        else
            $M \leftarrow$ select one edge on every cycle in $P_1$
            return $(P_1 \setminus M, M)$
    $(F_1, \ldots, F_{\lfloor k/2 \rfloor + 1}) \leftarrow$ Divide($P_1, \ldots, P_{\lfloor k/2 \rfloor}$)
     // $|E_1| = |F_1 \dot{\cup} \cdots \dot{\cup} F_{\lfloor k/2 \rfloor + 1}|$
    $(F_{\lfloor k/2 \rfloor + 2}, \ldots, F_{k+2}) \leftarrow$ Divide($P_{\lfloor k/2 \rfloor + 1}, \ldots, P_k$)
     // $|E_2| = |F_{\lfloor k/2 \rfloor + 2} \dot{\cup} \cdots \dot{\cup} F_{k+2}|$
    W.l.o.g. let $F_{k+2}$ be the forest of smallest cardinality
    foreach $e \in F_{k+2}$ do
        insert $e$ into $(F_1, \ldots, F_{k+1})$ using cyclic scanning // this
         insertion is always feasible
    return $(F_1, \ldots, F_{k+1})$
```

Without giving a proof, Gabow and Westermann claim that its runtime $T(|E|, k)$ is in $\mathcal{O}(|E|^2/k \log k)$. If we try to prove this by induction with the ansatz

$$T(|E|, k) \leq c'|E|^2/k \log k$$

for some $c' > 0$, we see that we obtain $2c'|E_i|^2/k \log(k/2)$ from each recursive call $T(|E_i|, k/2)$ by the induction hypothesis, as $1/(k/2) = 2/k$. While c' can be made arbitrarily large, it has to be a constant that is valid on all levels of the recursion. Hence, we think that the proof the authors had in mind is incorrect. This may be due to the fact that they wrote $n' = \min(n, 2m/k)$ and thus hid the k from view, and also did not introduce a constant c for the non-recursive term in the recurrence relation. Fortunately, the runtimes stated in [GW92, Table 1] are unaffected.

The analysis of $\mathcal{O}(|V|^2 k \log k)$ in [Wes88; GW92] is correct: We obtain $c'|V|^2 k/2 \log(k/2)$ from each call $T(|E_i|, k/2)$ by the induction hypothesis. However, it is unclear why this estimate was used at all: As $k \geq |E|/|V|$, the runtime $\mathcal{O}(|V||E| \log k)$ that is immediate from Westermann's thesis [Wes88, Equation (1) on page 46] is better and in turn, the alleged runtime $\mathcal{O}(|E|^2/k \log k)$ would have been even better. Let us finish the discussion by stating the recurrence for $\mathcal{O}(|V||E| \log k)$:

$$T(|E|, k) \leq T(|E_1|, \lfloor k/2 \rfloor) + T(|E_2|, \lceil k/2 \rceil) + c|V||E|,$$
$$T(|E|, 1) \leq c|V|.$$

The fact that every forest has less than $|V|$ edges simplifies the discussion, and the runtime analysis by induction is straightforward. We note that instead of inserting the edges one-by-one, one could try using the batch routine of [Wes88; GW92].

Theorem 9.1.1 (Implicit in [Wes88; GW92]). *A pseudoforest k-partition can be converted into a forest $(k+1)$-partition, and a forest k-partition if possible, in $\mathcal{O}(|V||E| \log k)$ time.*

When we apply this conversion after Kowalik's approximation scheme for a constant ϵ, the conversion clearly dominates the runtime. Therefore, Kowalik [Kow06] asked whether a fast conversion of k pseudoforestst into ck forests exists for some $1 < c < 2$. We will answer this in the affirmative in the following section.

From the knowledge of the existence of a pseudoforest k-partition, one can solve the k-forests and $(k+1)$-forests problem from scratch using the algorithms in [Wes88; GW92], which can be faster or slower than $\mathcal{O}(|E||V| \log k)$ depending on $|E|/|V|$ and k. The best general runtime is achieved with Gabow's algorithm for arboricity (Theorem 8.3.3). We will develop a near-linear time algorithm for every k in Chapter 10.

9.2 LINEAR-TIME CONVERSIONS FOR SMALL k

We will describe conversions from k pseudoforests to $k+1$ forests for $k \leq 3$ in this section.

Definition 9.2.1. Let (V, P) be a pseudoforest. For every cycle $C \subseteq P$, select one edge $e_C \in C$ arbitrarily. The set

$$M = \bigcup_{\substack{C \subseteq P \\ C \text{ cycle}}} \{e_C\}$$

is called a *P-matching*.

It is easy to verify that such an M is indeed a matching: Every vertex is in at most one cycle, and there is at most one cycle per connected component.

Lemma 9.2.2. *A pseudoforest (V, P) can be partitioned into a forest and a P-matching in linear time.*

Proof. Determine all cycles in (V, P) in linear time, for example with depth-first search. Arbitrarily select an edge on each cycle to obtain a P-matching M. $P \setminus M$ is a forest. □

The lemma implies that a pseudoforest k-partition can be converted into a forest $2k$-partition in linear time. As the greedy algorithm is a constructive 2-approximation algorithm, this itself is not very useful.

We call an algorithm that turns a pseudoforest k-partition into a forest $\lceil ck \rceil$-partition for some $c > 1$ a *c-conversion*. We will exploit the matching property in the following subsection to obtain a linear-time 5/3-conversion.

9.2.1 *A Linear-Time 5/3-Conversion*

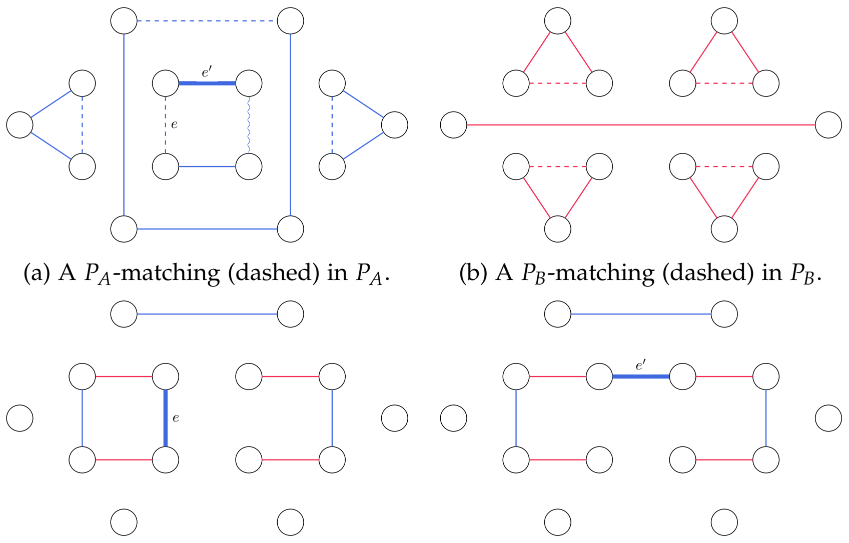
We can employ a lemma by Duncan, Eppstein and Kobourov for a first result.

Lemma 9.2.3 ([DEK04]). *Let G be a simple graph with $\Delta(G) \leq 3$. Then G can be partitioned into two linear forests in linear time.*

Proposition 9.2.4. *A pseudoforest partition (P_1, P_2, P_3) can be converted into a partition of five forests, two of which are linear forests, in linear time.*

Proof. To convert (P_1, P_2, P_3) into five forests, first partition each P_i into a forest F_i and a P_i-matching M_i according to Lemma 9.2.2, which is possible in linear time.

Next, consider the graph on V with edge set $M_1 \cup M_2 \cup M_3$. Clearly, it has maximum degree three. Thus it can be partitioned into two linear forests by Lemma 9.2.3 in linear time. □



(a) A P_A-matching (dashed) in P_A.

(b) A P_B-matching (dashed) in P_B.

(c) The union L of the P_A- and P_B-matchings.

(d) L after an exchange of edges e, e'. It is a linear forest.

Figure 9.1: Converting two pseudoforests into two forests and a linear forest.

This implies that a partition of k pseudoforests can be converted into $\lceil 5k/3 \rceil$ forests in linear time. Better conversion algorithms will be developed in the following subsections, which use the fact that the matching edges are from different connected components in the pseudoforests.

9.2.2 *A Linear-Time 3/2-Conversion*

We now develop a conversion procedure by choosing the edges on the cycles more carefully. The following lemma is trivial, but crucial to all algorithms to follow.

Lemma 9.2.5 ([Gal79]). *Let G be a simple graph with $\Delta(G) \leq 2$. Then every connected component of G is either a path or a cycle.*

Theorem 9.2.6. *There is a linear-time conversion of a pseudoforest partition (P_A, P_B) into a partition of two forests A, B and a linear forest L whose edges are from P_A and P_B alternatingly.*

Proof. Convert P_A and P_B into forests A, B and a P_A-matching M_A and a P_B-matching M_B as in Lemma 9.2.2. Let $L = M_A \cup M_B$. An example can be seen in Figure 9.1abc. Every vertex in L has a degree of at most two. By Lemma 9.2.5, (V, L) can only consist of paths and (even-length) cycles, their edges must be from M_A and M_B alternatingly. Determining all cycles is possible in linear time.

We now modify L by processing the cycles one after another in steps. Consider some cycle $Z \subseteq L$, and pick an arbitrary edge $e \in Z$ (Figure 9.1c). Without loss of generality, let this edge be from P_A.

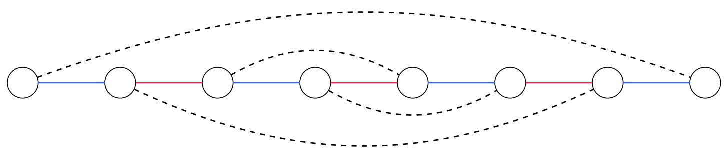


Figure 9.2: Inserting a P_C-matching (dashed edges) into the linear forest L obtained from Theorem 9.2.6 (isolated vertices not shown) could create many interlocked cycles.

Adding it to A would re-create the original cycle $C_e \subseteq P_A$ with $e \in C_e$. Let $e' \in C_e$ be adjacent to e in P_A (Figure 9.1a). Swap e and e'. The modified set $A' = (A \cup \{e\}) \setminus \{e'\}$ is a forest, and the modified set $L' = (L \cup \{e'\}) \setminus \{e\}$ is the union of a P_A- and a P_B-matching and thus still consists of paths and cycles by Lemma 9.2.5.

The edge $e' \neq e$ cannot link two vertices on $Z \setminus \{e\}$ in L' because this would imply a vertex of degree three and thus a contradiction. Therefore, the path $Z \setminus \{e\}$ must have been joined at one of its end-points to another component of L upon insertion of e' (Figure 9.1d). Again, as vertices have degree at most two, and the other components were not affected by the replacement, the vertex e' links to must have been the end of a path before the replacement. Thus the number of cycles in L' is one less than in L.

By breaking up cycles one by one and joining them to paths at their ends, we postprocess L to become a linear forest while maintaining the forest property for A and B. The whole process takes linear time because we only determine cycles once in L. $\square$

Note that exchanging an edge e for a *non-adjacent* edge e' on the original cycle could link two end vertices of the same path in L and thereby create a new cycle. An example is the squiggly edge in Figure 9.1a.

Theorem 9.2.6 implies a conversion into $\lceil 3k/2 \rceil$ forests in linear time. In the next subsection, we will exploit properties of L to improve the conversion ratio further.

9.2.3 *A Linear-Time* 4/3*-Conversion*

In this section, we will show how a pseudoforest 3-partition can be converted into a forest 4-partition in linear time. A first observation is that the linear forest L constructed in Theorem 9.2.6 is size-bounded: A pseudoforest can have at most $|V|/3$ cycles because the smallest unicyclic component is a 'triangle', i.e., the complete graph K_3. Therefore, the linear forest L has at most $2|V|/3$ edges.

If we tried to combine matchings from three pseudoforests into a set S, it would have $|S| \leq |V|$. There are instances where exactly $|V|$ edges are chosen, e.g., three pseudoforests on twelve vertices, each consisting of four triangles. As a forest has at most $|V| - 1$ edges, the set S cannot be a forest then, regardless of which edges we choose on the cycles! In terms of size, a surplus of one edge is not necessarily a problem, as inserting a single edge into a forest partition is possible in linear time [GW92] (see also Section 9.1). However, S could have many interlocked cycles (see Figure 9.2 for an example).[2]

The intuition behind our approach is as follows. For three pseudoforests P_A, P_B, P_C, we try to insert a P_C-matching M into L, which is obtained from P_A and P_B as in Theorem 9.2.6.

The key property we want to exploit is that the M_A-edges of L are from different connected components of P_A, the same holds for M_B-edges with P_B. Hence, if L is too full to insert an edge of M, we can hope to insert it between two components of A or B: If for an edge $(u, v) \in M$, there are two edges incident to u and at least one edge incident to v in L, then (u, v) links two connected components in A or B, or both, depending on which pseudoforest(s) the incident edges come from. Hence inserting (u, v) cannot create a cycle.

It is, however, possible that connected components in A or B become linked in a cycle by several such M-edges. This will be resolved by moving a certain edge of L to C, which allows inserting one carefully chosen M-edge that created the cycle in A or B into L.

As an isolated vertex u can be linked to a tree without creating a cycle, an M-edge with such an endpoint u can always be inserted into L.

The remaining case is where an M-edge links two vertices in L of degree one, i.e., end vertices of paths. The subcase where the adjacent L-edges are from different pseudoforests is problematic, because then the M-edge does not necessarily link different connected components in A or B. In the following lemma, however, we will take care of all M-edges linking end vertices of paths.

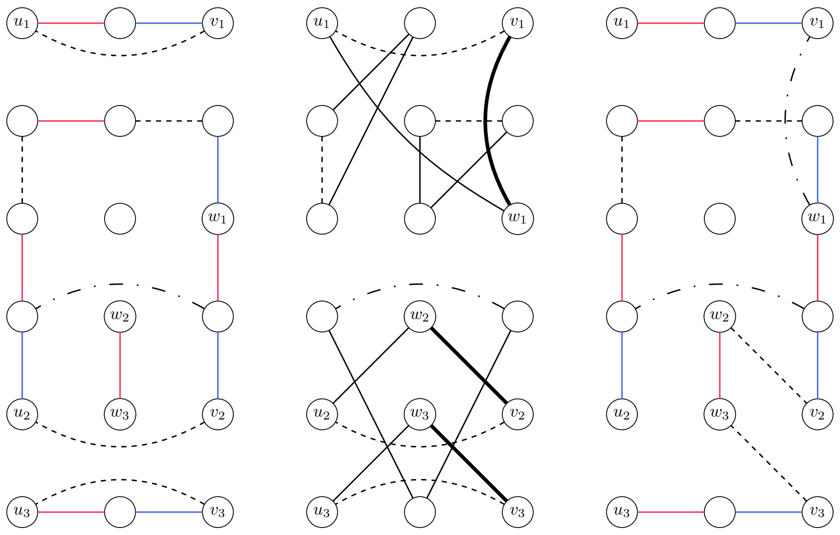
Lemma 9.2.7. *Given a pseudoforest partition (P_A, P_B, P_C), let A, B, L be as in Theorem 9.2.6. Then a P_C-matching M can be computed in linear time such that $(V, L \cup M_1)$ is a linear forest for*

$$M_1 = \{(u, v) \in M \mid \deg_L(u) = 1 = \deg_L(v)\}.$$

Proof. Choose an arbitrary P_C-matching M in linear time. Consider the set M_1 as defined in the theorem. As the degrees in $L \cup M_1$ are bounded by two, its connected components are paths and cycles by Lemma 9.2.5. A cycle can only arise if M_1-edges link paths in a cycle at their end vertices (possibly a single path). An example can be seen in Figure 9.3a on the facing page.

With respect to M_1, the paths of L behave essentially like the vertices of L in Theorem 9.2.6. We modify the choice M. It is possible to detect

2 We note that we tried to utilize proven cases of the Strong Nine Dragon Tree Conjecture (see Subsection 9.2.4) for $d^* < 4/3$ [Mon+12] and $d^* < 3/2$ [Kim+13], to no avail.



(a) The linear forest L with M_1-edges (dashed) and edges of $M \setminus M_1$ (dash-dotted).

(b) The pseudoforest P_C. Adjacent edges selected for an exchange are indicated in bold.

(c) L after the exchanges. It remains a linear forest after adding M_1'-edges (dashed).

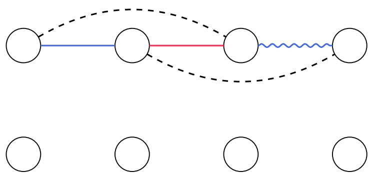
Figure 9.3: Dealing with M_1-edges in the proof of Lemma 9.2.7.

cycles in $L \cup M_1$ in linear time. For each such cycle Z, pick one edge $(u, v) \in M_1 \cap Z$. This edge is from a cycle in P_C. Exchange it with an adjacent edge on the original cycle in P_C, say (v, w) (Figure 9.3bc). This modified set M' is also a P_C-matching. Define M_1' analogously to M_1. We now argue that $L \cup M_1'$ is acyclic and hence a linear forest.

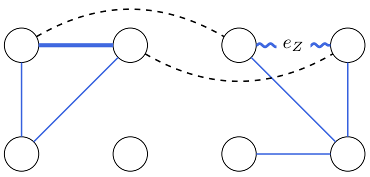
If one or several paths of L have been joined to form a cycle Z in $L \cup M_1$, then one of these paths has one end vertex u that is not incident to any edge of M_1'. Hence, the cycle has been broken into a path of linked-together paths, which is attached at end vertex v to a vertex w of some path, while end vertex u now has no incident M-edge. If w is an end vertex of a path, this path was not part of a cycle, in particular Z. Hence, the paths of Z are linked end-to-end to a sequence of paths, i.e., no new cycle is introduced. If w is an internal vertex of a path (w_1 in Figure 9.3bc), then $(u, w) \notin M_1'$, hence it cannot be part of a cycle in $L \cup M_1'$. (We will shortly see how to deal with such edges.) □

Equipped with Lemma 9.2.7, we can now attack the M-edges that link connected components in A and B.

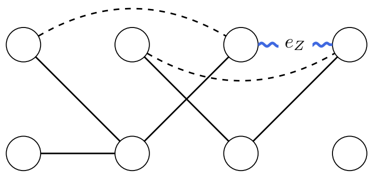
Theorem 9.2.8. *A pseudoforest partition (P_A, P_B, P_C) can be converted into a partition of four forests, one of which has maximum degree at most three, in linear time.*



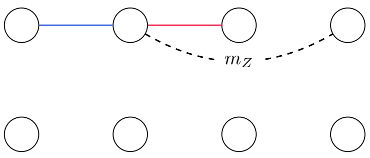
(a) The linear forest L with M_2 superimposed (dashed).



(b) Pseudoforest P_A with M_2^A (dashed). $P_A \cap L$ is shown in bold.



(c) The edge e_Z links two cyclic connected components of P_C, and thus, C.



(d) After removing the e_Z from L, the M_2^A-edge m_Z can be inserted into L.

Figure 9.4: Dealing with M_2-edges in the proof of Theorem 9.2.8.

Proof. Turn (P_A, P_B) into two forests A, B and a linear forest L according to Theorem 9.2.6. Apply Lemma 9.2.7 to obtain the special P_C-matching M. Define $C := P_C \setminus M$ and

$$\begin{aligned} M_0 &:= \{(u,v) \in M \mid \deg_L(u) = 0\}, \\ M_1 &:= \{(u,v) \in M \mid \deg_L(u) = 1 = \deg_L(v)\}, \\ M_2 &:= \{(u,v) \in M \mid \deg_L(u) = 2,\ \deg_L(v) \geq 1\}. \end{aligned}$$

We have $M = M_0 \,\dot\cup\, M_1 \,\dot\cup\, M_2$. We know that $L \cup M_1$ is a linear forest.

Consider the set M_2 (see Figure 9.4a for a running example). As three or four L-edges are adjacent to each $(u,v) \in M_2$, at least two of them must be from the same pseudoforest. We can hence partition M_2 into

$$M_2 = M_2^A \,\dot\cup\, M_2^B$$

such that for every $(u,v) \in M_2^A$, there exist $(t,u), (v,w) \in L \cap P_A$, and likewise for M_2^B. (The partition need not be unique.) The following discussion is analogous for P_B, B and M_2^B.

Every $(u,v) \in M_2^A$ links two different cyclic connected components in P_A and hence two different components in A (Figure 9.4b). Linking occurs only at the endpoints of edges $e \in P_A \setminus A = P_A \cap L$ (indicated in bold in Figure 9.4b).

Therefore, components of A behave like vertices that have at most two incident M_2^A-edges. In this contracted view, there are only paths and cycles according to Lemma 9.2.5. Inside the components, cycles of $A \cup M_2^A$ go through exactly the edges of A that were part of a cycle in P_A. It is possible to determine all cycles of $A \cup M_2^A$ in linear time. For every such cycle Z, consider one arbitrary edge $e_Z \in P_A \cap L$ that 'shortcuts the cycle', i.e., it is an edge chosen from P_A for L that is adjacent to two M_2^A edges (the squiggly line in Figure 9.4b). This implies that e_Z links two cyclic connected components in P_C, and hence two components in C (Figure 9.4c). Let Y_A denote the set of all such edges e_Z (Y_B is analogously defined). The goal is to remove all edges Y_A from L to make room for one M_2^A-edge m_Z on each cycle Z in $A \cup M_2^A$ (Figure 9.4d). By removing one such edge per cycle of $A \cup M_2^A$, its forest property is restored. Let X_A and X_B denote the sets of the m_Z for A and B, respectively. We will later carefully choose the X- and Y-sets such that $(L \cup M_1 \cup X_A \cup X_B) \setminus (Y_A \cup Y_B)$ is acyclic.

Add $Y_A \cup Y_B$ to C and, only for the sake of argument, also to P_C. Thereby, cyclic connected components of P_C are linked via Y_A-edges and Y_B-edges, and these must be incident to the endpoints of the M_2^A-edges.

Claim 9.2.9. *A P_C-component is linked via at most one Y_A-edge in $P_C \cup Y_A \cup Y_B$. Moreover, if it is linked via a Y_A-edge, then it is not linked via a Y_B-edge. The claim holds analogously with the roles of Y_A and Y_B reversed.*

Proof of Claim 9.2.9. If a component in P_C became linked by two different edges of Y_A, then these two edges would share an adjacent M_2^A-edge. Hence, they would shortcut the same cycle in $P_A \cup M_2^A$. This is a contradiction to the selection of exactly one edge e_Z on such a cycle. The 'moreover'-part of the claim follows from $M_2^A \cap M_2^B = \emptyset$. □

This implies that every component of C is isolated or linked to a single other component in $C \cup Y_A \cup Y_B$. As the components are trees, the set $C \cup Y_A \cup Y_B$ is acyclic for any specific choice of Y_A and Y_B.

We now choose which edges $X_A \subseteq M_2^A$ are inserted into $L \cup M_1$, and which edges $Y_A \subseteq A$ are removed from L.

For every cycle Z of $A \cup M_2^A$, consider the endpoints of the M_2^A-edges. If we remove an edge $e_Z \in P_A \cap L$ from $L \cup M_1$, the path disconnects into two paths (trees). If e_Z has an endpoint u_Z whose degree in L is one, the M_2^A-edge incident to u_Z can be inserted into $(L \setminus \{e_Z\}) \cup M_1$. If both endpoints have degree two in L, it is possible that adding either of the two incident M_2^A-edges on Z to $L \cup M_1$ creates a cycle. We would need to choose an M_2^A-edge on Z that 'bridges the gap', i.e., that connects the two different trees.

We describe a simple general way of choosing an edge e_Z together with an adjacent M_2^A-edge that also allows for a simple analysis of acyclicity: Number the vertices from 1 to n such that every path of $L \cup M_1$ consists of a contiguous segment of the sequence $(1, \ldots, n)$. In other words, the paths are arranged in a sequence from left to right. This is possible in linear time. We view edges (u, v) ordered as $u < v$.

Among the edges $(u, v) \in P_A \cap L$ that shortcut a cycle Z in $P_A \cup M_2^A$, choose 'the rightmost' as e_Z, i.e., the one that maximizes v. One of the two adjacent M_2^A-edges is $m_Z = (t, v)$ with $t < u$, which we add to $(L \setminus \{e_Z\} \cup M_1)$ (these are the choices in Figure 9.4 when the path is ordered from left to right). These edges can be determined in linear time in total by scanning each Z once for the rightmost shortcut edge, and selecting the appropriate adjacent M_2^A-edge. We now prove that performing all these deletions and insertions does not create a cycle.

Claim 9.2.10. *For the above specific choices of X_A, Y_A, X_B and Y_B, $(L \cup M_1 \cup X_A \cup X_B) \setminus (Y_A \cup Y_B)$ is a forest.*

Proof of Claim 9.2.10. We order the edges $e_Z = (u, v)$ in $Y_A \cup Y_B$ by their right endpoint v, and imagine the process of deleting them from $L \cup M_1$ and adding their adjacent edge $m_Z = (t, v) \in X_A \cup X_B$ in order of decreasing v ('from right to left').

We prove by induction on $i \geq 0$ that after the i-th deletion of $e_Z = (u, v)$ and insertion of $m_Z = (t, v)$, the graph is a forest. For every i, let $M_2^i \subseteq M_2$ denote the edges of $X_A \cup X_B$ inserted in iterations $1, \ldots, i$, and let L^i denote L without the edges of $Y_A \cup Y_B$ removed in these iterations. Before the first insertion and deletion, $L \cup M_1$ is a linear forest ($i = 0$).

Let the induction hypothesis hold for some $i \geq 0$. After deleting the $(i+1)$-th edge $e_Z = (u, v)$, the tree of $L^i \cup M_1 \cup M_2^i$ that e_Z was a part of becomes disconnected into two different trees, one of which contains u and the other v. The edge $m_Z = (t, v)$ has $t < u$. We have to show that inserting m_Z would not create a cycle. This could only happen if t were in the same tree as v. Assume this were the case. Then there is a unique path $P \subseteq (L^i \setminus \{e_Z\}) \cup M_1 \cup M_2^i$ from v to t. Note that the first edge must be from L, and no two consecutive edges of this path can be from $M_1 \cup M_2^i$ because it is a matching. (In particular, v cannot be isolated now.) Recall that we ordered the paths including the M_1-edges. As we deleted (u, v), P must pass through at least one edge $e = (x, y) \in M_2^i$ with $v < y$. Follow the path from v to t until the e with maximum y is visited. By construction, its left incident edge $(y-1, y)$ that was from L originally was deleted. Hence, there must be an $(x', y') \in M_2^i$ on P with $v < y < y'$ in order to reach $t < v$. This is a contradiction to y being maximum. $\square$

Note that $(L \cup M_1 \cup X_A \cup X_B) \setminus (Y_A \cup Y_B)$ may have vertices of degree three.

Lastly, we consider the set M_0. Clearly, an isolated vertex u can be linked to a tree of $(L \cup M_1 \cup X_A \cup X_B) \setminus (Y_A \cup Y_B)$ via $(u, v) \in M_0$ without creating a cycle. (This may also cause vertices of degree three.) As $M = M_0 \dot\cup M_1 \dot\cup M_2$, this concludes the proof. $\square$

Theorem 9.2.11. *Let G be a simple graph. A partition of G into k pseudoforests can be converted into a partition of $\lceil 4k/3 \rceil$ forests in linear time.*

Proof. Divide the pseudoforests into $\lfloor k/3 \rfloor$ triplets and convert each triplet into four forests as in Theorem 9.2.8. If k is divisible by three, the claim follows. If $k \equiv 1 \mod 3$, we convert the remaining pseudoforest into two forests. If $k \equiv 2 \mod 3$, we convert the two pseudoforests into three forests according to Theorem 9.2.6. The claim follows. □

We can now give a near-linear time algorithm with an approximation factor of $(4/3 + \epsilon)$ with a small additive constant due to rounding.

Theorem 9.2.12. *For every fixed $\epsilon > 0$, a simple graph $G = (V, E)$ can be partitioned into at most*

$$\lceil 4/3 \cdot \lceil (1+\epsilon) d^*(G) \rceil \rceil$$

forests in $\mathcal{O}(|E| \log |V|)$ time.

Proof. Use Theorem 7.3.1 to obtain the approximate pseudoforest partition for $\epsilon > 0$. Then apply Theorem 9.2.11. □

It seems difficult to obtain a linear-time conversion from four pseudoforests to five forests because we lost linearity of the forest in the proof of Theorem 9.2.8 and moved edges between A, B and C. As a follow-up to our result, Fischer[3] suggested that the approach can be generalized to k forests with a runtime of $\mathcal{O}(|E| k^2 \alpha(|V|))$ (see Section 2.3 for the definition of α, which can be regarded as constant for all practical purposes). This claim would imply a constructive approximation scheme for the arboricity.

In a combined effort, we elaborated on Fischer's roadmap. This indeed lead to a constructive approximation scheme for the arboricity, which will be the subject of the next chapter. In fact, the runtime of the conversion is better than what was hoped for.

9.2.4 *The Nine Dragon Tree Theorem*

We note the following pattern in the results of the previous sections: One of the constructed forests, say F_{k+1}, has $\Delta(F_{k+1}) \leq 1$ in the conversion from one pseudoforest, $\Delta(F_{k+1}) \leq 2$ from two pseudoforests, and $\Delta(F_{k+1}) \leq 3$ from three pseudoforests. Our (slightly slower) algorithm in the next chapter for any $k \in \mathbb{N}_0$ even outputs a forest $(k+1)$-partition where the sizes of the connected components in F_{k+1} are at most k. These observations may be related to a known result in graph theory.

Imagine a graph G has $\gamma(G) = 5.9$ and some other graph H (e.g., a proper subgraph of G) has $\gamma(H) = 5.1$, then $\Gamma(G) = 6 = \Gamma(H)$. Intuitively speaking, one may wonder if it is possible to identify one forest in the partitions with the fractional part of γ: The forest number F_Γ in the partition of H could be more 'restricted' than in the partition of G. Montassier et al. [Mon+12] conjectured the following 'Nine Dragon Tree Conjecture', of which they, Kim et al. [Kim+13] and Chen et al. [Che+17] proved special cases. The full conjecture was proved only recently by Jiang and Yang.

3 Frank Fischer, personal communication, November 2018.

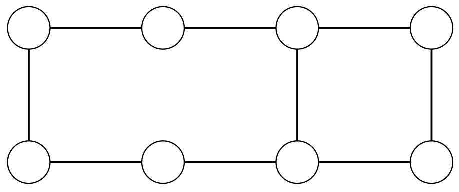


Figure 9.5: A graph with $d^* > 1$ and $\gamma \leq 1 + \frac{1}{3}$.

Theorem 9.2.13 (Nine Dragon Tree Theorem, [JY17]). *Let G be a simple graph. Let $k, d \in \mathbb{N}$. If*

$$\gamma(G) \leq k + \frac{d}{k+d+1},$$

then G can be partitioned into $k+1$ forests, one of which has maximum degree d.

The resemblance to the degree bound of our construction is striking for $d = k$. Our algorithm constructs $p + 1 = \lceil d^* \rceil + 1$ forests from a pseudoforest p-partition (unless all pseudoforests are forests), so $\Gamma = p + 1$ would be a necessary condition to prove the theorem for $d = k$ via our construction. One might think from Lemma 8.2.9 and Corollary 8.2.10 that if $k \leq \gamma \leq k + \frac{k}{2k+1}$, i.e., the fractional part is small, then $d^* \leq k$ and thus $p = \lceil d^* \rceil \leq k$ as desired.

Unfortunately, there are graphs where $\gamma \leq k + \frac{k}{2k+1}$, yet $\Gamma = p$. As an example, consider the graph in Figure 9.5. While it has $d^* = 9/8$, and hence $p = 2$, it has $\gamma = 4/3 \leq 1 + \frac{1}{1+1+1}$, and hence $\Gamma = 2$. Of course, it could be the case that such counterexamples only exist for certain γ, or that our algorithm can be improved.

Fan et al. prove the following analog to the Nine Dragon Tree Theorem for the maximum density.

Theorem 9.2.14 ([Fan+15]). *Let G be a simple graph. Let $k, d \in \mathbb{N}$. If*

$$d^*(G) \leq k + \frac{d}{k+d+1},$$

then G can be partitioned into $k+1$ pseudoforests, one of which has maximum degree d.

Montassier et al. [Mon+12] give a stronger variant of the Nine Dragon Tree conjecture.

Conjecture 9.2.15 (Strong Nine Dragon Tree Conjecture). *Let G be a simple graph. Let $k, d \in \mathbb{N}$. If*

$$\gamma(G) \leq k + \frac{d}{k+d+1},$$

then G can be decomposed into $k+1$ forests, one of which has at most d edges in every connected component.

This reduces to the Nine Dragon Tree Theorem for $k = d = 1$. The special case $k = 1$, $d = 2$ is proved by Kim et al. [Kim+13].

9.2.5 *An Application to Planar Graphs*

Schnyder [Sch90] and Chrobak and Eppstein [CE91] show that a planar graph can be partitioned into three forests[4] in $\mathcal{O}(|V|)$ time from an embedding of the graph into the plane (which can also be computed in linear time, see [HT74]). The algorithm of Grossi and Lodi [GL98] finds, also using an embedding, a partition into three forests in time $\mathcal{O}(|V| \log |V|)$, and four forests in $\mathcal{O}(|V|)$. By using the second 3-orientation algorithm of [CE91] and converting it to a pseudoforest 3-partition (see Theorem 8.2.4), we can obtain four forests in linear time by applying Theorem 9.2.11 *without computing an embedding first*. Note that there are planar graphs with pseudoarboricity three. Chrobak and Eppstein [CE91] also note that an acyclic 5-orientation can be computed in linear time, which is in fact achieved by the greedy algorithm (Section 7.1) because every planar graph has a vertex of degree at most five.

9.2.6 *Partitioning a Planar Graph into Three Forests*

In this subsection, we will review the linear-time algorithm of Chrobak and Eppstein [CE91] for partitioning a planar graph into three forests. The algorithm was stated as a 3-orientation algorithm, its forest partitioning capability was only given as a remark. We now give a full proof adapted to forests.

The vertices that lie on the outer face of the graph are called *external vertices*. We denote the membership to the three forests by colors red, blue and black.

Theorem 9.2.16 ([Sch90; CE91]). *Let $G = (V, E)$ be a planar graph. Then G can be partitioned into three forests in linear time.*

Proof. Without loss of generality, we assume G is connected. Find an embedding of the graph into the plane in linear time [HT74]. We will prove the following claim by induction on the number of vertices, which yields the theorem.

Claim. *The edges of G can be partitioned into the red, blue, and black forests such that no two external vertices are in the same tree in the black forest.*

The claim obviously holds for a single vertex. Now let $|V| \geq 2$. There is at least one external vertex that has at most two external neighbors. Let u be such an external vertex. By the induction hypothesis, $G' = G[V \setminus \{u\}]$ has a coloring such that no two external vertices are in the same tree in the black forest.

4 That a planar graph has arboricity at most three can be proved from Euler's formula [CN85; Wes88]. Gonçalves proves that one of the forests can be restricted to maximum degree four [Gon09], and hence a partition into two forests and two linear forests is possible.

Let v and (if present) w denote the external neighbors of u in G. Color the edge uv red and uw blue. This amounts to linking the vertex u to a single tree in the red and (possibly) blue forests, hence they remain forests. Consider the external vertices $v_1, \dots, v_k$ ($k \geq 0$) of G' that are not external in G and connected to u in G via edges $(u, v_1), \dots, (u, v_k)$. Color these edges black. As the $v_1, \dots, v_k$ were in different trees of the black forest, these are joined to a single larger tree via u.

As v and w were not in the same tree of the black forest, this is still the case, and they are also not in the black tree u is contained in now. Any external vertex $t \notin \{v, w\}$ of G' that is also external in G has not been touched and hence the claim holds for G.

The proof is easily converted into a recursive algorithm. Its runtime analysis is identical to the first 3-orientation algorithm of [CE91]. □

An example can be seen in Figure 1.3b on page 3, where the algorithm derived from above proof starts at the top vertex and proceeds counterclockwise.

We note that if the graph can be partitioned into two forests, the algorithm may return three forests. This is the case for the complete graph K_4. It is an interesting open question how a partition into two forests can be determined in linear time, if possible. The 3-partitioning algorithm of Grossi and Lodi [GL98] also does not seem to achieve this.[5]

5 Roberto Grossi, personal communication, December 2018.

10 A CONSTRUCTIVE ARBORICITY APPROXIMATION SCHEME

In this chapter, we will devise a constructive approximation scheme for the arboricity problem.

Theorem 10.0.1. *For every $\epsilon > 0$, a simple graph can be partitioned into at most $\lceil (1+\epsilon) \cdot \lceil (1+\epsilon) d^* \rceil \rceil$ forests in $\mathcal{O}(|E| \log |V| \log \Gamma \, \epsilon^{-1})$ time. Furthermore, if ϵ is fixed, the runtime can be bounded as $\mathcal{O}(|E| \log |V|)$.*

The proof uses generalizations of the ideas from the previous chapter for a fast conversion of k pseudoforests into $k+1$ forests: Lemmata 10.1.1, 10.1.2 and 10.2.1, which are due to Fischer.[1] Our conversion also implies a near-exact arboricity algorithm whose runtime scales with Γ.

Theorem 10.0.2. *A simple graph can be partitioned into at most $\Gamma + 2$ forests in $\mathcal{O}(|E| \log |V| \, \Gamma \log^* \Gamma)$ time.*

10.1 THE SURPLUS GRAPH

Throughout the remainder of the chapter, we maintain the edges E of the graph as a partition $E = F \mathbin{\dot\cup} M$, where $F = F_1 \mathbin{\dot\cup} \cdots \mathbin{\dot\cup} F_k$ for forests $F_1, \ldots, F_k$ and $M = M_1 \mathbin{\dot\cup} \cdots \mathbin{\dot\cup} M_k$ such that $F_i \cup M_i = P_i$ is a pseudoforest and M_i is a P_i-matching for $i = 1, \ldots, k$. We call (F, M) a *valid partition* of the graph and $H = (V, M)$ its *surplus graph*. Initially, a valid partition is obtained by applying Lemma 9.2.2 to each P_i of a given pseudoforest k-partition. Edges in both F_i and M_i are considered to have color i. Note that any two adjacent edges of H must have different colors. By turning H into a forest, i.e., F_{k+1}, we will give a constructive proof of Theorem 8.2.6.

We will use a *swap operation* in order to move edges from H to the forests $F_1, \ldots, F_k$, it is described in the following lemma and illustrated in Figure 10.1 on the following page.

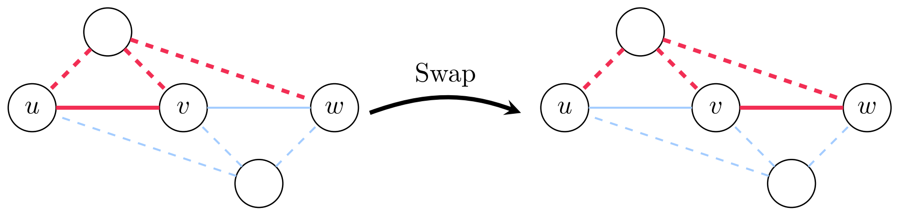
Lemma 10.1.1. *Let (F, M) be a valid partition of a simple graph G, and let $H = (V, M)$ be its surplus graph. Let $(u, v) \in M$ with color i and $(v, w) \in M$ with color $j \neq i$. Then one of the following applies.*

1. *We may swap the colors of (u, v) and (v, w) in H, i.e., modify*

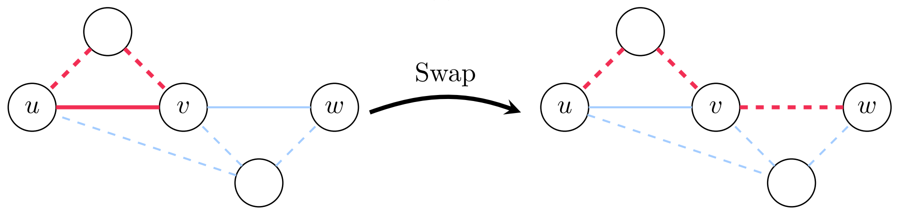
$$\begin{aligned} M_i &\leftarrow M_i \setminus \{(u,v)\} \cup \{(v,w)\}, \\ M_j &\leftarrow M_j \setminus \{(v,w)\} \cup \{(u,v)\}, \end{aligned}$$

   such that (F, M) is still valid.

[1] Frank Fischer, personal communication, December 2018.



(a)



(b)

Figure 10.1: (a) Situation 1. of Lemma 10.1.1. The colors i (red, thick) and j (blue, thin) of the edges (u, v) and (v, w) are swapped. Dotted edges represent the forests F_i and F_j. (b) Situation 2. of Lemma 10.1.1. v and w are in different trees of forest F_i (red, thick, dotted), hence the edge (v, w) can be inserted into it after swapping colors.

2. *We may assign color j to (u, v) in H and insert (v, w) into F_i, i.e., modify*

$$M_j \leftarrow M_j \setminus \{(v, w)\} \cup \{(u, v)\},$$
$$F_i \leftarrow F_i \cup \{(v, w)\},$$

such that (F, M) is still valid.

3. *Symmetrically to 2., we may assign color i to (v, w) in H and insert (u, v) into F_j such that (F, M) is still valid.*

4. *We may insert (v, w) into F_i and (u, v) into F_j such that (F, M) is still valid.*

Furthermore, if there is an edge $(w, x) \in M$ of color i, then 2. or 4. applies.

Proof. Since (F, M) is valid, u and v are in the same tree in F_i and v and w are in the same tree in F_j. Let us swap the colors of (u, v) and (v, w) in H, i.e., modify M_i and M_j accordingly. We distinguish several cases:

1. If u and v are in the same tree of F_j, and v and w are in the same tree of F_i, then after swapping the colors of (u, v) and (v, w), M_i and M_j are still P_i- and P_j-matchings, respectively. This is illustrated in Figure 10.1a.

2. If v and w are in different trees in F_i, then $F_i \cup \{(v, w)\}$ is a forest. If u and v are in the same tree in F_j, change the color of (u, v) to j, now no edge in M_i exists whose endpoints are both in the tree

of F_i that v is contained in. Since (F, M) had been valid, there is at most one edge in M_i whose endpoints are both in the tree of F_i that w is contained in. Hence, after inserting (v, w) into F_i, there is still at most one such edge for the joined tree. This is illustrated in Figure 10.1b.

3. If v and w are in the same tree in F_i, and u and v are in different trees in F_j, we have a case that is symmetric to 2.

4. If v and w are in different trees in F_i, and u and v are in different trees in F_j, then we can insert (v, w) into F_i and (u, v) into F_i. (F, M) is easily seen to be valid.

For the 'furthermore'-claim, we observe that if $(w, x) \in M_i$, then v and w must be in different trees of F_i because (F, M) is valid. □

Lemma 10.1.2. *Let a path in the surplus graph H be given by a sequence of distinct edges $(e_1, \ldots, e_l)$ where $e_1, e_l \in M_i$ for the same color i. Then we can modify (F, M) such that the cardinality of M decreases.*

Proof. We can move the color i from e_1 towards e_l in a sequence of swaps using Lemma 10.1.1, i.e., swap the colors of e_t and e_{t+1} for $t = 1, \ldots$ until one of the cases 2.-4. of Lemma 10.1.1 applies. This happens at the latest when e_{l-2} has color i, because then we are in the 'furthermore'-part of Lemma 10.1.1. Thus we can move some edge on the path from M to F. □

A connected component of the surplus graph H is called *colorful* if every color appears at most once in it. A surplus graph is called colorful if all its connected components are colorful. Note that each component in a colorful surplus graph has at most k edges. We will exploit this for the runtime analyses in the following sections.

We can implement the swap operation with k union-find data structures that keep track of the vertex sets of the connected components of each F_i. An edge $(u, v) \in H$ connects two different trees in F_i if and only if the sets $S_u = \text{find}(u)$ and $S_v = \text{find}(v)$ in the union-find structure of F_i are different. When (u, v) is to be inserted into F_i, we call $\text{union}(S_u, S_v)$ in order to merge S_u and S_v.

Since we will be performing $\mathcal{O}(|E|k)$ find operations, but only $\mathcal{O}(|E|)$ union operations, we will use the union-find data structure of [AHU74, Theorem 4.3]. In our scenario, this is better than the well-known union-find implementation where both find and union run in $\mathcal{O}(\alpha(|V|))$ amortized time [Tar75] (this is optimal, see [Tar79; Ban80] and [FS89]).

The data structure uses a list representation of the sets that grants constant worst-case time for each find operation. A union is performed by moving the elements of the shorter list to the longer, which implies that each such element is moved at most $\log(|V|)$ times because the resulting list is at least twice as long as the shorter list. This implies that up to $|V| - 1$ unions can be performed in time $\mathcal{O}(|V| \log |V|)$ [AHU74]. However, if we perform $u < |V| - 1$ unions in a forest, then a bound of $\mathcal{O}(u \log |V|)$ does not hold[2], hence we need a modification.

Lemma 10.1.3. *Given a forest partition $F = (F_1, \dots, F_k)$ of $G = (V, E)$, it is possible to perform f find and u union operations in F within a runtime of $\mathcal{O}(k|V| + |E| + f + u \log |V|)$.*

Proof. Determine the connected components of each F_i. Assign a unique label to each such component. We create k union-find data structures that represent the components of the F_i with these labels after contracting them, all this is possible in $\mathcal{O}(k|V| + |E|)$. Let $|V_i|$ denote the number of contracted components in F_i. For $v \in V$, we record the label of its connected component in each F_i in an array of size $|V| \times k$ in order to be able to look up the labels in constant time when performing find and union operations. We will implicity assume this in the remainder of the chapter.

After performing u union operations, we can easily see that every component of size $s \in \mathbb{N}$ in an F_i was created by $s - 1$ union operations. Let l denote the number of non-singleton components in all $F_1, \dots, F_k$. The sum of the sizes of all such components equals $u + l \leq 2u$. Since each element is moved at most $\log(|V_i|) \leq \log(|V|)$ times, the total cost of these elements in union operations is bounded by $\mathcal{O}(u \log |V|)$. Obviously, find operations still take constant time. $\square$

Lemma 10.1.4. *A colorful surplus graph can be obtained within a runtime of $\mathcal{O}(|E|k + |E| \log |V|)$.*

Proof. Obtain an arbitrary surplus graph H in linear time, and apply Lemma 10.1.3. First we need to identify duplicate colors in the components of H. The following simplified algorithm is due to Althaus.[3] We identify a duplicate color in a connected component of H by performing a depth-first search in it: Record the colors encountered in the search in a Boolean array of size k. If there is a duplicate color i, we will encounter one such color and recognize it after at most $k + 1$ steps of the DFS (without backtracking steps). Otherwise, the search is unsuccessful and the component already is colorful. The number of unsuccessful searches is at most $|V|$.

We can now apply Lemma 10.1.2 to a path of length at most $k + 1$ from the edge of color i encountered first to the edge of color i encountered second. We charge the costs of the $\mathcal{O}(k)$ find and at most two union operations to some edge that was removed in the swap sequence and start the next search. We perform the searches in each connected component of H until all duplicate colors have

2 Note that we start with already given forests. If a forest has $\Theta(\sqrt{|V|})$ components of size $\Theta(\sqrt{|V|})$, then inserting $\Theta(\sqrt{|V|})$ edges could incur costs of $\Theta(|V| \log |V|)$. The trivial bound of $\mathcal{O}(k|V| \log |V|)$ for all union operations can be improved to $\mathcal{O}(k|V|\alpha(i, |V|))$ for every fixed i with the data structure of La Poutré [Pou90].

3 Ernst Althaus, personal communication, February 2019.

been eliminated. Note that components may disconnect during the algorithm. There are at most $|E|$ successful searches. The total cost is thus $\mathcal{O}(|E|k + |E| \log |V|)$. $\square$

Because the union-find data structures in the above proof do not store the edges that we insert, we store them and their colors separately in a list so we can construct the forests F_i in the next algorithm.

10.2 EXCHANGING EDGES ON CYCLES

In order to remove cycles from a colorful surplus graph H, we want to replace an edge e in H that is on some cycle in a connected component C with an edge from some F_i that goes to a vertex outside of C. This reduces the number of edges that are on at least one cycle in H. After at most $|E|$ such operations, H will be a forest. To do so, we will insert e into F_i, and take an adjacent edge on the resulting cycle instead. We call this the *cycle exchange*.

First, we store the (uncontracted) forests $F_1, \dots, F_k$ in k link-cut tree data structures [ST81; ST83] in total time $\mathcal{O}(|V|k + |E| \log |V|)$. In these structures, each tree is considered to be a rooted tree (which is stored in a compressed way) with all edges oriented towards the root, and the root of the tree containing vertex u can be accessed via root(u). There is an operation evert(u) that makes u the root of its tree. The operation cut(u) deletes the parent edge of u and thereby splits the tree. There is an operation link(u,v), where u is a root and v is in a different tree than u, that makes u point to v. All these operations can be performed in $\mathcal{O}(\log |V|)$ amortized time (in fact, worst-case time [ST83]).[4]

We will also maintain the union-find structures. The reason for this is that the vertex sets of the connected components of an F_i do not change in the cycle exchange. Thus, the union-find structures still work correctly. It is advantageous to simultaneously keep the union-find structure for the faster find runtime.

An edge suitable for the cycle exchange can always be found if H is cyclic. How to determine this edge fast will be shown in the next section.

Lemma 10.2.1. *Let $H = (V, M)$ be a surplus graph, and let $C = (V_C, E_C)$ be a colorful cyclic connected component of H. For any $v \in V_C$, there is a color i such that there is an edge of color i in E_C, and v has no neighbors in F_i that are in V_C.*

Proof. Since C is colorful, exactly $|E_C|$ different colors $c_1, \dots, c_{|E_C|}$ appear in C. As C is cyclic, we have $|E_C| \geq |V_C|$. If v had a neighbor

4 Note that link-cut trees optimally solve the fully dynamic connectivity problem on forests: there is no data structure that supports insertion and deletion of edges both in time $o(\log |V|)$, and this holds even with randomization and amortization in the cell-probe model [PD06].

among the vertices V_C in every F_i, $i = c_1, \dots, c_{|E_C|}$, then v would have at least $|V_C|$ neighbors among V_C in G, a contradiction. □

We will show in the next section how such an edge can be determined efficiently.

Lemma 10.2.2. *Let H be a colorful surplus graph. If for a vertex v on a cycle in H a color i as in Lemma 10.2.1 can be determined in time $T(k, |V|, |E|)$ with $P(k, |V|, |E|)$ preprocessing time, then we can obtain an acyclic colorful surplus graph in time*

$$\mathcal{O}(|E|(T(k, |V|, |E|) + k + \log |V|) + P(k, |V|, |E|)).$$

Proof. We start from a colorful surplus graph H. We can determine if a connected component C of H is acyclic in time $\mathcal{O}(k)$ with DFS. If it is, there is nothing to be done with it. Otherwise, let (u, v) be an edge on a cycle. Determine the color i as in Lemma 10.2.1 in time $T(k, |V|, |E|)$.

Determine a path from the edge of color i in C to v. As in the proof of Lemma 10.1.4, move i towards (u, v) in a sequence of swaps. If an edge is removed from H by this, we charge the costs including the $\mathcal{O}(k)$ find and at most two union operations to the removed edge. We then start looking for cycles again. There can be at most $|E|$ such removals in H in total.

If no edge is removed from H, then v is now incident to an edge (u, v) of color i in H. Since (F, M) is valid, we know that inserting (u, v) into F_i would create a cycle. Make u the root of its link-cut tree in F_i by calling evert(u), i.e., the link-cut tree represents the tree where all edges are directed towards u. If (u, v) were to be inserted into this tree, then it would create a cycle that passes through u and v. Call parent(v) to obtain an edge (v, w) on this cycle incident to v. By the choice of i, $w \notin V_C$, i.e., the edge must leave C in H. Call cut(v) to remove the edge from the link-cut tree, which breaks it into a tree rooted at u and the subtree rooted at v. Call link(u, v) to insert the edge (u, v) into the link-cut tree. As remarked earlier, the union-find structure still represents the trees of F_i after these changes. The number of edges in H that are on at least one cycle decreases, which happens at most $|E|$ times, so the costs for all cycle exchanges amount to $\mathcal{O}(|E| \log |V|)$ in total.

$C \setminus \{(u, v)\}$ is joined to another colorful connected component of H via (v, w). We detect and remove duplicate colors in the resulting component of size $\mathcal{O}(k)$ as we did in the proof of Lemma 10.1.4 in order to keep H colorful. We charge costs to the decrease of $|M|$ rather than the edges themselves because edges can re-enter M in cycle exchanges. If no duplicate color is found, we charge the cost of $\mathcal{O}(k)$ to the cycle exchange. □

We have now obtained an algorithmic proof of Theorem 8.2.6. In addition, each connected component of F_{k+1} has at most k edges.

10.3 FINDING THE EXCHANGE EDGE

We will describe two ways of finding the exchange edge with a runtime that does not depend on $|V|$. The first approach uses the dynamic data structure of Brodal and Fagerberg [BF99], which stores a graph of arboricity at most k and hence can be used for $F_1 \cup \cdots \cup F_k$. In its more elaborate variant that uses balanced search trees (see Section 4 of [BF99]) for storing adjacencies, it allows querying whether two vertices are adjacent in time $\mathcal{O}(\log k)$, inserting an edge in $\mathcal{O}(\log k)$ amortized time, and deleting an edge in $\mathcal{O}(\log |V|)$ amortized time. The structure can be built for a given graph in $\mathcal{O}(|E| \log |V| + |V|)$ time. In fact, $\mathcal{O}(|E| + |V|)$ is possible.[5]

The representation used by the data structure is an orientation of the graph such that every vertex has indegree at most $4k$. Every edge is stored only once, namely in the adjacency list/balanced search tree of the vertex it points to. Hence, the size of each list/search tree is $\mathcal{O}(k)$. We can store the current color of each edge with it without affecting the runtimes.

Lemma 10.3.1. *In the situation of Lemma 10.2.1, we can determine the exchange edge in time $\mathcal{O}(k \log k)$ using the data structure of Brodal and Fagerberg with $\mathcal{O}(|V| + |E|)$ preprocessing time. All other operations have the same asymptotic complexity as in Lemma 10.2.2.*

Proof. Create the data structure for $F_1 \cup \cdots \cup F_k$ in time $\mathcal{O}(|V| + |E|)$ (see the remark at the end of Section 10.1). When looking for a cycle in a component $C = (V_C, E_C)$ of the colorful surplus graph (with $|V_C| \leq k + 1$), we use a Boolean array of size k to mark the colors of the component and remember the respective edges. When some vertex v on a cycle has been determined, we test for each $u \in V_C \setminus \{v\}$ whether $(u, v) \in E \setminus M$ in $\mathcal{O}(\log k)$ with the color-augmented Brodal-Fagerberg data structure. If the edge is present in some F_i, then we obtain color i from the data structure and unmark it in the Boolean array. Once all $u \in V_C \setminus \{v\}$ have been tested, search for a color i that is still marked: the attached edge is the one we were looking for, i.e., v has no neighbors in V_C in F_i. All these operations cost $\mathcal{O}(k \log k)$ in total.

During the cycle exchange algorithm in Lemma 10.2.2, at most $|E|$ edges are inserted into the forests $F_1, \ldots, F_k$. An edge is only deleted in a cycle exchange, which happens at most $|E|$ times. Thus, these cost amount to $\mathcal{O}(|E|)$ insertions and deletions in the data structure, and each such operation costs $\mathcal{O}(\log |V|)$ amortized time. Hence, the runtime of Lemma 10.2.2 can indeed be achieved. □

We now prove Theorems 10.0.1 and 10.0.2.

5 Recall from the proof of Theorem 4.3.1 that the adjacency lists of a graph can be sorted in linear time in total. It is then possible to recursively turn each sorted adjacency list into a balanced binary search tree in linear time.

Proof of Theorem 10.0.1. Obtain a pseudoforest K-partition with $K \leq \lceil (1+\epsilon)d^* \rceil$ in time $\mathcal{O}(|E| \log |V| \log \Gamma \, \epsilon^{-1})$ with Theorem 4.2.3 via the conversion of Theorem 8.2.4. For fixed ϵ, this is possible in $\mathcal{O}(|E| \log |V|)$ time with Proposition 7.3.1.

We can assume for the remainder that $\epsilon \geq 1/K$. Let $k \leq K$ be the smallest integer such that $(k+1)/k \leq 1+\epsilon$. We have $k \in \mathcal{O}(\epsilon^{-1})$ and $\log k \in \mathcal{O}(\log |V|)$. In particular, if ϵ is a constant, so is k.

Divide the K pseudoforests evenly into k-tuples of pseudoforests, if possible, otherwise $l \leq k-1$ pseudoforests remain. Convert each k-tuple into $k+1$ forests and the remaining l pseudoforests into $l+1$ forests[6] with Lemma 10.2.2 and Lemma 10.3.1. The runtime is $\mathcal{O}(|E| k \log k + |E| \log |V|) \subseteq \mathcal{O}(|E| \log(|V|) \epsilon^{-1})$. □

Proof of Theorem 10.0.2. Obtain a partition into $p+1$ pseudoforests with Theorem 6.0.2 in $\mathcal{O}(|E| \log |V| \, p \log^* p)$ time via the conversion of Theorem 8.2.4. Using Lemma 10.2.2 and Lemma 10.3.1, we convert this into a partition of at most $p+2 \leq \Gamma + 2$ forests in $\mathcal{O}(|E| p \log p + |E| \log |V|)$ time. The claim follows. □

The second approach uses perfect hashing: For the set E edges from the universe $V \times V$, we construct a perfect hash function and maintain the set $E \setminus M = F$ in a hash table and store the current color information of each edge with it. The perfect hashing scheme of Fredman, Komlós, and Szemerédi [FKS84] allows worst-case constant runtimes for querying, insertion, and deletion. Constructing the perfect hash function is possible in $\mathcal{O}(|V|^2 |E|)$ time, and alternatively in $\mathcal{O}(|E|)$ expected time. The following lemma is proved analogously to Lemma 10.3.1.

Lemma 10.3.2. *In the situation of Lemma 10.2.1, we can determine the exchange edge in $\mathcal{O}(k)$ time with $\mathcal{O}(|E|)$ expected time for preprocessing using perfect hashing. All other operations have the same asymptotic complexity as in Lemma 10.2.2.*

Lemma 10.3.2 could be useful for the development of a randomized exact algorithm for arboricity. Recall that Gabow's algorithm has a runtime of $\mathcal{O}(|E|^{3/2} \log(|V|^2/|E|))$ [Gab98]. Since $p \leq \Gamma \in \mathcal{O}(\sqrt{|E|})$, even in the worst case we can convert k pseudoforests into $k+1$ forests in time $\mathcal{O}(|E|^{3/2})$ after constructing the perfect hash function. Since algorithms for pseudoarboricity are available that run in time $\mathcal{O}(|E|^{3/2}\sqrt{\log \log p})$ and even $\mathcal{O}(|E|^{3/2})$ (Theorems 4.1.4 and 6.0.1 via Theorem 8.2.4), we would obtain a faster exact algorithm if we can insert all edges of the constructed $(k+1)$-th forest into $F_1, \ldots, F_k$ fast enough, if this is feasible. Likewise, a deterministic exact algorithm with a runtime of $\mathcal{O}(|E| \log |V| \, \Gamma \log^* \Gamma)$ could then be within reach using Theorem 10.0.2.

6 In fact, unless $(1+\epsilon)K < K+1$ for the original ϵ, we can omit the outer ceiling in the statement of the theorem by converting one $(k+l)$-tuple.

11 PREPROCESSING ORIENTATIONS

We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil.
Yet we should not pass up our opportunities in that critical 3%.

— Donald E. Knuth, Structured Programming with go to Statements (1974)

In this chapter, we give a new conditional runtime estimate for computing the arboricity and pseudoarboricity. For the latter, this was published in [Blu16].

Recall from Theorem 8.3.3 that Gabow's algorithm for arboricity runs in time

$$\mathcal{O}\left(|E|^{3/2}\log\frac{|V|^2}{|E|}\right).$$

If we could increase the average density of the graph sufficiently, the logarithmic term would vanish. Bearing this in mind, we next discuss a preprocessing which does not affect the values $\lceil d^*\rceil$ and γ, but may decrease the graph size.

In the densest subgraph problem, a vertex can be safely removed from the graph if its degree is smaller than d^*. This was claimed in [KP94; KS09a; Nas+17], and we give a rigorous proof. A similar result holds for the fractional arboricity.

Lemma 11.0.1. *([KP94; KS09a; Nas+17],[TG16]) Let $G = (V, E)$ be a simple graph. A vertex u with $\deg(u) < d^*(G)$ cannot be in a densest subgraph of G. Furthermore, a vertex u with $\deg(u) < \gamma(G)$ cannot be in a subgraph in*

$$\arg \max_{\substack{(V_H,E_H)\subseteq G\\ |V_H|\geq 2}} \frac{|E_H|}{|V_H|-1}.$$

Proof. If $|E| = 0$, we have $d^*(G) = 0$ and thus the claim holds. If $|E| \geq 1$, a densest subgraph $H = (V_H, E_H)$ has at least two vertices. Assume that $\deg(u) < d^*(G)$ for some $u \in V_H$.

Let $(V_{H'}, E_{H'}) = G[V_H \setminus \{u\}]$. We have

$$\begin{aligned}
& \deg_H(u) \leq \deg(u) < d^*(G) = \frac{|E_H|}{|V_H|} \\
\Leftrightarrow\quad & -|E_H| < -|V_H| \deg_H(u) \\
\Leftrightarrow\quad & |V_H||E_H| - |E_H| < |V_H|(|E_H| - \deg_H(u)) \\
\Leftrightarrow\quad & \frac{|E_H|}{|V_H|} < \frac{|E_H| - \deg_H(u)}{|V_H|-1} = \frac{|E_{H'}|}{|V_{H'}|},
\end{aligned}$$

a contradiction to the assumption that H is a densest subgraph. The proof for γ is similar, see [TG16, Lemma 6]. □

We can now repeatedly remove vertices by applying Lemma 11.0.1 without changing the maximum density and the fractional arboricity of the graph. Note that the average density of a graph may *decrease* when a vertex v with $\deg(v) < d^*$ is removed.[1]

Lemma 11.0.2. *Let $d \leq d^*(G)$ for a simple graph $G = (V, E)$. Then we can obtain a subgraph $G' \subseteq G$ with $d^*(G') = d^*(G)$ and $\gamma(G') = \gamma(G)$ where $\deg_{G'}(v) \geq d$ for all $v \in V$ in $\mathcal{O}(|E|)$ time.*

Proof. Note that $d \leq d^*$ implies $d < \gamma(G)$.

We initialize a queue with all vertices whose degree is less than d, and store the current vertex degrees. While the queue is not empty, a vertex u is dequeued. Mark it as removed and decrease the degrees of all its unmarked neighbors by one. If a neighbor's degree falls below d for the first time, add it to the queue. Once the queue is empty, build the graph induced by the set of unmarked vertices. The algorithm's correctness is established by Lemma 11.0.1. It runs in $\mathcal{O}(|E|)$ time. □

Equipped with a constant-factor approximation algorithm, we can perform a fast preprocessing with Lemma 11.0.2, which can significantly reduce the input size.

Lemma 11.0.3. *Let $G = (V, E)$ be a simple graph. Then a subgraph $G' = (V', E') \subseteq G$ with $d^*(G') = d^*(G)$, $\gamma(G') = \gamma(G)$ and $|E'| \geq |V'|\, d^*/4$ can be obtained in $\mathcal{O}(|E|)$ time.*

Proof. Compute a 1/2-approximation $d^*(G)/2 \leq d \leq d^*(G)$ in time $\mathcal{O}(|E|)$ (see Chapter 7). Obtain the subgraph $G' = (V', E') \subseteq G$ from Lemma 11.0.2 with d in time $\mathcal{O}(|E|)$. For every vertex $v \in V'$, $\deg_{G'}(v) \geq d \geq d^*(G)/2$. We have

$$2|E'| = \sum_{v \in V'} \deg_{G'}(v) \geq |V'|\, d^*(G)/2,$$

thus $|E'| \geq |V'|\, d^*(G)/4$. □

We are now able to prove a new runtime bound for determining $\lceil d^* \rceil$. Note that if a graph is dense, i.e., $|E| \in \Theta(|V|^2)$, then $d^* \in \Theta(\sqrt{|E|})$. The reverse, however, is not true, which is why preprocessing is needed in the following theorem.[2]

1 An example for this is the complete graph K_6, where we attach a path of four vertices to some vertex. The average density is $19/10 = 1.9$, and $d^* = 15/6 = 2.5$. If one removes a vertex on the path with degree two, then the average density becomes $17/9 < 1.9$. However, exhaustively removing vertices is a different matter.

2 To see this, consider a complete graph K_n with a path of length n^2 attached to it: the graph is sparse, but the maximum density is in $\Theta(\sqrt{|E|})$.

Theorem 11.0.4. *Let $G = (V, E)$ be a simple graph. If $d^*(G) \in \Omega(\sqrt{|E|})$, the smallest maximum indegree $\lceil d^* \rceil$ and the arboricity Γ can be determined in time $\mathcal{O}(|E|^{3/2})$. A subgraph of density greater $\lceil d^* \rceil - 1$ can be found within the same runtime.*

Proof. If we have $d^* \in \Omega(\sqrt{|E|})$, then $d^* \geq c\sqrt{|E|}$ for some $c > 0$. Obtain $G' = (V', E')$ as in Lemma 11.0.3, we have

$$\begin{aligned} & |E'| \geq |V'|\, d^*/4 \geq |V'|\sqrt{|E|}\, c/4 \geq |V'|\sqrt{|E'|}\, c/4 \\ \Rightarrow\ & |E'| \geq |V'|^2\, c^2/16, \end{aligned}$$

and thus $|E'| \in \Omega(|V'|^2)$. The arboricity $\Gamma(G') = \Gamma(G)$ (by Theorem 8.2.7) can now be computed in $\mathcal{O}(|E|^{3/2})$ time with Gabow's algorithm. By applying Theorem 8.2.6 we obtain

$$\lceil d^*(G) \rceil + 1 = \lceil d^*(G') \rceil + 1 \geq \Gamma(G') \geq \lceil d^*(G') \rceil = \lceil d^*(G) \rceil .$$

As $\Gamma(G')$ has to be either $\lceil d^*(G) \rceil$ or $\lceil d^*(G) \rceil + 1$, a single test for $\Gamma(G') - 1$ suffices to determine $\lceil d^* \rceil$ in $\mathcal{O}(|E|^{3/2})$ time, for example by a test with the bipartite orientation network (see the proof of Theorem 5.0.1).

A subgraph of density greater $\lceil d^* \rceil - 1$ can be found with a single maximum flow computation on Goldberg's network, see Theorem 3.3.2. □

12 EXPERIMENTAL COMPARISONS FOR THE ORIENTATION PROBLEM

A class, in Java, is where we teach objects how to behave.

— Richard E. Pattis

12.1 RELATED WORK

We are not aware of performance comparisons of algorithms for the densest subgraph problem and the orientation and pseudoarboricity problems. However, extraction of densest subgraphs is a common application. To this end, Tsourakakis et al. [Tso+13] implement Goldberg's method and the greedy algorithm. Bălălău et al. [Băl+15] use Charikar's linear program and the greedy algorithm to find minimal densest subgraphs with small overlap. They consider Goldberg's method unsuitable for their purposes due to poor performance; they did not elaborate which flow algorithm this statement alludes to. Valari et al. [VKP12] and Nasir et al. [Nas+17] propose streaming algorithms for the top-k densest subgraph problem, which asks for the k densest edge-disjoint subgraphs (see [GGT16; Don+18] for algorithms and complexity in the overlapping case). Valari et al. implement Goldberg's method, but do not state which flow algorithm is used in their implementation.

12.2 SELECTION OF ALGORITHMS

We tested binary search methods with maximum flow algorithms, which were implemented in Java and run with OpenJDK 11.0.3. We implemented Dinitz's (D) algorithm and variants of the push-relabel algorithm. The relabel-to-front variant runs in $\mathcal{O}(|V|^3)$ [Cor+01], the highest-label variant (HL) runs in $\mathcal{O}(|V|^2\sqrt{|E|})$ time [CM88]. For the highest label variant, we added the global relabeling and gap heuristics [CG97]. The performance of the relabel-to-front variant is significantly worse on all instances, we do not report its runtimes for a more compact presentation. The interested reader is referred to [Blu16].

We also extend the highest-label variant to an algorithm for parametric flow networks (that does not use a binary search) as described by Gallo et al. (P-HL) [GGT89] (see Subsection 3.3.3). It has a runtime of $\mathcal{O}(|V|^2\sqrt{|E|})$ if we restrict ourselves to integral parameter guesses from the interval $\{0, \ldots, |V|-1\}$ [GT94]. We did not implement the variant with link-cut trees, which would yield a runtime of

$\mathcal{O}(|V||E|\log(|V|^2/|E|))$, because this data structure is sometimes less efficient in practice.[1]

The aforementioned algorithms are used to determine $\lceil d^* \rceil$ with Goldberg's method (which also computes a subgraph of density greater $\lceil d^* \rceil - 1$), its modification by Georgakopoulos and Politopoulos, and the re-orientation algorithm. The latter performs exact computation without approximation phases. The reason for this is that the stopping criterion was not met for the parameter choices in the proof of Theorem 6.0.1, and thus the approximation scheme outputs the optimum solution. We give more details on the length of augmenting paths in Section 12.5. The balanced binary search technique was also not implemented.

The lower and upper search interval bounds were initialized with the bounds from Section 8.4, also for LPs, but not bounds obtained from a constant-factor approximation. The reason for this is twofold: We could compute a $(1+\epsilon)$-approximation for an arbitrarily small ϵ with a runtime bound that depends on ϵ. While the linear-time 2-approximation algorithm could serve as a gold standard here, it may happen to compute a very good approximation and thus influence the binary search unduly.

The tests were repeated with the preprocessing from Lemma 11.0.3 to reduce the input sizes, it uses the greedy algorithm in order to obtain a lower bound on d^*. As noted in Section 7.1, the greedy algorithm computes a $1/2$-approximation to d^* that is never greater than the one obtained from the 2-approximation to its dual problem that is also computed by the greedy algorithm. The latter was used in [Blu16] for preprocessing. By using the former, we were able to improve the already significant reduction by preprocessing considerably on some instances. Again, preprocessing could be done using a $(1+\epsilon)$-approximation, which we choose not to do. We note another possible improvement that was not implemented: Every time a new lower bound on d^* is computed through an unsuccessful[2] test, we can again invoke preprocessing.

We also implemented a parameterized LP of Georgakopoulos and Politopoulos [GP07] (not discussed in this thesis) for d^* with a binary search for integral test values.[3] We implemented the orientation LP (3.16)-(3.20), once with the (continuous) variable d to be minimized and once with a binary search for integral test values that test the LP's feasibility for constant d. Recall that in the latter, the constraint matrix

1 An experimental study of link-cut trees in the Edmonds–Karp algorithm [EK72] is available, here the runtime heavily depends on the chosen link-cut trees variant and the class of input graphs [Wer06; TW07].

2 Note that in Kowalik's approximation scheme, a test may be reported to be unsuccesful although the test value is feasible. However, this only happens if the stopping criterion terminates the flow algorithm early, otherwise the value is indeed a lower bound.

3 This LP generalizes to vertex- and edge-weighted hypergraphs. Its constraint matrix is totally unimodular.

is totally unimodular, which guarantees integral extreme points. The LP solver might exploit this property further.

12.3 LP SOLVER AND HARDWARE CONFIGURATION

We use Gurobi 9.0 [Gur19], which is free for academic purposes, to solve the LPs. We report results for the dual simplex method on otherwise default settings. The dual simplex method is faster on average and uses less memory than the primal simplex method.

The tests were run on a single core of an Intel i7-5820K CPU with 3.3 GHz and 64 GB DDR-4 RAM. Note that Gurobi has multithreading capabilities, but we only allow it to use a single thread. The operating system is Debian 8.2.0 (64-bit). Note that for the largest graph, *Friendster*, it was necessary to save memory by using the data type byte (which uses eight bits) for arcs of capacity one and two.

12.4 INPUT GRAPHS

We use large simple graphs from the Stanford SNAP database [LK14] as input graphs, namely the *Amazon*, *DBLP*[4], *YouTube*, *LiveJournal*, *Orkut* and *Friendster* networks with about 1 million to 1.8 billion edges. The exact sizes are presented in Table 12.1 on the next page.

Even and Tarjan [ET75] propose a family of flow networks on which Dinitz's algorithm needs as much time as its asymptotic worst-case runtime estimate. Since our flow networks do not belong to this family, we propose a graph family where we expect the shortest augmenting paths to become quite long because of a mixture of varying degrees, high local densities, and a large diameter. For $n \in \mathbb{N}$, we define graph G_n as a union of the complete graphs $K_1, K_2, \ldots, K_n$, where every vertex of K_i is additionally connected to all vertices of K_{i+1} for $i = 1, \ldots, n-1$. G_n has $n(n+1)/2$ vertices and $(n^3 - n)/2$ edges in total and an average density $|E|/|V| = n - 1 \in \Theta(\sqrt{|V|})$. If $n \geq 2$, the vertices in K_{n-1} have a degree of $3n - 4$ in G_n, which is the maximum degree $\Delta(G_n)$. By Lemma 3.2.3, we have $d^* \leq 3n/2 - 2$, so $d^* \in \Theta(\sqrt{|V|})$. We denote the smallest $\epsilon > 0$ for which the stopping criterion of the approximation scheme is met by $\tilde{\epsilon}$. We expect that $\tilde{\epsilon}(G_n) \to 0$ as $n \to \infty$. We will not report runtimes for these graphs in LP-based approaches, and also no runtimes after preprocessing.

4 The *DBLP* graph is a collaboration network of computer scientists: Two authors are connected by an edge if they have at least one joint publication in the DBLP database. We note that Goldberg's method returns a subgraph with an average density of approximately 56.57 for test value $\lceil d^* \rceil - 1 = 56$. It has 115 vertices (authors). Unfortunately, we do not know how to resolve the vertex numbers to the corresponding author names, which would have been quite interesting.

Table 12.1: Characteristics of the input graphs. The maximum pathlength encountered in the re-orientation algorithm with Dinitz's algorithm is denoted by k, $\tilde{\epsilon}$ denotes the smallest ϵ for which the approximation scheme would terminate early (rounded to two decimal places). The runtime of the greedy algorithm is given in seconds.

Graph	Vertices	Edges	k	$\tilde{\epsilon}$	$\lceil d^* \rceil$	G. Time
Amazon	334,863	925,872	21	0.96	5	0
DBLP	317,080	1,049,866	5	67.2	57	0
YouTube	1,134,890	2,987,624	24	0.89	46	1
LiveJ.	3,997,962	34,681,189	12	3.58	194	7
Orkut	3,072,441	117,185,083	34	0.60	228	21
Friendster	65,608,366	1,806,067,135	31	0.86	274	529
G_{100}	5,050	499,950	11	1.58	134	0
G_{200}	20,100	3,999,900	13	1.47	277	0
G_{400}	80,200	31,999,800	18	1.03	567	0
G_{800}	320,400	255,999,600	25	0.74	1,152	6
G_{1600}	1,280,800	2,047,999,200	35	0.54	2,332	49

12.5 RESULTS

Table 12.2 on the facing page shows the effects and the runtime of preprocessing. (This does not include the time to greedily compute a 1/2-approximation, which is given in Table 12.1.) On the *Friendster* instance, the preprocessing takes less than a minute. On all other instances from the SNAP database, the runtimes is less than two seconds. The reduction of the number of edges of the graphs is slightly less than 38.2% for the Amazon instance and between 94.8% and 99.3% for the other instances. Therefore, preprocessing is a very effective tool in real-world instances whose degree distribution typically follows a power law [BA99]. However, graphs with different degree distributions cannot be expected to be reduced as much.

The runtime results for the linear programs are presented in Table 12.3 on the next page. For flow-based methods, this is done in Table 12.4 on page 154.

The orientation LP (3.16)-(3.20) is solved considerably faster than the Georgakopoulos–Politopoulos LP with a binary search. Using a binary search for the orientation LP with integral test values d often results in even better runtimes. The preprocessed *Friendster* instance is an exception, we suspect this is an outlier.

Table 12.2: Sizes of the input graphs from the SNAP database after preprocessing. The preprocessing time (excluding the time to compute the 1/2-approximation) is given in seconds.

Graph	Vertices	Edges	k	$\tilde{\varepsilon}$	P. Time
Amazon	169,008	572,762	20	0.96	0
DBLP	280	13,609	4	15.74	0
YouTube	2,269	103,342	16	0.74	0
LiveJ.	2,539	466,625	8	2.70	0
Orkut	26,670	6,077,055	22	0.67	1
Friendster	49,370	13,503,583	11	2.33	47
G_{100}	2,839	351,813	11	1.42	0
G_{200}	10,509	2,666,751	14	1.17	0
G_{400}	40,014	20,586,976	19	0.87	0
G_{800}	154,800	160,614,000	26	0.65	1
G_{1600}	601,605	1,256,057,830	35	0.50	11

Table 12.3: Runtimes (in seconds) for computing d^* with linear programs using Gurobi. The orientation LP coupled with a binary search uses integral test values only, i.e., $\lceil d^* \rceil$ is determined. If the computation requires more than the available 64 GB of RAM, 'out of memory' (OOM) is stated in the table.

Graph	Orientation LP	BS for d-orientation	G.–P. LP
Amazon	152	12	136
DBLP	8	7	43
Youtube	452	57	309
LiveJ.	314	314	OOM
Orkut	OOM	OOM	OOM
Friendster	OOM	OOM	OOM
Amazon-P	82	7	67
DBLP-P	0	0	1
Youtube-P	3	3	7
LiveJ.-P	10	9	69
Orkut-P	2,168	1,747	6,529
Friendster-P	3,018	23,806	36,098

Table 12.4: Time (in seconds) needed to compute $\lceil d^* \rceil$ with several flow algorithms. The suffix -P denotes instances reduced by preprocessing. The flow algorithms were Dinitz's algorithm (D), the highest-label (HL) variants of the push-relabel algorithm, and the extension of the latter to parametric flow problems (P-HL). If the time limit of 10 hours is exceeded, this is indicated by TLE.

	Goldberg's			Geor.-Polit.		Re-orientation	
Graph	D	HL	P-HL	D	HL	D	HL
Amazon	3	200	9,164	2	262	2	189
DBLP	2	11	8,417	0	7	1	6
Youtube	15	665	TLE	5	635	15	210
LiveJ.	113	14,047	TLE	34	2,165	77	1,264
Orkut	391	TLE	TLE	119	18,252	352	TLE
Friendster	14,178	TLE	TLE	6,066	TLE	14,209	TLE
Amazon-P	1	204	1,903	2	225	1	205
DBLP-P	0	0	0	0	0	0	0
Youtube-P	0	4	4	0	4	0	5
LiveJ.-P	0	27	22	0	26	0	27
Orkut-P	9	3,055	2,948	14	3,044	8	3,165
Friendster-P	17	13,680	14,304	31	13,488	17	15,152
G_{100}	0	1	44	0	1	0	1
G_{200}	7	22	1,306	2	29	6	11
G_{400}	84	434	TLE	21	554	90	352
G_{800}	995	4,870	TLE	228	4,773	118	5,619
G_{1600}	10,572	TLE	TLE	4,149	TLE	1,399	TLE

For flow-based methods, Dinitz's algorithm significantly outperforms push-relabel algorithms, and also the LP-based approaches. The highest-label variant with a binary search is often much faster than its parametric extension. We note that the runtime of a test heavily depends on the test value. For example, the number of relabelings in the HL algorithm on the preprocessed *Friendster* instance is $71,629$ for test value $d = 274 = \lceil d^* \rceil$, but $2,221,784,361$ for $d = 273$. Shrinking the search interval may thus not provide a significant benefit in practice because the values slightly below $\lceil d^* \rceil$ appear to be the hardest to test. This implies parallelizing the binary search cannot be expected to yield a significant speedup.

The Georgakopoulos–Politopoulos technique, which removes vertices after unsuccessful tests in Goldberg's method, is considerably faster than Goldberg's on a few instances, but slightly slower on others. A single unsuccessful test removes $0 - 90\%$ of the remaining vertices. The effect seems to be roughly the same on the original and preprocessed instances. However, as the lower bounds are typically tighter on preprocessed instances, there are less unsuccessful tests. Sometimes, all tests but one are successful. If the only unsuccessful test is the last test, then there clearly is no improvement at all. There is no known estimate of how many vertices can be expected to be removed from an unsuccessful test [GP07].

We report our findings on the augmenting path lengths in the execution of Kowalik's scheme in Tables 12.1 and 12.2. The value k is the maximum length of an augmenting path in Dinitz's algorithm encountered among all test values of the binary search. Perhaps non-surprisingly, this maximum length is often (but not always) attained at test value $\lceil d^* \rceil - 1$. We note that this value is empirical and may depend on the implementation of Dinitz's algorithm, as well as the initial orientation[5]. The value $\tilde{\epsilon}$ denotes the smallest ϵ for which the stopping criterion is met for k.

Choosing $\epsilon \geq 1$ in the approximation scheme is not reasonable because the linear-time 2-approximation algorithm is preferable. Therefore, assume that $\epsilon < 1$. On some graphs, the stopping criterion is not met for any $\epsilon < 1$, i.e., the exact solution will be returned. On all input graphs, the choices[6] for ϵ in the proof of Theorem 6.0.1 are smaller than the critical epsilon, i.e., the (first) approximation phase computes the optimum solution. This is easily recognized and the second phase need not be performed. It would be interesting to see a family of graphs where the approximation phase does not compute the optimum solution, but helps in accelerating the binary search as in our theoretical result.

5 Different initial orientations were tried, but the effect is negligible.

6 Recall that ϵ can be set using any constant-factor approximation. We assume here that ϵ is set with the exact value of d^* and $\ln |V|$, the latter stems from the Taylor expansion (see Theorem 4.2.3).

13 PROBLEMS INVOLVING CONNECTED SUBGRAPHS

That the (connected) densest subgraph problem is solvable in polynomial time is a fortunate exception to the rule of thumb that subgraph problems are NP-complete. A restriction on the number of vertices or edges typically makes a problem NP-hard.

In the densest k-subgraph problem, we are interested in the densest subgraph on k vertices. The decision variant asks whether a subgraph on k vertices exists whose average density is at least d. This decision problem is easily seen to be NP-complete by a reduction from the clique problem (Definition 2.6.4).

Theorem 13.0.1. *The densest k-subgraph problem is NP-complete.*

Proof. A solution to the problem can be verified in polynomial time. There is a k-clique in G if and only if the densest k-subgraphs of G are complete. Hence, using the same k and setting $d = k(k-1)/2$ constitutes a polynomial-time reduction. □

The best known approximation algorithm is due to Bhaskara et al.

Theorem 13.0.2 ([Bha+10])**.** *A $|V|^{1/4+\epsilon}$-approximation of the densest k-subgraph problem can be determined in $\mathcal{O}(|V|^{1+\epsilon})$ time for every $\epsilon > 0$.*

In seminal work, Khot ruled out a PTAS assuming a variant of the exponential time hypothesis.

Theorem 13.0.3 ([Kho06])**.** *There is no PTAS for the densest k-subgraph problem unless SAT can be solved in expected time $\mathcal{O}(2^{n^{\epsilon}})$ for an arbitrarily small constant $\epsilon > 0$.*

There are numerous other problems involving subgraphs, density or weights, and connectivity. For example, determining whether an induced subgraph with exactly k edges exists is NP-complete [Tri93; AT94]. We note that if we require the induced subgraph to be connected, the problem is also NP-complete by the same reduction from vertex cover.

Definition 13.0.4 (Vertex Cover)**.** Given a simple graph $G = (V, E)$ and $k \in \mathbb{N}$, is there a set $S \subseteq V$ with $|S| \leq k$ such that for every edge $e \in E$, at least one endpoint of e is in S?

NP-hardness of vertex cover can be proved by a reduction from the clique problem (Definition 2.6.4).

Theorem 13.0.5 ([Tri93; AT94])**.** *Given a simple graph $G = (V, E)$ and $1 \leq k \leq |V|$, it is NP-complete to decide whether an induced (or connected induced) subgraph of G with exactly k edges exists.*

Very general results of Yannakakis and Lewis will be given in the following. A class of graphs is called a *graph property* if any two isomorphic graphs either both are in the class or both are not. Put differently, a graph property is preserved under isomorphisms and does not depend on a representation such as a labeling or drawing of the graph. A graph property Π is called *hereditary* if it holds for all induced subgraphs of G if it holds for G itself.

Theorem 13.0.6 ([Yan78; Lew78]). *Let Π be a hereditary graph property that does not hold for all graphs, but there are arbitrarily large graphs for which it does hold. Then the problem of deciding whether there is a set $S \subseteq V$ of at least k vertices such that the subgraph $G[S]$ has property Π is NP-hard. If Π can be decided in polynomial time, the problem is NP-complete.*

The result holds analogously for induced connected graphs.

We now informally describe the three NP-complete problems that we will deal with in the following chapters. They are all related to each other, and they can be solved with a mixed integer linear program (MILP) based on the maximum density. We will describe this MILP only for the k-cardinality problem because it is strongest when the number k of vertices in the subgraph is specified. The k-cardinality tree problem is to find a subtree with k vertices of minimum total edge weight. It will be dealt with in the next chapter.

The Steiner tree problem in graphs is to find a subtree of minimum total edge weight that spans a set of given terminal vertices. ILP approaches, as well as approximation algorithms based on them, will be discussed in Chapters 16 to 18.

The maximum weight connected subgraph problem (MWCS) is to find a connected subgraph of maximum total vertex weight. It will be introduced in Chapter 19 and some preprocessing rules will be developed.

14 THE K-CARDINALITY TREE PROBLEM

Given a weighted graph, the k-cardinality tree problem asks for a subtree with k vertices that minimizes the sum of edge weights.

Definition 14.0.1 (k-Cardinality Tree Problem)**.** Given a non-negatively weighted simple graph $G = (V, E, w)$ and integers $k, l \in \mathbb{N}$, decide whether a subtree of G of exactly k vertices[1] exists with total weight at most l.

It is also called the k-MST problem, because the special case $k = |V|$ is the minimum spanning tree problem (MST). A minimum spanning tree can be computed in time $\mathcal{O}(|E|\alpha(|E|, |V|))$ [Chaoob], where α denotes a variant of the inverse Ackermann function (Section 2.3). Pettie and Ramachandran describe a time-optimal algorithm to compute the MST [PR02], although its runtime is currently unknown. An MST can also be computed in $\mathcal{O}(|E|)$ expected time [KKT95].

For constant k, the k-cardinality tree problem can be solved in polynomial time by brute force. For general k, it can be shown to be NP-complete, which was proved independently by several authors [LZ93; Fis+94; Rav+96].

Theorem 14.0.2 ([Rav+96])**.** *The k-cardinality tree problem is NP-complete, even for planar graphs and graphs with weights from the set* $\{1, 2, 3\}$.

This can be proved by a reduction from the Steiner tree problem in graphs (Definition 16.2.1), which is NP-complete for planar graphs and unweighted graphs (see Section 16.3).

There is also a rooted variant of the problem, where a designated root vertex $r \in V$ must be selected. We will only consider the unrooted problem, which can clearly be solved by $|V|$ applications of an algorithm for the rooted problem. Garg [Gar05] points out that the rooted problem also reduces to the unrooted problem: By adding $|V|$ vertices to the root and adding edges (r, v) with zero weight for each of the additional vertices, selection of the root is forced for $k' = k + |V|$.

Several approximation algorithms have been developed for the k-cardinality tree problem [Awe+95; RV95; Rav+96]. Blum et al. [BRV96; BRV99] gave the first constant-factor approximation with a factor of 17 that uses the primal-dual method of Goemans and Williamson [GW95] for the prize-collecting Steiner tree problem (Definition 19.1.1). Garg [Gar96] extended their approach for a 3-approximation algorithm and a simpler 5-approximation algorithm. He also shows the integrality gap of the LP formulation he uses to be at least three.

1 Note that some authors require selection of k edges and thus of $k+1$ vertices.

The latter algorithm was recast by Chudak et al. [CRW04] using the prize-collecting Steiner tree problem, which is a Lagrangean relaxation of the k-cardinality tree problem. Arya and Ramesh [AR98] give a 2.5-approximation algorithm by applying a pruning procedure before Garg's 3-approximation algorithm. Arora and Karakostas [AK06] give a $(2+\epsilon)$-approximation algorithm for every $\epsilon > 0$, which is also a modification of Garg's 3-approximation algorithm. Finally, Garg [Gar05] gives a 2-approximation algorithm, which is the best known to date. In the following section, we will investigate integer linear programs for solving the problem exactly.

There are also metaheuristic approaches to the problem, see [BS04] and the literature cited therein. We close our review by noting that an exact algorithm based on tree decompositions exists [CMZ12] (see also [Alt+14]).

14.1 INTEGER LINEAR PROGRAMS

Although the connected components of a graph can be determined easily in time $\mathcal{O}(|V|+|E|)$ with DFS or BFS, it is surprisingly difficult to model connectivity in an integer linear program. Several approaches use an exponential number of constraints. There are so-called *compact* formulations that use only a polynomial number of constraints. For example, Wong [Won84] gives a multi-commodity flow formulation for the Steiner tree problem with $\Theta(|V||E|)$ variables. This is a theoretically important result because solving the LP relaxation is possible in polynomial time. Building upon an idea of Cohen [Coh10], Althaus et al. [Alt+14] propose a mixed integer linear program (MILP) based on the maximum density to model connectivity with only $\mathcal{O}(|V|+|E|)$ constraints.

We compare this MILP to a well-known existing integer linear program, the subtour elimination constraints. Chimani et al. [Chi+10] perform similar comparisons specifically for the k-cardinality tree problem, and we loosely follow their notation. Notable comparisons of several linear programming relaxations for Steiner tree problems were done by Goemans and Myung [GM93] and Polzin and Vahdati Daneshmand [PD01].

14.1.1 *Subtour Elimination Constraints*

Before we consider subgraphs, let us first model a spanning tree of a graph in an ILP. Recall Lemma 2.2.2: A graph of $|V|$ vertices is a tree if and only if it has exactly $|V|-1$ edges and is acyclic. By introducing binary variables x_e for every edge $e \in E$, we can easily model the number of edges:

$$\sum_{e\in E} x_e = |V|-1. \tag{14.1}$$

To enforce acyclicity, we forbid *subtours*: For a set $S \subseteq V$, we require that at most $|S| - 1$ edges are selected, i.e., there is no cycle going through all vertices in S:

$$\sum_{e \in E[S]} x_e \leq |S| - 1, \quad S \subseteq V : |S| \geq 2. \tag{14.2}$$

Constraints (14.2) are called *subtour elimination constraints* (the constraint for $S = V$ is redundant by (14.1)). They were introduced by Dantzig et al. [DFJ54] for the NP-hard travelling salesman problem (TSP), which asks for a simple cycle through all vertices of the graph of minimum cost. (The constraint for set $S = V$ is hence excluded for TSP.)

Now consider the subgraph case. For every vertex $v \in V$, define a binary variable y_v that determines whether v is selected or not. If we are looking for a subtree of unknown size, we can replace (14.1) by

$$\sum_{v \in V} y_v = \sum_{e \in E} x_e + 1, \tag{14.3}$$

If we are looking for a fixed size k, as in the k-cardinality tree problem, we can use $\sum_{v \in V} y_v = k$ and $\sum_{e \in E} x_e = k - 1$ instead. Constraints (14.2) are replaced by

$$\sum_{e \in E[S]} x_e \leq \sum_{v \in S} y_v - y_t, \qquad t \in S,\ S \subseteq V : |S| \geq 2, \tag{14.4}$$

the *generalized subtour elimination constraints* (*GSEC*) [Fis+94]. The *GSEC* subtree polyhedron is defined as

$$P_{GSEC} := \{(x, y) \in [0, 1]^{|E|+|V|} \mid (x, y) \text{ satisfies (14.3) and (14.4)}\}.$$

As writing down the ILP takes exponential time, one can use the cutting-plane method (Subsection 2.11.7): Start with a subset of the constraints, for example the constraints for sets S with $|S| = 2$. After solving, find violated inequalities and add at least one of them. One continues solving in this way until no violated inequalities can be found. Fischetti et al. [Fis+94] show that a violated *GSEC* inequality can be found with $2|V| - 2$ maximum flow computations, if one exists. The flow network has to be updated with every separation call. Chimani et al. [Chi+10] use a directed equivalent of the *GSEC* constraints that requires at most $|V|$ flow computations on a flow network that has to be built only once, only its capacities are updated according to the LP solution.

Another equivalent formulation with $\mathcal{O}(|V||E|)$ constraints, based on multi-commodity flows, can be obtained analogously to the approach of [Lju04] (see [Chi+10]). While it can be solved in polynomial time and is thus theoretically pleasing, $|V||E|$ can be prohibitively large in practice, and it is known from related problems that solving this LP is inferior to using the cutting-plane method [Lju+06; CKM07].

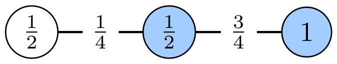


Figure 14.1: A fractional feasible of P_{MD} for $k = 2$ on a graph with three vertices that violates a *GSEC* constraint for $S = \{C, R\}$, the blue vertices in the center and to the right: $x_{CR} = \frac{3}{4} > \frac{1}{2} = y_C$. The corresponding Constraint (14.5) is $x_{CR} = \frac{3}{4} \leq (1 - \frac{1}{2})(1 + \frac{1}{2})$.

14.1.2 *A Formulation Based on the Maximum Density*

To enforce acyclicity, one can alternatively employ Lemma 3.1.5: A graph is acyclic if and only if its maximum density is smaller than one, or more precisely, at most $1 - 1/|V|$. Thus, we can use the constraints

$$\sum_{e \in E[S]} x_e \leq \left(1 - \frac{1}{|S|}\right) \sum_{v \in S} y_v \qquad S \subseteq V : 2 \leq |S| < |V|. \tag{14.5}$$

in addition to the cardinality constraint (14.3). Note that the constraints imply that the endpoints of a selected edge must be selected as well in a feasible integral solution. We denote the polyhedron by

$$P_{MD} := \{(x, y) \in [0, 1]^{|E|+|V|} \mid (x, y) \text{ satisfies (14.3) and (14.5)}\}.$$

One can think of the constraints based on the maximum density as averages of *GSEC* constraints for a set S. It is not surprising that they are weaker.

Proposition 14.1.1. *Constraints (14.5) are implied by the GSEC constraints (14.4), i.e.,*

$$P_{GSEC} \subseteq P_{MD}.$$

Furthermore, there are instances where the containment is proper.

Proof. Let $G = (V, E)$ be a simple graph, $(x, y) \in P_{GSEC}(G)$, and let $S \subseteq V$ with $|S| \geq 2$. We consider the strongest constraint for this set: Let $t^* \in S$ with $t^* = \arg\max_{t \in S} y_t$. We have

$$\begin{aligned} \sum_{uv \in E[S]} x_{uv} \overset{(14.4)}{\leq} \sum_{v \in S} y_v - y_{t^*} &= \sum_{v \in S} y_v - \frac{1}{|S|} \sum_{v \in S} y_{t^*} \\ &\leq \sum_{v \in S} y_v - \frac{1}{|S|} \sum_{v \in S} y_v \\ &= \left(1 - \frac{1}{|S|}\right) \sum_{v \in S} y_v. \end{aligned}$$

The *GSEC* constraints are indeed strictly stronger, even on a graph with three vertices and two edges, as the example in Figure 14.1 shows. □

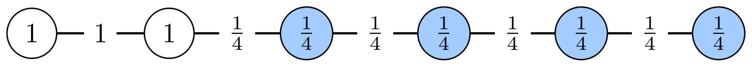


Figure 14.2: A fractional assignment of (x, y) for $k = 3$ on a path of six vertices. Let A, B, C, D, E, F denote the vertices from left to right. The constraints for the set $\{C, D, E, F\}$ (shaded blue) are respected in the *GSEC* formulation (and, by Proposition 14.1.1, also in (14.5)), but not by Constraints ($14.5_{>\bar{k}}$). All other sets have feasible constraints in all three formulations.

If an upper bound $\bar{k}$ is imposed on the number of vertices to select, it suffices to consider sets S with $|S| \leq \bar{k}$. In this bounded case, the maximum density of any feasible integral solution is at most $1 - 1/\bar{k}$.

However, fractional solutions of the LP could have more than $\bar{k}$ vertices with positive values y_v. Adding the constraints for $|S| > \bar{k}$ cuts off some of these fractional solutions, as we shall see shortly. For these constraints, we can use the constant $\bar{k}$, which yields a tighter bound than $|S|$:

$$\sum_{uv \in E[S]} x_{uv} \leq \left(1 - \frac{1}{|S|}\right) \sum_{v \in S} y_v, \qquad S \subseteq V : 2 \leq |S| \leq \bar{k}, \quad (14.5_{\leq \bar{k}})$$

$$\sum_{uv \in E[S]} x_{uv} \leq \left(1 - \frac{1}{\bar{k}}\right) \sum_{v \in S} y_v, \qquad S \subseteq V : \bar{k} < |S| \leq |V|. \quad (14.5_{> \bar{k}})$$

We denote the corresponding polyhedron by

$$P_{MD}^{\bar{k}} := \{(x, y) \in [0, 1]^{|E|+|V|} \mid (x, y) \text{ satisfies } (14.3), (14.5_{\leq \bar{k}}), \text{ and } (14.5_{> \bar{k}})\}.$$

Constraints ($14.5_{>\bar{k}}$) are strictly stronger than the corresponding Constraints (14.5). An example of this can be seen in Figure 14.2 and Table 14.1 on the next page. Moreover, ($14.5_{>\bar{k}}$) and ($14.5_{\leq\bar{k}}$) define a polyhedron incomparable to the *GSEC* polyhedron, i.e., neither is a subset of the other. This can be shown with an example of Althaus et al. [Alt+14, Figure 2] for a related formulation that will be introduced in the next subsection. However, the solution in it does not satisfy an additional cut that will be introduced in Subsection 14.1.4. The example in Figure 14.2, however, shows incomparability (for $k \geq 3$) even with the additional cut.

14.1.3 *An MILP Based on Orientations*

Recall the fractional orientation problem, which is dual to the densest subgraph problem: The maximum density of a graph equals the smallest maximum fractional indegree (see Section 3.5). Cohen [Coh10] uses this to obtain an MILP for a spanning tree problem. A tree has a maximum density of exactly $1 - 1/|V|$, and thus there is an orientation

Table 14.1: Several constraints for set $S = \{C, D, E, F\}$ in the example of Figure 14.2 on the previous page.

Constraints	$S = \{C, D, E, F\}$
(14.4)	$\frac{3}{4} \leq 4 \cdot \frac{1}{4} - \frac{1}{4}$
(14.5)	$\frac{3}{4} \leq (1 - \frac{1}{4})\,(4 \cdot \frac{1}{4})$
(14.5$_{>\bar{k}}$)	$\frac{3}{4} \not\leq \frac{2}{3} = (1 - \frac{1}{3})\,(4 \cdot \frac{1}{4})$

of maximum fractional indegree $1 - 1/|V|$. Moreover, since $|V| - 1$ edges are to be oriented among $|V|$ vertices, every vertex receives *exactly* $1 - 1/|V|$ in such a fractional orientation.

Althaus et al. [Alt+14] generalize Cohen's idea to subgraphs. The smallest maximum fractional indegree of a tree that has k vertices can be $(1 - 1/\bar{k})$ at most. It must be $(1 - 1/\underline{k})$ at least when a lower bound $\underline{k}$ on the number of vertices is given. The MILP is

$$f_{uv,u} + f_{uv,v} = x_{uv}, \qquad uv \in E, \tag{14.6}$$

$$\sum_{uv \in E} f_{uv,v} \leq \left(1 - \frac{1}{\bar{k}}\right) y_v, \; v \in V, \tag{14.7}$$

$$\sum_{uv \in E} f_{uv,v} \geq \left(1 - \frac{1}{\underline{k}}\right) y_v, \; v \in V, \tag{14.8}$$

$$y_v \in \{0, 1\}, \qquad v \in V,$$

$$x_{uv} \in \{0, 1\}, \qquad uv \in E,$$

$$f_{uv,u}, f_{uv,v} \in [0, 1], \qquad uv \in E.$$

Note that Constraint (14.8) is not necessary to enforce acyclicity. Constraints (14.7) and (14.8) form an equation if $\underline{k} = \bar{k} = k$:

$$\sum_{uv \in E} f_{uv,v} = \left(1 - \frac{1}{k}\right) y_v, \qquad v \in V. \tag{14.9}$$

Proposition 14.1.2 ([Alt+14]). *For $\underline{k} = \bar{k}$, (14.3) is implied by (14.9) and one of the following:*

$$\sum_{u \in V} y_v = k, \tag{14.10}$$

$$\sum_{uv \in E} x_{uv} = k - 1. \tag{14.11}$$

Proof. If (14.9) and (14.10) hold, we obtain

$$\sum_{uv \in E} x_{uv} = \sum_{v \in V} \sum_{uv \in E} f_{uv,v} \overset{(14.9)}{=} \sum_{v \in V} \left(1 - \frac{1}{k}\right) y_v \overset{(14.10)}{=} k - 1.$$

Likewise, if (14.9) and (14.11) hold, we have

$$k-1 \overset{(14.11)}{=} \sum_{uv\in E} x_{uv} = \sum_{v\in V}\sum_{uv\in E} f_{uv,v} \overset{(14.9)}{=} \left(1-\frac{1}{k}\right)\sum_{v\in V} y_v.$$

Division by $(1-1/k)$ (for $k \neq 1$) yields Constraint (14.10). □

In the paper by Althaus et al. [Alt+14], 'spanning edge' constraints $x_{uv} \leq y_u, y_v$ for $uv \in E$ were part of the formulation. These are exactly the *GSEC* constraints for $S = \{u, v\}$. They are not necessary for the formulation because Constraints (14.6) and (14.7) guarantee that the endpoints of a selected edge must be selected in every integral solution. Adding them cuts off fractional solutions, however (such as the solutions in Figure 14.1 on page 162 and Figure 14.3 on the next page), and the number of these constraints is only linear. We can now define the polyhedron based on orientations:

$$P_O^{\underline{k},\overline{k}} := \{(x, f_u, f_v, y) \in [0,1]^{3|E|+|V|} \mid (x, f_u, f_v, y) \text{ satisfies (14.3)} \\ \text{and (14.6)-(14.8)}\}.$$

If $\underline{k} = \overline{k}$ or only one bound on the number of vertices is required, we alter the superscript accordingly. In order to compare this polyhedron to polyhedra with different variables, we will use the projection $\text{proj}_{x,y}$ onto the variable space (x, y) via $x_{uv} = f_{uv,u} + f_{uv,v}$.

Proposition 14.1.3. *Constraints* ($14.5_{\leq \overline{k}}$) *for* $|S| = \overline{k}$ *and Constraints* ($14.5_{>\overline{k}}$) *are implied by Constraints* (14.6) *and* (14.7).

Proof. For any $S \subseteq V$ with $|S| \geq \overline{k}$, we have

$$\sum_{uv\in E[S]} x_{uv} \overset{(14.6)}{=} \sum_{v\in S}\sum_{\substack{uv\in E\\ u\in S}} f_{uv,v} \overset{(14.7)}{\leq} \left(1-\frac{1}{\overline{k}}\right)\sum_{v\in S} y_v.$$

□

Corollary 14.1.4 ([Alt+14]). *P_{GSEC} and $\text{proj}_{x,y}\left(P_O^{\overline{k}}\right)$ are not comparable in general.*

Theorem 14.1.5. *We have*

$$P_{GSEC} \cap \text{proj}_{x,y}\left(P_O^{\overline{k}}\right) \subseteq P_{MD}^{\overline{k}}.$$

Furthermore, there are instances where the containment is proper.

Proof. This follows from Proposition 14.1.1 and Proposition 14.1.3. □

Constraints (14.7) for $|S| < \overline{k}$ are strictly weaker than ($14.5_{\leq \overline{k}}$), an example can be seen in Figure 14.3 on the following page.

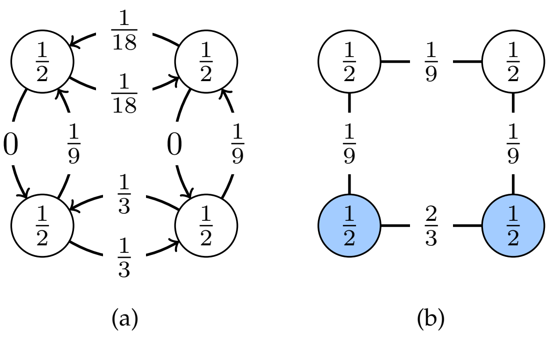


Figure 14.3: An example graph that shows Constraints ($14.5_{\le \bar{k}}$) are strictly stronger than (14.7) for $\underline{k} = 1, \bar{k} = 3$. (a) A feasible P_O-solution is shown. It does not respect (14.12). (b) The projection onto (x, y) respects (14.7). The set of two vertices on the bottom (shaded blue) does not respect ($14.5_{\le \bar{k}}$), however.

14.1.4 *Additional Cuts*

Another constraint, described by Althaus et al. [Alt+14], can be added to the orientation MILP:

$$f_{uv,v} \ge \frac{1}{\underline{k}} x_{uv}, \qquad uv \in E. \tag{14.12}$$

The constraint is obviously valid for $x_{uv} = 0$. For $x_{uv} = 1$, we have $y_u = 1 = y_v$ as remarked earlier, and see from Constraint (14.7) that $f_{uv,u} \le 1 - \frac{1}{\underline{k}}$, which by (14.6) implies that Constraint (14.12) is respected by every feasible integral solution. Althaus et al. did not provide an example to substantiate their claim that this cuts off fractional solutions. One such example is given in Figure 14.3a. However, the solution in the example, projected onto (x, y), would also be cut off by the *GSEC* constraints for $|S| = 2$.

Lucena and Resende [LR01] describe the following cut: If at least two vertices are to be selected, i.e., $\underline{k} \ge 2$, every selected vertex must have at least one incident edge:

$$\sum_{uv \in E} x_{uv} \ge y_v, \qquad v \in V. \tag{14.13}$$

As remarked earlier, the example of a fractional *GSEC* solution used by Althaus et al. [Alt+14] to show Corollary 14.1.4 does not respect (14.13). The question whether P_O^k is contained in P_{GSEC} with (14.13) added can be answered in the negative, however: Figure 14.2 on page 163 shows a fractional *GSEC* solution that respects (14.13), but is not in P_O^k. Could it be the case that Constraints (14.13) are implied?

Proposition 14.1.6. *For $\underline{k} = \bar{k}$, Constraints (14.8) and (14.12) imply Constraints (14.13).*

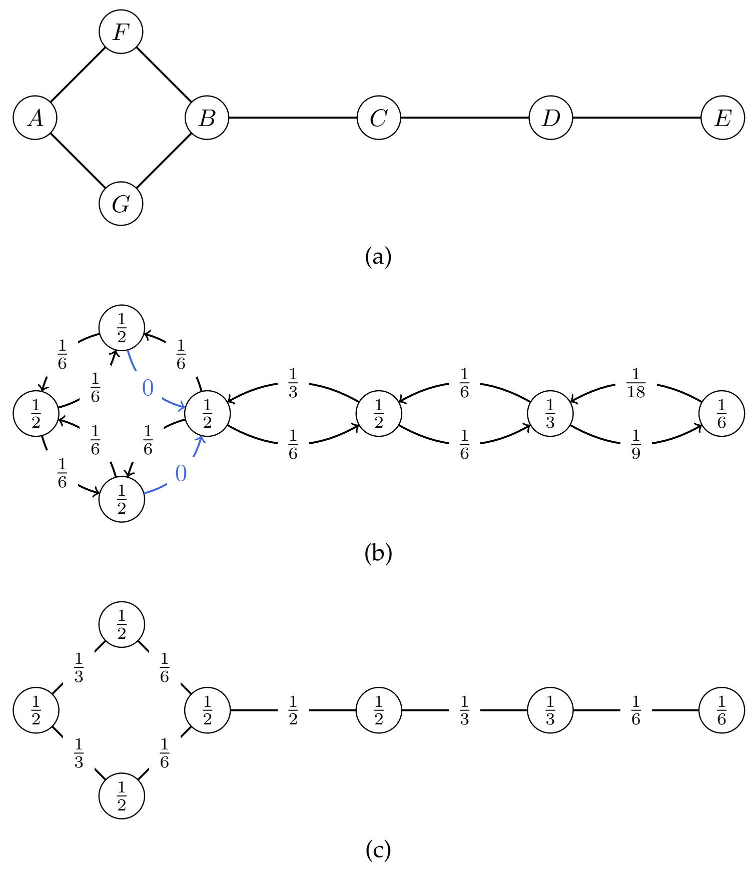


Figure 14.4: (a) A graph of a k-cardinality tree instance for $k = 3$. (b) A feasible solution to P_O^k. It does not respect Constraint (14.12) in the blue edges. (c) The projected solution is feasible for the *GSEC* constraints. It also respects Constraint (14.13).

Proof. If $\overline{k} = 1$, there is nothing to show. For every $v \in V$, we have

$$\left(1 - \frac{1}{\underline{k}}\right) y_v \overset{(14.8)}{\leq} \sum_{uv \in E} f_{uv,v} \overset{(14.6)}{=} \sum_{uv \in E} 1 - f_{uv,u}$$
$$\overset{(14.12)}{\leq} \left(1 - \frac{1}{\overline{k}}\right) \sum_{uv \in E} x_{uv}.$$

For the case $2 \leq \underline{k} = \overline{k}$, we can divide by $\left(1 - \frac{1}{k}\right)$ and obtain exactly (14.13). □

Could it be the case that Constraints (14.12) are implied by the intersection of P_O^k, P_{GSEC} and Constraints (14.13)? We answer this in the negative.

Proposition 14.1.7. *By adding Constraints (14.12), we obtain a strictly stronger formulation than $P_{GSEC} \cap \operatorname{proj}_{x,y}\left(P_O^k\right)$ with Constraint (14.13).*

Proof. Consider the example graph in Figure 14.4a for $k = 3$. The feasible P_O-solution in Figure 14.4b projects onto (x, y) in Figure 14.4c.

We show that there is no feasible P_O-solution that projects onto this (x, y) while respecting (14.12). If Constraint (14.12) were required to hold, we would have $f_{FB,B}, f_{GB,B} \geq 1/18$. Therefore, B may receive at most $2/9$ from C by (14.7), i.e., $f_{BC,B} \leq 4/18$. Thus $f_{BC,C} \geq 5/18$ by (14.6). Since C may receive at most $1/3$, $f_{CD,C} \leq 1/18$. But (14.12) dictates that $f_{CD,C} \geq 1/9 = 2/18$. Hence, there can be no feasible solution for this choice of (x, y) that respects all constraints. □

Our new results show that (M)ILP formulations for the k-cardinality tree problem are far from understood completely. We wonder whether they can be used to design new approximation algorithms for the problem.

15 EXPERIMENTAL COMPARISON FOR K-CARDINALITY TREES

As we saw in the previous chapter, there is an MILP for connectivity (of subgraphs) with only a linear number of constraints. If the number of vertices to be selected is bounded from above by $\bar{k}$, as is the case in the k-cardinality tree problem, this formulation is tightened and becomes incomparable to *GSEC*. Furthermore, we saw two additional cuts that can be used to cut off feasible fractional solutions. In this chapter, we will only report results for a fixed number of vertices k where the MILP is tightest.

15.1 FORMULATIONS SELECTED FOR COMPARISON

We implemented a directed variant of the *GSEC* ILP as described by Chimani et al. (see Subsection 14.1.1). The separation is performed with the Edmonds–Karp algorithm [EK72] in an implementation by Sedgewick and Wayne [SW11]. Following [Lju04; Chi+10], we initially added constraints whose projections are the 'spanning edge' constraints

$$x_{uv} \le y_u, y_v, \qquad uv \in E, \tag{15.1}$$

of *GSEC* for $|S| = 2$. Using them initially considerably speeds up the computation. We search for violated inequalities for every feasible integral solution that is found in the solving process. This is not necessary[1], but allows us to report the gap for the whole ILP when we terminate early, not just the gap for the current set of constraints. Additionally, we search for violated inequalities when solving the relaxation of a node has yielded a feasible fractional solution.

We do not try extracting multiple cuts from a single maximum flow, which was used in [KM98; Lju+06], but only one. Still, we typically find a few dozen to a few hundred[2] cuts in the flow computations of each separation step.

We implemented the orientation MILP (14.6), (14.9), (14.10), and (14.12). Recall that Constraints (14.13) are implied by Proposition 14.1.6 for a fixed number of vertices k. We also added Constraints (15.1).

1 Assuming that we search for violated inequalities when the solver returns an optimum integral feasible solution, and continuing the solving process after adding such an inequality.

2 This way, one can only find as many violated cuts as vertices with $y_v > 0$ exist in the solution, i.e., at most k in integer solutions. Feasible fractional solutions can involve more than k such vertices, however.

This formulation of size $\mathcal{O}(|V| + |E|)$ is referred to as 'O-MILP' in the following.

We also combined the two approaches: The O-MILP constraints (with the directed equivalent of (15.1)) are used as a basis that guarantees connectivity for integral feasible solutions, and the *GSEC* equivalent is used to tighten the LP relaxation further on fractional feasible solutions. We note that the directed *GSEC* equivalent has an additional artificial root vertex $r \notin V$ that does not have a corresponding variable y_r, but arc variables $x_{(r,v)}$ to every $v \in V$. These latter variables do not appear in the O-MILP constraints.

15.2 INPUT DATA, MACHINE CONFIGURATION, AND SOLVER SETTINGS

There is a library of instances for the k-cardinality tree problem, KCTLIB[3] [BB05]. It includes 35 instances generated by Blesa and Xhafa [BX00] with a tool by Jörnsten and Løkketangen [JA97]. With current hardware and advanced ILP solvers, solving these latter instances exactly with up to $1,000$ vertices and $2,000$ edges is not a challenge for $k = 21$.[4] All instances but one could be solved to optimality within 20 seconds using the *GSEC* equivalent. The remaining instance took 146 seconds to solve. We thus exclude the instances from our experiments.

The library also includes ten instances based on grid graphs generated in [BB05], and one Leighton graph [Bea90]. The dimensions of the grids are specified in the instance name. The Leighton graph has 450 vertices and 8,168 edges. There are also five instances based on the Steiner tree problem [Bea90], but these are also solved in a matter of seconds and hence excluded. We report results for the eleven relevant instances with a time limit of one hour.

We also use the ACTMOD dataset of the 11th DIMACS Implementation Challenge[5] for the MWCS problem (see Table 19.1 in Chapter 19), which has vertex weights. Because MWCS is a maximization problem, we multiplied the weights with -1 for our minimization problem. Some instances have several connected components. All components except the largest have less than 30 vertices, they are all discarded.

We use the same machine configuration as in Section 12.3 and utilize only one processor core and one Gurobi thread. We ran the tests once with Gurobi on default settings, and once with modified settings for a fair comparison:

3 As of January 2020, available at http://www.iiia.csic.es/~christian.blum/downloads/KCT_benchmark.tgz.

4 Note that the $k = 20$ used in [BB05] is the number of edges. We continue to denote the number of vertices by k.

5 http://dimacs11.zib.de

1. Gurobi tries to reduce and tighten the LP model before the solving process, which is called *presolve* (see [Ach+16]). This capability was deactivated.

2. Gurobi tries to find cutting planes of its own that are not problem-specific. This was also deactivated.

3. Gurobi uses heuristics in order to find feasible solutions. The time to be spent on heuristics was set to zero percent.

On default settings, Gurobi aims to spend about 5% of the available time on heuristics. We note that in approaches that add cutting planes via 'callbacks', some presolve capabilities must be shut off with the *PreCrush* parameter.

For all approaches, we provide the solver with a tree of k vertices as a feasible start solution. We choose the tree returned by a lexicographic BFS started in vertex 1. No reductions are performed on the graphs. Note that all graphs are (after removing small components described above) connected, so connected components are not treated separately.

15.3 RESULTS

We list gaps and runtimes after one hour for the two approaches and their combination for $k = 20$ in Table 15.1 and for $k = 40$ in Table 15.2 with modified settings for Gurobi, and for default settings in Tables 15.3 and 15.4, respectively.

With modified settings, the orientation MILP is solved to optimality on only one instance within an hour for $k = 20$, and not a single one for $k = 40$. Several instances are solved to optimality with the *GSEC* equivalent, but its gap is not always better. More instances are solved to optimality with the combination of the two; the gap is often, but not always better. Overall, the results suggest that the combination provides an advantage, the effect is slightly more prononced for $k = 20$. This aligns with theoretical considerations, as the O-MILP becomes less tight with growing k.

With default settings, the results are different: Most instances are solved to optimality with the O-MILP, and it can be regarded as the clear winner. The combination comes close for $k = 20$, but is significantly outperformed for $k = 40$. The *GSEC* ILP clearly exhibits the worst performance for both cardinalities.

The reason that the O-MILP exhibits this excellent performance with default settings may be that the fewer constraints there are, the more time Gurobi can expend on its built-in solving procedures. Thus, a sparse model may be better than a tighter model, even if separation is used. While more experiments should be undertaken, we think that the O-MILP is a good alternative to *GSEC*, at least for moderate values of k. We note that in order to be competitive with state-of-the-art approaches, one should use problem-specific heuristics and parameter tuning. For certain graph families, one might fare better with settings tailored to them.

Table 15.1: Time in seconds (rounded) to solve the k-cardinality tree problem for $k = 20$ with (M)ILP approaches. The gap after one hour of solving is reported for modified settings of Gurobi.

	O-MILP		*GSEC*		Both	
Instance	Gap	Time	Gap	Time	Gap	Time
bb15x15_1	39.58%	3600	0.00%	95	0.00%	2
bb15x15_2	33.47%	3600	0.00%	521	0.00%	72
bb33x33_1	85.59%	3600	83.92%	3600	82.02%	3600
bb33x33_2	77.51%	3600	86.07%	3600	83.24%	3600
bb45x5_1	25.42%	3600	0.00%	3169	0.00%	1727
bb45x5_2	32.00%	3600	0.00%	522	0.00%	24
bb50x50_1	88.40%	3600	84.83%	3600	82.85%	3600
bb50x50_2	90.72%	3600	87.10%	3600	85.63%	3600
bb100x10_1	58.72%	3600	76.57%	3600	17.28%	3600
bb100x10_2	75.58%	3600	84.65%	3600	0.00%	19
le450_15a	0.00%	3072	0.00%	435	0.00%	331
drosophila001	291.42%	3600	247.10%	3600	247.10%	3600
drosophila005	470.96%	3600	225.47%	3600	680.32%	3600
drosophila0075	2470.30%	3600	964.67%	3600	964.67%	3600
HCMV	254.83%	3600	26.35%	3600	0.00%	579
lymphoma	54.07%	3600	0.00%	167	0.00%	27
metabol. . . 1	81.58%	3600	16.52%	3600	0.01%	83
metabol. . . 2	123.42%	3600	8.43%	3600	0.00%	235
metabol. . . 3	10.53%	3600	432.61%	3600	0.00%	50

Table 15.2: Time in seconds (rounded) to solve the k-cardinality tree problem for $k = 40$ with (M)ILP approaches. The gap after one hour of solving is reported for modified settings of Gurobi.

	O-MILP		*GSEC*		Both	
Instance	Gap	Time	Gap	Time	Gap	Time
bb15x15_1	65.30%	3600	42.52%	3600	72.26%	3600
bb15x15_2	82.07%	3600	71.38%	3600	32.01%	3600
bb33x33_1	87.09%	3600	81.50%	3600	80.85%	3600
bb33x33_2	93.12%	3600	81.83%	3600	80.25%	3600
bb45x5_1	60.41%	3600	73.02%	3600	49.99%	3600
bb45x5_2	58.04%	3600	0.00%	860	0.00%	80
bb50x50_1	93.47%	3600	81.87%	3600	80.82%	3600
bb50x50_2	89.20%	3600	85.30%	3600	84.80%	3600
bb100x10_1	89.61%	3600	73.97%	3600	0.00%	716
bb100x10_2	87.89%	3600	82.89%	3600	81.62%	3600
le450_15a	68.03%	3600	0.00%	3418	0.00%	3060
drosophila001	205.13%	3600	173.44%	3600	123.76%	3600
drosophila005	408.64%	3600	197.60%	3600	389.50%	3600
drosophila0075	403.88%	3600	531.19%	3600	531.19%	3600
HCMV	161.00%	3600	104.43%	3600	103.29%	3600
lymphoma	110.74%	3600	0.00%	71	0.00%	27
metabol…1	315.65%	3600	303.45%	3600	70.40%	3600
metabol…2	794.96%	3600	162.54%	3600	159.62%	3600
metabol…3	90.89%	3600	251.84%	3600	248.04%	3600

Table 15.3: Time in seconds (rounded) to solve the k-cardinality tree problem for $k = 20$ with (M)ILP approaches. The gap after one hour of solving is reported for Gurobi's default settings.

	O-MILP		*GSEC*		Both	
Instance	Gap	Time	Gap	Time	Gap	Time
bb15x15_1	0.00%	2	0.00%	422	0.00%	1
bb15x15_2	0.00%	5	0.00%	237	0.00%	18
bb33x33_1	0.00%	30	84.10%	3600	0.00%	46
bb33x33_2	0.00%	23	86.07%	3600	0.00%	69
bb45x5_1	0.00%	5	39.69%	3600	0.00%	293
bb45x5_2	0.00%	7	0.00%	327	0.00%	31
bb50x50_1	0.00%	312	85.21%	3600	7.90%	3600
bb50x50_2	0.00%	47	87.02%	3600	0.01%	941
bb100x10_1	0.00%	17	75.99%	3600	0.00%	9
bb100x10_2	0.00%	11	30.01%	3600	0.00%	13
le450_15a	0.00%	4	0.00%	823	0.00%	277
drosophila001	66.01%	3600	247.10%	3600	252.39%	3600
drosophila005	27.29%	3600	680.32%	3600	694.43%	3600
drosophila0075	27.73%	3600	964.67%	3600	984.61%	3600
HCMV	0.01%	205	110.29%	3600	0.00%	71
lymphoma	0.00%	51	0.00%	133	0.00%	36
metabol...1	0.00%	21	320.99%	3600	0.00%	47
metabol...2	0.00%	19	0.00%	2485	0.00%	48
metabol...3	0.00%	4	426.17%	3600	0.00%	5

Table 15.4: Time in seconds (rounded) to solve the k-cardinality tree problem for $k = 40$ with (M)ILP approaches. The gap after one hour of solving is reported for Gurobi's default settings.

	O-MILP		*GSEC*		Both	
Instance	Gap	Time	Gap	Time	Gap	Time
bb15x15_1	0.00%	93	48.46%	3600	0.00%	1167
bb15x15_2	0.00%	13	58.93%	3600	0.00%	2001
bb33x33_1	0.00%	1209	81.29%	3600	80.86%	3600
bb33x33_2	0.00%	369	81.83%	3600	5.58%	3600
bb45x5_1	0.00%	56	72.83%	3600	0.00%	2258
bb45x5_2	0.00%	9	0.00%	1514	0.00%	32
bb50x50_1	0.00%	1578	81.72%	3600	6.78%	3600
bb50x50_2	0.00%	925	85.35%	3600	100.00%	3600
bb100x10_1	0.00%	18	73.61%	3600	0.00%	1740
bb100x10_2	0.00%	191	82.43%	3600	8.58%	3600
le450_15a	0.00%	28	0.00%	1887	0.00%	1581
drosophila001	64.66%	3600	173.44%	3600	68.19%	3600
drosophila005	17.72%	3600	389.50%	3600	396.54%	3600
drosophila0075	15.65%	3600	531.19%	3600	2.74%	3600
HCMV	0.00%	581	0.00%	401	0.00%	267
lymphoma	0.00%	21	0.00%	710	0.00%	30
metabol...1	0.01%	589	299.19%	3600	621.15%	3600
metabol...2	0.01%	443	161.41%	3600	1.91%	3600
metabol...3	0.01%	78	249.15%	3600	4.82%	3600

16 THE STEINER TREE PROBLEM

16.1 GEOMETRIC STEINER TREE PROBLEMS

The Euclidean Steiner tree problem is often attributed to the eponymous Jakob Steiner, but was considered earlier by Joseph Diez Gergonne and Heinrich Christian Schumacher. For a full historical overview, see Brazil et al. [Bra+14]. In the geometric variants of the problem, some *terminal points* are given that have to be connected by straight lines. In addition, points in the plane can be selected at will to serve as 'hubs' between the terminals. These optional points are called *Steiner points*. The solution must be a tree because distances (metrics) are non-negative, it is called a *Steiner tree*.

Definition 16.1.1 (Euclidean Steiner Tree Problem)**.** Given a set of n points in the plane, is there a Steiner tree connecting these points that has total Euclidean length at most k?

This contrasts the Euclidean spanning tree problem, where only the given points are to be used. NP-hardness of the Euclidean Steiner tree problem was established by Garey et al.

Theorem 16.1.2 ([GGJ77])**.** *The Euclidean Steiner Tree Problem is NP-hard.*

It is unknown if the problem is in NP. The reason is that the distances can be irrational numbers, even when the point coordinates are restricted to be integers. However, by rounding up distances, one obtains a modified distance function for which the problem is in NP. This problem can then be used to approximate the original problem arbitrarily close by scaling [GGJ77].

A rectilinear Steiner tree is a tree whose line segments are all either horizontal or vertical. In other words, the L_1 metric is to be minimized. No irrational numbers can arise when the points have integer coordinates, in this case, the problem is in NP.

Definition 16.1.3 (Rectilinear Steiner Tree Problem)**.** Given a set of n points in the plane with integer coordinates, is there a Steiner tree connecting these points that has total L_1-length at most k?

Theorem 16.1.4 ([GJ77])**.** *The Rectilinear Steiner tree problem is NP-complete.*

The proof uses a chain of reductions from the vertex cover problem (see Definition 13.0.4) in planar graphs, which Garey et al. show to be NP-complete [GJS76].

16.2 THE STEINER TREE PROBLEM IN GRAPHS

Let us now turn to the Steiner tree problem in the graph setting.

Definition 16.2.1 (Steiner Tree Problem in Graphs)**.** Given a weighted graph $G = (V, E, w)$ with $w : E \to \mathbb{R}_0^+$, a set $R \subseteq V$ of *terminals*, and $k \in \mathbb{N}$, is there a subtree of G spanning R (a *Steiner tree*) with minimum total edge weight at most k?

The vertices $V \setminus R$ are the *Steiner vertices*. We will call the objective value of a Steiner tree its *cost*. If all weights are positive, one can equivalently ask for a subgraph because removing an edge on a cycle of positively weighted edges reduces the cost. However, we do allow edge weights of zero because we will add 'dummy' edges of weight zero in the analysis in Chapter 17. For convenience, we shall call this problem the Steiner tree problem in the following chapters.

A *terminal spanning tree* is a Steiner tree that does not contain Steiner vertices. Without loss of generality, one can replace the graph (V, E) with its *metric closure*, i.e., the complete graph on V where the weight of an edge uv is the length of the shortest path from u to v with respect to the original weight function on the edges. A terminal spanning tree of minimum cost in the metric closure is a 2-approximation for the Steiner tree problem [GP68; Vaz01].

16.3 NP-COMPLETENESS AND SPECIAL CASES

We will deal with two special cases of the Steiner tree problem in graphs.

Definition 16.3.1 (Quasi-Bipartiteness and Claw-Free Instances)**.** An instance of the Steiner tree problem in graphs is *quasi-bipartite* if no two Steiner vertices are neighbors. If for every Steiner vertex v, all edges incident to v have the same weight, the instance is *uniformly quasi-bipartite*.

A *Steiner claw* is a Steiner vertex with at least three Steiner vertices as neighbors. An instance is *claw-free* if it does not contain any Steiner claws.[1]

We will see in Section 17.3 that in these special cases, certain LP relaxations are equivalent from the polyhedral point of view, and that better approximation factors can be obtained.

The NP-completeness of the Steiner tree problem is widely reported to have been proved by Karp [Kar72]. However, the proof is erroneous, and we shall see a working proof in this section. Karp claims to prove NP-completeness of the Steiner tree problem by a reduction from the exact cover problem.

1 This differs from the usual definition of claw-freeness: A graph is claw-free if it does not contain a claw as an *induced* subgraph. In this definition, K_4 would be claw-free.

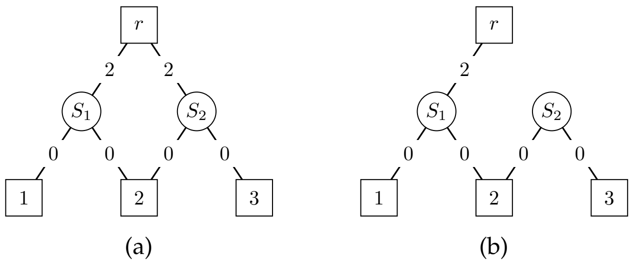


Figure 16.1: (a) A Steiner tree instance, constructed from an exact cover instance $U = \{1, 2, 3\}, S_1 = \{1, 2\}, S_2 = \{2, 3\}$ in the purported reduction by Karp. (b) An optimum Steiner tree of this instance.

Definition 16.3.2 (Exact Cover)**.** Given a ground set $U = \{1, \dots, m\}$ and subsets $S_1, \dots, S_n \subseteq U$, can U be covered with a selection $S_{i_1}, \dots, S_{i_j}$ of pairwise disjoint subsets, for some $j \in \mathbb{N}$?

The reduction Karp proposes is the following: Create one vertex for every subset and one vertex for every element of U, and add a 'root' vertex r:

$$V := \{r\} \cup \bigcup_{i=1}^{n} \{S_i\} \cup U,$$
$$E := \{(r, S_i) \mid i \in \{1, \dots, n\}\} \cup \{(S_i, x) \mid i \in \{1, \dots, n\}, x \in S_i\}.$$

The edge weights are defined as

$$\begin{aligned} w(r, S_i) &:= |S_i|, & i &= 1, \dots, n, \\ w(S_i, x) &:= 0, & x \in S_i, i &= 1, \dots, n. \end{aligned}$$

Consider the following exact cover instance: $U = \{1, 2, 3\}$, $S_1 = \{1, 2\}$, $S_2 = \{2, 3\}$. Clearly, the set U cannot be covered by pairwise disjoint subsets, so it is a 'no'-instance.

However, the purported reduction would create a Steiner tree instance (Figure 16.1a) with edges from the root having a weight of two each, and $k = 3$. A Steiner tree of weight two exists (Figure 16.1a): Select the edge from r to S_1 and all zero-weight edges, for a total cost of two, so it is a 'yes'-instance. Thus, this does not constitute a reduction!

We now give a correct and surprisingly simple NP-completeness proof, which is implicit in instances derived from set cover that exhibit a large integrality gap (e.g., [Byr+10; KPT11]). The reduction is quite similar to Karp's idea. The set cover problem is defined as follows.

Definition 16.3.3 (Set Cover)**.** Given a ground set $U = \{1, \dots, n\}$ and subsets $S_1, \dots, S_m \subseteq U$, can U be covered with at most $k \in \mathbb{N}$ subsets?

NP-hardness is immediate because vertex cover (Definition 13.0.4) is a special case of the set cover problem. There is a greedy approximation algorithm for set cover with approximation factor $\ln(n) + 1$ [Joh74; Lov75]. Determining a $(1 - \epsilon)\ln(n)$-approximation is NP-hard for every $\epsilon > 0$ [DS13; DS14].

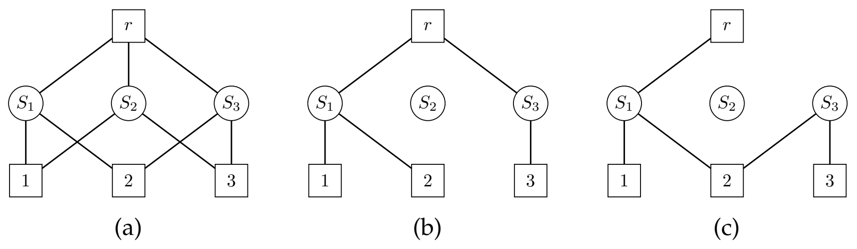


Figure 16.2: (a) A Steiner tree instance constructed in the proof of Theorem 16.3.4 from a set cover instance $S_1 = \{1, 2\}, S_2 = \{1, 3\}, S_3 = \{2, 3\}$ on $U = \{1, 2, 3\}$. The weights are uniform. (b) An optimum solution that directly corresponds to an optimum solution $S_1 \cup S_3 = U$ of the underlying set cover instance. (c) An optimum solution that does not directly correspond to an optimal solution of the underlying set cover instance.

Theorem 16.3.4 (Partially claimed by [RV99]). *The Steiner tree problem in graphs is* NP*-complete, even on quasi-bipartite graphs with uniform weights.*

Proof. Given a Steiner tree T for a graph G, it can be verified in polynomial time that T spans the terminals and that its cost are bounded by k. Thus the problem is in NP.

To show NP-hardness, we state a reduction from the set cover problem. It is illustrated in Figure 16.2.

Given a set cover instance $(n, S_1, \dots, S_m)$, construct a graph G with $m + n + 1$ vertices. Set $n + 1$ vertices to be terminals. Of these, n correspond to the n elements of U to be covered. The extra terminal r can be thought of as a 'root' from which the sets are selected. The m remaining Steiner vertices correspond to the m sets. Connect r to every Steiner vertex with weight 1. For every S_i, connect the Steiner vertex corresponding to it to all terminals corresponding to elements in S_i, each time with weight 1. Set $k' = n + k$ in the Steiner tree instance.

If there is a set cover of size at most k, then there is a Steiner tree of weight $n + k$: Select the edges from r to the sets in the set cover, and select the n element terminals from them in an arbitrary fashion. Note that this is not necessarily the only optimal solution, see Figure 16.2c for an example.

If there is a Steiner tree of weight at most $n + k$ in the transformed instance, then there must be at least one Steiner vertex that is 'selected from the root' r. If for some non-root terminal t, more than one incident edge is selected (again, see Figure 16.2c with $t = 2$), we apply the following algorithm: Let (u, t) be a selected edge that goes to such a non-root terminal t such that (r, u) is not selected. De-select (u, t) and select (r, u) instead. This is also a Steiner tree, and its cost stay the same. We can repeat the step until we end up with a Steiner tree of cost at most $n + k$ that corresponds to a set cover of size at most k. $\square$

NP-completeness for quasi-bipartite instances was claimed without proof by Rajagopalan and Vazirani [RV99], possibly with the above proof in mind. A proof of NP-completeness in general graphs with uniform weights can also be given via a reduction from 3-SAT [PS02]. The following inapproximability result is due to by Chlebík and Chlebíková, it uses an inapproximability result of Håstad [Hås01] for a system of linear equations over $\{0, 1\}$ with exactly three variables per equation, where the number of satisfied equations is to be maximized.

Theorem 16.3.5 ([CC08]). *It is NP-hard to approximate the Steiner tree problem in graphs within a factor of* 1.01063*, even for claw-free instances, and within* 1.00791 *for uniformly quasi-bipartite instances.*

The addition 'even for claw-free instances' is our observation – the gadgets in [CC08, Figure 1] do not contain Steiner claws, and gadgets are only linked together at terminals [CC08, Figure 3]. An interesting open question is if gadgets that contain Steiner claws lead to a better inapproximability factor.

An earlier inapproximability result due to Thimm [Thi03] is, according to [CC08], flawed, but not necessarily beyond repair.

Lichtenstein [Lic82] shows that planar 3-SAT is NP-complete, i.e., a certain graph corresponding to the 3-SAT instance is planar. He suggests that this can be used to show NP-completeness of the Steiner tree problem on planar instances. In fact, the aforementioned reduction from 3-SAT to the Steiner tree problem can be analogously carried out for planar 3-SAT instances, which shows NP-completeness in the planar case.

Theorem 16.3.6 ([Lic82]+[PS02]). *The Steiner tree problem in planar graphs is NP-complete.*

However, a PTAS does exist for the planar case [BKK07].

16.4 OVERVIEW OF APPROXIMATION ALGORITHMS

Based on an ILP, the so-called undirected cut formulation, it is possible to obtain a 2-approximation for the Steiner tree problem in graphs with a primal-dual scheme [GW95] or iterative rounding [Jai01]. The undirected cut relaxation has an integrality gap of two even for $R = V$ (i.e., the minimum spanning tree problem) [Vaz01]. Hence, it alone cannot be used to obtain an approximation factor of less than two.

Better approximation factors were achieved in a sequence of papers [KZ97; HP99], the best being $1 + \ln(3)/2 + \epsilon < 1.55$ [RZ00; RZ05]. These algorithms are all combinatorial and use k-restricted components. We only discuss LP-based approximation algorithms in this thesis, some are also based on k-restricted components.

The bidirected cut relaxation, which will be introduced in the following section, is strictly stronger than the undirected cut relaxation

[CR94]. However, the best known upper bound on its integrality gap is also two, and the best known lower bound is $36/31 \approx 1.161$ (see Chapter 18). In quasi-bipartite instances, a $(3/2+\epsilon)$-approximation algorithm exists [RV99], it is a variant of the primal-dual scheme.

The open question whether there is an LP relaxation with integrality gap smaller than two was answered affirmatively by Byrka et al. [Byr+10; Byr+13]. Their algorithm approximately solves the hypergraphic LP relaxation (*HYP*, see Section 16.6), which is then used to sample a component of the graph. This component is contracted. The algorithm continues in the same way on the contracted graph until it has been contracted into a single vertex. Although their algorithm provides an approximation factor of $\ln(4)+\epsilon \approx 1.39$, they were only able to show an integrality gap of at most $1+\ln(3)/2 \approx 1.55$. (An alternative proof of this gap was subsequently given by Chakrabarty et al. [CKP10b].) This discrepancy is due to the fact that the LP is solved after each contraction. Goemans et al. [Goe+12] show how to update the LP solution instead of re-solving, which shows an integrality gap of $\ln(4)$ for *HYP*. However, solving *HYP* exactly is NP-hard.[2]

The special cases of quasi-bipartite and claw-free instances will be discussed in Section 17.3.

16.5 THE BIDIRECTED CUT RELAXATION

In the bidirected cut relaxation (*BCR*), we create an auxiliary directed graph that has two directed edges $(u,v),(v,u)$ for every $uv \in E$, and there are variables $z_{(u,v)}, z_{(v,u)}$. Both have the cost $w(uv)$ of the original edge in the objective function. One vertex is chosen as the root terminal r. The goal is to find an assignment to the variables x such that every cut $S \subseteq V \setminus \{r\}, S \cap R \neq \emptyset$ is crossed by at least one edge. This ensures that there is a path from every terminal to the root. Let $\delta^+(S)$ denote the set of directed edges (u,v) with $u \in S, v \notin S$. The bidirected cut relaxation is as follows.

$$\min \quad \sum_{uv \in E} w(uv)(z_{(u,v)} + z_{(v,u)}) \tag{16.1}$$

$$\text{s.t.} \quad \sum_{(u,v)\in\delta^+(U)} z_{(u,v)} \geq 1, \quad U \subseteq V \setminus \{r\} : U \cap R \neq \emptyset, \tag{16.2}$$

$$z_{(u,v)}, z_{(u,v)} \geq 0, \quad uv \in E. \tag{16.3}$$

BCR can be shown to equivalent for all choices of the root terminal [GM93]. Let us now adapt the generalized subtour elimination constraints (Subsection 14.1.1) to the Steiner tree problem:

2 The number of constraints in *HYP* is exponential in the graph size. Hence, the fact that solving it is NP-complete (in the graph size) does not violate the widely believed hypothesis P $\neq$ NP via a polynomial-time algorithm for solving LPs.

$$\sum_{e \in E} x_e = \sum_{v \in V \setminus \{r\}} y_v, \tag{16.4}$$

$$\sum_{e \in E[S]} x_e \leq \sum_{v \in V} y_v - y_t, \qquad t \in S, S \subseteq V : S \cap T \neq \emptyset, \tag{16.5}$$

$$y_t = 1, \qquad t \in R, \tag{16.6}$$

$$x_e \geq 0, \qquad e \in E, \tag{16.7}$$

$$y_v \geq 0, \qquad v \in V. \tag{16.8}$$

Note that Constraint (16.5) for $S = V$ implies $y_v \leq 1$ for all $v \in V$, otherwise the cardinality constraint (16.4) would be violated for the maximum y_t [Fel+16]. Goemans and Myung compared the strength of *BCR* and *GSEC*.

Theorem 16.5.1 ([GM93]). *BCR and GSEC (as in (16.4)-(16.8)) are polyhedrally equivalent.*

The conversion from a feasible *BCR* solution to a *GSEC* solution of the same value is simply the projection $x_{uv} = x_{(u,v)} + x_{(v,u)}$ and $y_u = \sum_{(u,v) \in \delta^+(\{u\})} z_{(u,v)}$ for $u \in V \setminus \{r\}$.

Edmonds proves the following for the minimum spanning tree problem.

Theorem 16.5.2 ([Edm67]). *For a Steiner tree problem instance with $R = V$, the polyhedron of GSEC is integral.*

This also shows that the *BCR* polyhedron is integral [Byr+13], and hence *BCR* has a smaller integrality gap (namely, 1) than the undirected cut formulation for $R = V$. However, in general, the best known upper bound is two, although it is believed that this is not tight [RV99].

16.6 THE HYPERGRAPHIC RELAXATION

The hypergraphic relaxation was introduced by Warme [War98]. It is a generalization of the subtour constraints to hypergraphs.

A *component* is a tree in G where all terminals contained in the tree are leaves. If in addition, every leaf of the tree is a terminal, it is called a *full component*. Let $\mathcal{K}$ denote the set of full components, and create a variable x_C for all $C \in \mathcal{K}$. Let $R(C)$ denote the set of terminals in C. The cost $c(C)$ of a full component C is the cost of a minimum

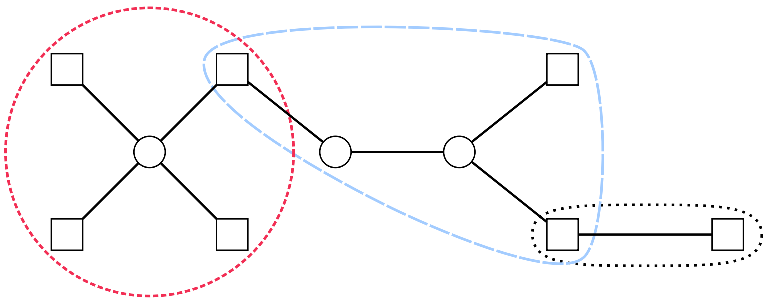


Figure 16.3: A Steiner tree that is the edge-disjoint union of three full components (dashed and dotted in red, blue and black).

Steiner tree in the component. For any $t \in \mathbb{R}$, let $(t)_0^+ = \max(0,t)$. The hypergraphic relaxation (*HYP*) is

$$\begin{aligned}
\min \quad & \sum_{C \in \mathcal{K}} c(C)x_C && && (16.9)\\
\text{s.t.} \quad & \sum_{C \in \mathcal{K}} x_C(|R(C) \cap S| - 1)_0^+ \leq |S| - 1, && \emptyset \neq S \subseteq R, && (16.10)\\
& \sum_{C \in \mathcal{K}} x_C(|R(C)| - 1) = |R| - 1, && && (16.11)\\
& x_C \geq 0, && C \in \mathcal{K}. && (16.12)
\end{aligned}$$

In a *full* Steiner tree for a subset $R' \subseteq R$ of terminals the leaves are exactly R'. Without loss of generality, we can assume that all leaves of a Steiner tree are terminals because a Steiner leaf can be safely removed.[3] A full Steiner tree can be written as a union of edge-disjoint full components (see Figure 16.3 for an example). Moreover, the union of full components selected in a feasible integral solution of (16.10)-(16.12) form a full Steiner tree. The objective value of an integral feasible solution is exactly the cost of the corresponding full Steiner tree.

The directed component relaxation (*DCR*) is the directed analog of the hypergraphic relaxation. It was introduced by Polzin and Vahdati Daneshmand [PD03] and was subsequently used by Fung et al. [Fun+12] and Byrka et al. [Byr+13].

Let $\mathcal{C}$ denote the set of all pairs $C = (r', R')$ with $R' \subseteq R$ and $r' \in R'$, the *directed components*. We interpret C to be the minimum Steiner tree on R' with total cost $c(C)$ and all edges directed towards the *sink* r', i.e., an arborescence rooted at r' with the direction of edges reversed. For a set $S \subseteq R$, we say that C *crosses* S if $r' \notin S$, and $S \cap R' \neq \emptyset$. Let $\delta^+(S)$ denote the set of directed components that cross S.

Fix an arbitrary root $r \in R$. The directed component relaxation (*DCR*) is

3 Note that a minimum Steiner tree can only have such vertices if their incident edges have weight zero.

$$\begin{aligned}
\min \quad & \sum_{C \in \mathcal{C}^k} c(C) x_C && && (16.13)\\
\text{s.t.} \quad & \sum_{C \in \delta^+(S)} x_C \geq 1, && \emptyset \neq S \subseteq R \setminus \{r\}, && (16.14)\\
& x_C \geq 0, && C \in \mathcal{C}. && (16.15)
\end{aligned}$$

Every Steiner tree, when viewed as an (inverted) arborescence, can be written as a union of edge-disjoint directed components. The components selected in a feasible integral solution form an arborescence. If the solution is optimal, the union of the minimum Steiner trees corresponding to these components form a minimum Steiner tree.

Theorem 16.6.1 ([PD03; CKP10a]). *DCR and HYP are polyhedrally equivalent.*

If we only consider components (r', R') with $|R'| \leq k$, we obtain the set $\mathcal{C}^k$ of directed component of maximum size k, the *k-restricted directed components*, and denote the restricted ILP as DCR^k. Since there are at most $\binom{|V|}{k}$ subsets of R of size k, we have $|\mathcal{C}^k| \in \mathcal{O}(kn^k)$. An optimum Steiner tree of a k-restricted undirected component can be computed with the Dreyfus–Wagner algorithm [DW71] in polynomial time if k is a fixed constant. Moreover, in this case DCR^k can be solved in polynomial time. To this end, DCR^k can be re-formulated as a multi-commodity flow problem with polynomially many constraints. This can then be solved with a (strongly) polynomial-time linear programming algorithm [Byr+13]. Solving DCR exactly for $k = |V|$ is NP-hard in general [Goe+12].

The following (rephrased) theorem of Borchers and Du shows that solving the k-restricted LP exactly yields a good approximation.

Theorem 16.6.2 ([BD97]). *Let $k = 2^r + s$ with $r \in \mathbb{N}, s < 2^r$ be fixed. Let ρ_k denote the supremum among all Steiner tree instances of the ratio of the optimal value of DCR^k and the cost of an optimal Steiner tree. Then*

$$\rho_k = \frac{(r+1)2^r + s}{r2^r + s} \leq 1 + \frac{1}{\lfloor \log_2 k \rfloor}.$$

To obtain a $(1+\epsilon)$-approximation to DCR, one thus sets $k \in \Theta(2^{1/\epsilon})$. If, for example, one sets $k = 64$, then the ratio is rather reasonable with $7/6 = 1.1\bar{6}$. However, solving the LP then becomes too demanding in practice. Beyer and Chimani [BC14] report surprisingly good results for the algorithm by Goemans et al. [Goe+12] for $k = 3$ ($\rho_3 = 5/3$) in terms of both quality and runtime.

17 ITERATIVE RANDOMIZED ROUNDING

17.1 THE ALGORITHM OF BYRKA ET AL.

Algorithm 17.1, due to Byrka et al. [Byr+10; Byr+13], solves *DCR* approximately (via DCR^k), chooses a directed component C with probability proportional to value x_C, and *contracts* it. This is repeated on the contracted graph until a single terminal remains. A contraction

Algorithm 17.1: $(\ln 4 + \epsilon)$-approximation algorithm for the Steiner tree problem in graphs by Byrka et al.

```
Input: An edge-weighted simple graph $G = (V, E)$ and $\epsilon > 0$.
Output: A collection of full components whose union is a
        Steiner tree.
for $t = 1, 2, \dots$ do
    Compute a $(1 + \epsilon/2)$-approximate solution $x$ to $DCR$
    $C^t \leftarrow$ draw one component $C \in \mathcal{C}_k$ with probability $\frac{x_C}{\sum_{C' \in \mathcal{C}_k} x_{C'}}$
    $G \leftarrow G/C^t$
    if $|V| = 1$ then
        return $\bigcup_{i=1}^{t} C^i$
```

G/C of a directed component C collapses the vertices of C into the sink of C. The sink inherits the edges incident to C. If this results in multiple edges from the sink to the same vertex outside C, all but the cheapest are removed.

The algorithm can be derandomized [Byr+13]. The randomized algorithm is also treated in a recent edition of a textbook by Korte and Vygen [KV18]. Goemans et al. [Goe+12] show how to keep a feasible solution to the LP throughout the algorithm in order to avoid re-solving the LP. This establishes the approximation factor as an integrality gap upper bound for *HYP*. Additionally, it provides a computational advantage, which is the reason why this variant was chosen by Beyer and Chimani for an implementation [BC14]. Note that the number of terminals decreases with every iteration, hence the algorithm runs in polynomial time.

A Steiner tree S, without loss of generality a full Steiner tree, can be turned into a rooted full (but possibly noncomplete) binary tree of the same cost by introducing dummy vertices and zero-cost edges such that its leaves are exactly the terminals [KZ97; Byr+13]. The height of this tree can be shown to be at most $|R| - 1$. We will assume in the following that the optimum Steiner tree S^* is such a tree; the upper bounds we will obtain are upper bounds for the original tree. It helps for an easier understanding to imagine that S^* is complete.

Let us review the analysis of Byrka et al. As $\epsilon > 0$ is fixed, there is a number M polynomial in the input size[1] such that

$$\sum_{C \in \mathcal{C}^t} x_C^t \leq M,$$

and we can require this to hold with equality in our analysis by introducing a zero-cost dummy component $\overline{C}$ into $\mathcal{C}$ that consists of the root terminal. The corresponding variable $x_{\overline{C}}$ will provide the missing probability mass. The modified algorithm may sample this component, and contraction of it does not change the graph. Hence, the algorithm could perform some idle iterations (it could potentially run endlessly), but has the same expected objective value. Furthermore, the expected number of iterations until an edge is contracted is bounded from above by that number in the modified algorithm.

Let us consider the expected cost of the solution the modified algorithm returns. Let OPT^t and OPT_f^t denote the cost of the minimum Steiner tree and the cost of the optimum fractional DCR solution in the t-th iteration, respectively. We state the relevant sampling iterations in the subscript of the expected values[2] and in the superscript of $x_C^t, \mathcal{C}_k^t$, the solutions and the set of k-restricted components on the contracted graph in iteration t.

$$\begin{aligned}
\sum_{t \geq 1} \mathbf{E}_t \left[c(C^t) \right] &\leq \sum_{t \geq 1} \mathbf{E}_{t-1} \left[\sum_{C \in \mathcal{C}_k^t} \frac{x_C^t}{M} c(C) \right] \\
&\leq \frac{1}{M} \sum_{t \geq 1} \mathbf{E}_{t-1} \left[(1+\epsilon/2) OPT_f^t \right] \\
&\leq \frac{1+\epsilon/2}{M} \sum_{t \geq 1} \mathbf{E}_{t-1} \left[OPT_f^t \right] \\
&\leq \frac{1+\epsilon/2}{M} \sum_{t \geq 1} \mathbf{E}_{t-1} \left[OPT^t \right]. \qquad (17.1)
\end{aligned}$$

Let S^* denote an optimal (without loss of generality, binary) Steiner tree for the instance with cost $OPT = \sum_{e \in S^*} c(e)$. The *deletion time* $D(e)$ of an edge $e \in S^*$ is the number of iterations the algorithm takes until the edge is removed by a contraction. After all edges of S^* have been deleted, it is guaranteed the graph has been contracted into a single terminal. Let $S^1 = S^*$, and let S^t be S^{t-1} minus the edges in the t-th sampled component (and possibly duplicate edges that are eliminated). For any $t' \geq 1$ the graph

$$S^{t'} \cup \bigcup_{t=1}^{t'-1} C^t$$

1 Chakrabarty et al. [CKP10a] show that the support of a basic feasible solution has cardinality at most $|R|$, hence this can be used as an upper bound.

2 $\mathbf{E}_0[\cdot]$ means that no previous sampling has occured.

spans R. For the cost, we have

$$\begin{aligned}\sum_{t\geq 1}\mathbf{E}_t[c(C^t)] &\overset{(17.1)}{\leq} \frac{1+\epsilon/2}{M}\sum_{t\geq 1}\mathbf{E}_{t-1}[OPT^t] \\ &\leq \frac{1+\epsilon/2}{M}\sum_{t\geq 1}\mathbf{E}_{t-1}[c(S^t)] \\ &= \frac{1+\epsilon/2}{M}\sum_{e\in S^*}\mathbf{E}[D(e)]c(e), \end{aligned} \tag{17.2}$$

since an edge e is part of S^t if and only if it survives the first $t-1$ iterations. If we can bound $\mathbf{E}[D(e)] \leq \alpha \cdot M$ for every $e \in E$ for some $\alpha \leq 2$, we get

$$\sum_{t\geq 1}\mathbf{E}_t[c(C^t)] \overset{(17.2)}{\leq} \frac{1+\epsilon/2}{M}\frac{\alpha}{M}\sum_{e\in S^*}c(e) \leq (\alpha+\epsilon)OPT, \tag{17.3}$$

the desired approximation factor.

In order to keep track of the edges that are deleted in the course of the algorithm, an artificial *witness tree* that spans the terminals will be constructed for the analysis. For now, let W be any choice of such a tree. An example can be seen in Figure 17.1a on page 193. Each edge $e = (u,v) \in S^*$ partitions the terminals R into two sets, the terminals R_u that are connected by the subtree of S^* rooted at u and the terminals R_v connected by the subtree of S^* rooted at v. The *witness edges* $W(e) \subseteq W$ are the edges in the witness tree that go between terminals of R_u and R_v, hence they model how e connects the terminals. More precisely,

$$W(e) := \{uv \in W \mid e \text{ is on the path from } u \text{ to } v \text{ in } S^*\}.$$

When a component is sampled and contracted by the algorithm, a corresponding (random) subset of W is marked for the analysis. As soon as all edges $W(e)$ have been marked, we can conclude that $e \in S^*$ has been contracted, which has brought us closer to termination. The expected time until this happens is an upper bound for $\mathbf{E}[D(e)]$.

Given a directed component C with terminals S, which witness edges can be marked that correspond to the connectivity that C provides?

Definition 17.1.1. For a witness tree W and $R' \subseteq R$, the family of *bridge sets* is

$$\mathcal{B}_W(R') := \{B \subseteq W \mid (W \setminus B)/R' \text{ is a tree}\}.$$

The following technical lemma, absent from [Byr+13], establishes correctness of the marking procedure.

Lemma 17.1.2 ([KV18]). *Let $W =: W_0$ be a tree on R, $t \in \mathbb{N}$, and $C^1, \ldots, C^t \in \mathcal{C}^k$. For $i = 1, \ldots, t$, let $B_i \in \mathcal{B}_{W_{i-1}/C_1/\cdots/C_{i-1}}(R)$ and*

$W_i := W_{i-1} \setminus B_i$. *Then all terminals are in the same connected component of*

$$\{e \in E(S^*) \mid W(e) \cap W_t \neq \emptyset\}/C_1/\ldots/C_t.$$

Byrka et al. consider a random variable W, i.e., the witness tree is chosen from a certain probability distribution of trees on $\binom{R}{2}$. In our opinion, both the reasoning behind the overall analysis and the specific choice of the witness tree distribution of Byrka et al. are rather obscure. We therefore give an attempt at an intuitive explanation and will postpone specific witness tree distributions to the following sections. A witness tree chosen deterministically without regard to the cost of the edges in S^* would be prone to an 'adversary' that chooses the edge cost in order to maximize deletion times of costly edges. Thus for an analysis that does not assume anything about the edge cost, the risk is spread out over a distribution of witness trees. Note that a witness tree edge influences the size $|W(e)|$ of several edges $e \in S^*$.

Byrka et al. choose a witness tree distribution that we shall describe as the result of a greedy randomized algorithm. This algorithm prefers edges for W that connect terminals close to each other. This ensures that overall, the sizes $|W(e)|$ are small with a high probability for each $e \in S^*$. However, $W(e)$ can be large with a small probability as well, this helps to keep $W(e')$ small for other edges $e' \neq e$. We shall see in Theorem 17.1.5 that the expected time until all edges in a subset of W are marked depends *logarithmically* on the size of the set: Because the logarithm increases slowly, we should assign smaller, but non-zero probabilities to larger sets for an optimal balance.

Byrka et al. never state why they choose their specific distribution, and if it is (in the cost-oblivious sense) optimal, at least asymptotically with growing tree height.

Once the distribution has been chosen, Byrka et al. show that every edge $e \in W$ is marked with probability at least $1/M$ in an iteration of the algorithm. It is important to stress that the algorithm does not mark the edges of W. Rather, the random marking is performed in the analysis in such a way that the probability bound of $1/M$ holds. As we understand it, this is an existential argument that the randomness of the algorithm can be captured with respect to an arbitrary witness tree distribution by choosing suitable distributions for the marking procedure (which are never explicitly used in the analysis).

In order to do so, we need to relate the cost of a terminal spanning tree (in particular, our witness tree W) to the cost of any fractional *DCR* solution.

Lemma 17.1.3 (Bridge Lemma, [Byr+13])**.** *Let W be a terminal spanning tree and x be a feasible solution to DCR. Let*

$$br_W(C) := \max\{c(B) \mid B \in \mathcal{B}_W(R(C))\}.$$

Then

$$c(W) \leq \sum_{C \in \mathcal{C}} x_C br_W(C).$$

A sketch of the proof is as follows. First, a terminal spanning tree is constructed for every component C of weight $br_W(C)$. From these terminal spanning trees, a fractional solution $y : R \times R \to \mathbb{R}_0^+$ to BCR of cost

$$c(y) = \sum_{e \in R \times R} w(e)y(e) = \sum_{C \in \mathcal{C}} x_C br_W(C)$$

is constructed. By Theorem 16.5.2, BCR is integral for terminal spanning trees, hence there is an integral optimal BCR solution, i.e., a terminal spanning tree T. The cost of T can be shown to be a most $c(y)$.

We now give the existential lemma for the marking procedure.

Lemma 17.1.4 ([Byr+13]). *Let $W \subseteq \binom{R}{2}$ be a tree on R. There is a probability distribution*

$$\overline{Br}_W : \mathcal{B}_W(C) \to [0,1]$$

such that each edge $e \in W$ is marked with probability at least $1/M$ in each iteration.

The proof is by contradiction, we give a sketch. Assuming there are no such probability distributions $\overline{Br}_W(\cdot)$, Farkas's Lemma (Corollary 2.11.7) would imply the existence of a vector $(y,c) \in \mathbb{R}^{|\mathcal{C}| \times |W|}$ such that

$$\sum_{C \in \mathcal{C}} x_C br_W(C) \leq \sum_{C \in \mathcal{C}} y_C < \sum_{e \in W} c_e = c(W),$$

contradicting the Bridge Lemma 17.1.3.

Using the previous lemma, it is shown that all edges of $W(e)$ are marked within a small number of iterations. This is a variant of the Coupon collector problem (see Subsection 2.4.1):

Lemma 17.1.5 ([Byr+13]). *Let $\tilde{W} \subseteq W$. For the number $X(\tilde{W})$ of iterations until all edges in $\tilde{W}$ are marked, we have*

$$\mathbf{E}[X(\tilde{W})] \leq H_{|\tilde{W}|} \cdot M, \tag{17.4}$$

where H_n is the n-th Harmonic number.

The proof is by induction over $|\tilde{W}|$. We now have the necessary tools to prove approximation factors for a given witness tree distribution. For convenience in the following sections, we provide Table 17.1 on the next page with the first six values of H_n.

Table 17.1: The first six harmonic numbers.

Harmonic Number	H_1	H_2	H_3	H_4	H_5	H_6
Fraction	1	$\frac{3}{2}$	$\frac{11}{6}$	$\frac{25}{12}$	$\frac{137}{60}$	$\frac{49}{20}$
Decimal value	1	1.5	$1.8\overline{3}$	$2.08\overline{3}$	$2.28\overline{3}$	2.45

17.2 WITNESS TREE DISTRIBUTIONS

Byrka et al. choose the following witness tree distribution. Recall that S^* is a full binary tree of height $h \leq |R| - 1$. For every Steiner vertex in S^*, choose one of its edges to its two children with equal probability. Let B denote the set of chosen edges, and let P_{uv} denote the unique path from u to v in S^*. The witness tree is chosen as

$$W = \left\{ uv \in \binom{R}{2} \mid |P_{uv} \cap B| = 1 \right\}. \tag{17.5}$$

An example is given in Figure 17.1a on the facing page. Such a witness tree W on a complete Steiner tree of a given height h always has the same structure up to isomorphism. In particular, the set of witness set sizes $\{|W(e)|\}_e$ of edges e on a given level l are the same for every complete Steiner tree.

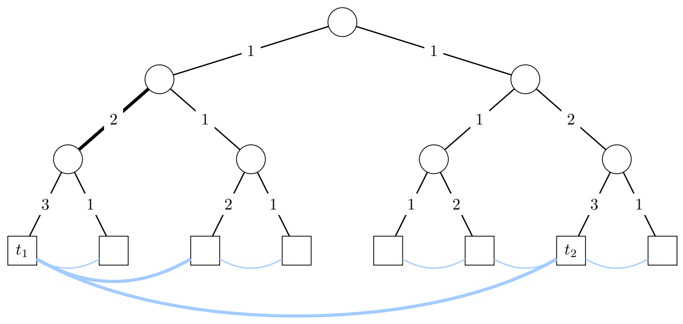
Lemma 17.2.1 ([Byr+13]). *Let W be chosen on a Steiner tree S^* as in Equation (17.5). Then for every edge $e \in S^*$ at level $l_e \leq |R| - 1$ (where edges incident to the root are at level one), we have*

$$\Pr[|W(e)| = q] = \begin{cases} 1/2^q, & \text{if } 1 \leq q < l_e, \\ 2/2^q, & \text{if } q = l_e, \\ 0, & \text{otherwise.} \end{cases} \tag{17.6}$$

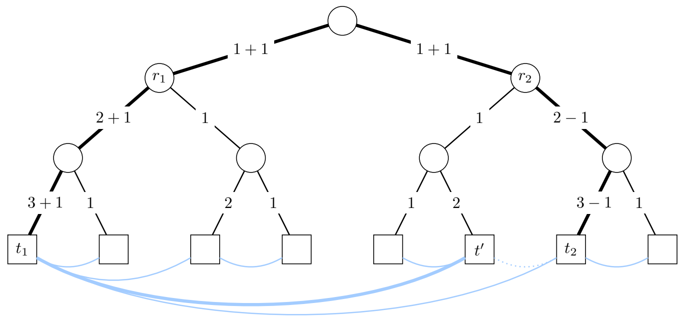
We give an alternative way of generating this distribution.

Proof for complete Steiner trees S^*. We describe a randomized greedy algorithm to construct a witness tree W, and use the following nomenclature. Let S be a proper subtree of S^* rooted at r_S, and let p denote the parent of r_S. The subtree S' rooted at p's other child is the *neighboring subtree* of S.

The algorithm starts at the leaves of S^* (the terminals) and works upward through the tree. Every subtree S is linked to its neighboring subtree S' by a W-edge between a leaf of S and a leaf of S'. Between all leaves of S with maximum degree in the current W, we choose one uniformly at random, the same is done for S'. Once all subtrees at level l have been connected to their neighbor, we proceed with the



(a)



(b)

Figure 17.1: (a) A witness tree (blue arcs) of the Byrka distribution for an optimal binary Steiner tree S^*. The cardinalities $W(e)$ are shown on the edges of S^*. One edge $e \in S^*$ is shown in bold, the set $W(e)$ of its witness edges is also shown in bold. (b) A modification of the witness tree as described in the proof of Theorem 17.2.6 is shown. The dotted witness edge (t_2, t') is removed and the bold edge (t_1, t') is added. The total witness set sizes per level stay the same except for the topmost level, and they are favorable.

subtrees at level $l - 1$ in the same fashion. There are exactly two leaves of maximum degree in W in every subtree (not counting a single terminal as a subtree) before the next step. Note that the lowest level is chosen deterministically: there is only one terminal in the subtree, which is linked to the neighboring terminal via an edge.

Since the tree is complete, the sizes of $W(e)$ are identically distributed for all edges e on the same level. This follows inductively from the construction: If we have identical distributions on a level l, then the uniform selection process ensures this on level $l - 1$. As the lowest level is chosen deterministically, its set sizes are identically distributed in particular.

The probabilities $\Pr[|W(e)| = q]$ in the complete case are now easily calculated from the following argument: With every level l, the witness set sizes are set to one for level l by the construction in the algorithm, and the set sizes on the levels below increase by one on exactly half the edges whose set size is maximum on that level in the current W. Therefore, the probabilities are as claimed. □

The rationale behind the algorithm in the above proof is as follows. An edge uv in W that has a long path P_{uv} in S^* is undesirable, as it adds to the size of many sets $W(e)$. Hence, neighboring subtrees are chosen in order to keep the pathlength small. Which terminal should be chosen for connecting two subtrees? The pathlengths are the same for all choices if S^* is complete. As the time until all edges in a set $\tilde{W} \subseteq W$ are marked increases logarithmically with the size of the set by Lemma 17.1.5, we should greedily select the vertices that have the highest degree.

It is easy to see from the iterative construction that the topmost k levels of the (transformed) Steiner tree S^* have the same witness set sizes as a Steiner tree of height k, assuming completeness for both. The lowest level has the highest witness set sizes on average, and hence determines the maximum expected deletion time. Note also that the maximum degree in W is exactly the height h of the tree.

Let us now prove the main result of Byrka et al.

Theorem 17.2.2 ([Byr+13]). *There is a randomized polynomial-time approximation algorithm for the Steiner tree problem in graphs with expected approximation factor* $\ln(4) + \epsilon < 1.3863 + \epsilon$ *for every* $\epsilon > 0$.

Proof. From complex analysis one knows the function

$$Li_1(z) = \ln(1/(1-z)) = \sum_{q \geq 1} z^q q^{-1}.$$

Therefore,

$$\sum_{q \geq 1} \frac{1}{2^{q-1}} q^{-1} = 2 \ln\left(\frac{1}{1-\frac{1}{2}}\right) = \ln(4). \tag{17.7}$$

Let $D(e) = \max\{t \mid e \in S^t\}$ denote the iteration in which e is deleted. By Lemmata 2.4.8 and 17.1.5 the expected deletion time of $e \in S^*$ is

$$\begin{aligned}
\mathbf{E}[D(e)] &= \sum_{q=1}^{k_e} \Pr[|W(e)| = q] \mathbf{E}[D(e) \mid |W(e)| = q] \\
&\leq \sum_{q=1}^{k_e} \Pr[|W(e)| = q] H_q M \\
&\overset{(17.6)}{\leq} \sum_{q=1}^{k_e} \left(\frac{1}{2}\right)^q H_q M + 2/2^{k_e} H_{k_e} M \\
&\leq \sum_{q \geq 1} \left(\frac{1}{2}\right)^q H_q M \\
&= M \sum_{q \geq 1} \frac{1}{q} \sum_{i \geq 0} \left(\frac{1}{2}\right)^{q+i}
\end{aligned}$$

$$= M \sum_{q \geq 1} \frac{1}{q} \left(\frac{1}{2}\right)^{q-1}$$
$$\overset{(17.7)}{=} M \ln(4).$$

Therefore, the expected cost of the solution returned by the modified algorithm satisfy

$$\mathbf{E}\left[\sum_{t\geq 1} c(C^t)\right] = \sum_{t \geq 1} \mathbf{E}_t[c(C^t)] \overset{(17.3)}{\leq} \frac{1+\epsilon/2}{M} \sum_{e \in S^*} \mathbf{E}[D(e)]c(e)$$
$$\leq (\ln(4)+\epsilon)OPT, \tag{17.8}$$

and this bound also holds for Algorithm 17.1 on page 187. □

The algorithm can be derandomized.

Theorem 17.2.3 ([Byr+13]). *There is a polynomial-time approximation algorithm for the Steiner tree problem in graphs with approximation factor* $\ln(4) + \epsilon < 1.3863 + \epsilon$ *for every* $\epsilon > 0$.

Byrka et al. further prove that 1.55 is an upper bound on the integrality gap of *HYP*. A short proof of this was provided by Chakrabarty et al. [CKP10b].

As described earlier, Goemans et al. modify the algorithm such that *DCR* is not re-solved after each contraction. Instead, a feasible solution to the original LP is maintained by a greedy procedure on a certain matroid. The approximation factor then establishes a gap upper bound.

Theorem 17.2.4 ([Goe+12]). *The integrality gap of DCR is at most* $\ln(4)$.

Since Byrka et al. do not provide insight how they came to chose their distribution in the general case, or if it is the optimal choice, we tried to come up with a better choice.

Proposition 17.2.5. *There is a witness tree distribution* $\tilde{W}$ *on a complete binary Steiner tree of four terminals with* $\max_{e\in E} \mathbf{E}[D(e)] = 32/26M < 1.231M$*, as opposed to the bound* $1.25M$*, which one obtains from* (17.4) *with the Byrka distribution.*

Proof. The distribution of Byrka et al. consists of four trees of probability $1/4$ each. In each such tree W, there are two 'terminal edges' (i.e., edges incident to a terminal) with $|W(e)| = 2$ and two 'Steiner edges' (i.e., incident to a Steiner vertex) with $|W(e)| = 1$ (see Figure 17.2a on page 197). Hence,

$$\mathbf{E}[D(e)] = M \cdot H_1 = M.$$

for Steiner edges and for edges e incident to a terminal, we have

$$\mathbf{E}[D(e)] = M\left(\frac{2}{4}H_1 + \frac{2}{4}H_2\right) = M \cdot \frac{5}{4} = 1.25M.$$

In addition to these trees, we consider the four trees where one terminal has degree three, each with probability $1/4$ (see Figure 17.2b on the next page). In each such tree W', there are three terminal edges with $|W'(e)| = H_1$ and one with $|W'(e)| = H_3$. The Steiner edges both have $|W'(e)| = H_2$. For edges incident to a terminal vertex, we thus have

$$\mathbf{E}[D(e)] = M\left(\frac{3}{4}H_1 + \frac{1}{4}H_3\right) = M \cdot \frac{29}{24} = 1.208\overline{3}M.$$

For edges between Steiner vertices, we have

$$\mathbf{E}[D(e)] = M \cdot H_2 = \frac{3}{2}M.$$

A convex combination of the two distributions yields a mixture distribution, which is advantageous.[3] Let us choose the Byrka distribution with probability x and with probability $(1-x)$ the latter degree-3 distribution. For terminal edges, we obtain

$$\frac{5}{4}x + \frac{29}{24}(1-x),$$

and for the Steiner edges, we have

$$x + \frac{3}{2}(1-x).$$

We equate the two, because if one of the two terms were strictly larger than the other, we could decrease the maximum of the two by shifting mass from or to x. By solving this equality for x one obtains the optimum objective $32/26 < 1.231$ at $x = 7/13$. □

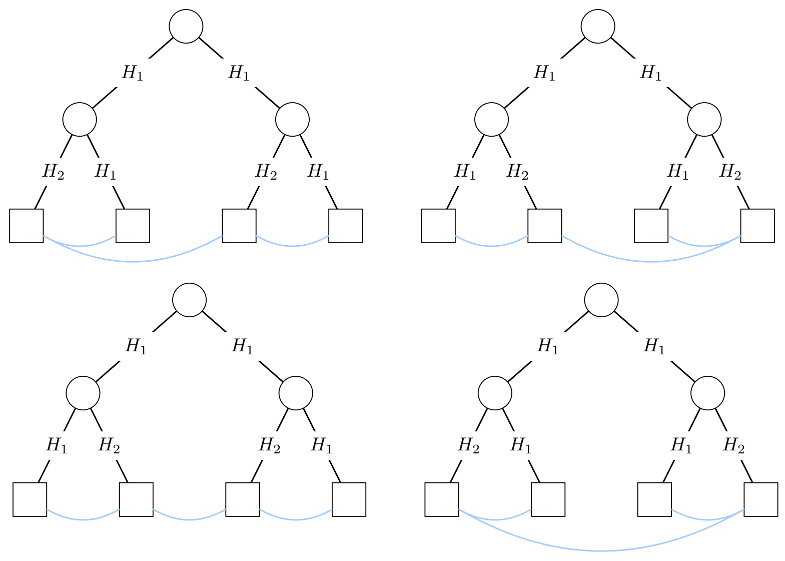
In fact, we can also describe a distribution which is better than the one by Byrka et al. for all heights (on complete trees), but only equally good in the limit.

Proposition 17.2.6. *For every complete binary Steiner tree S^* of height $h \geq 2$ there is a witness tree distribution $\tilde{W}$ for which the expected deletion time bound (17.4) of each edge is smaller than the maximum expected deletion time bound for the Byrka distribution.*

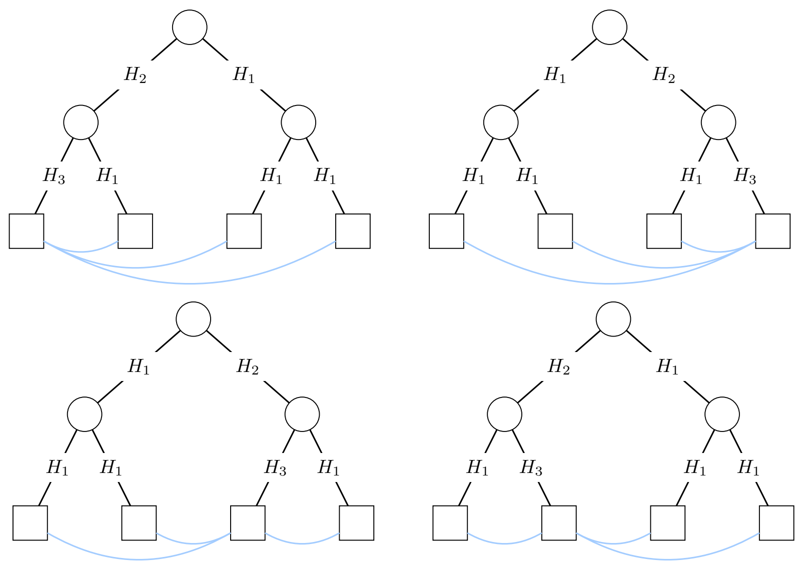
Proof. Consider the random witness tree W that the greedy randomized algorithm outputs. There are two terminals t_1, t_2 of maximum degree in W, and there is an edge $(t_1, t_2) \in W$. Note that the set sizes on the path from t_1 to the root and on the path from t_2 to the root are the same (see Figure 17.1a on page 193 for an example).

Let us modify the tree as follows for a distribution W' (Figure 17.1b on page 193). Choose one of the two said terminals uniformly at

3 There are $4^{4-2} = 16$ possible terminal spanning trees, but the remaining eight are clearly worse than the ones presented. Hence, by computing the optimum distribution on the eight trees mentioned, we obtain the optimum approximation factor one can get from a cost-oblivious analysis.



(a)



(b)

Figure 17.2: (a) The four equiprobable witness trees from the distribution proposed by Byrka et al. The expected number of iterations until all edges in $W(e)$ are marked, divided by M, is shown on each edge e. (b) Four witness trees where the expected deletion time is smaller on the edges of the lower level, yet larger on the edges of the upper level.

random, say t_2. It has an edge in W to a terminal t' whose degree is one less. Delete the edge (t_2, t') and add the edge (t_1, t') in W. From the logarithmic nature of the marking procedure (Lemma 17.1.5) it follows that the lowest level has improved because the largest set size was once increased and once decreased, and the other set sizes are unaffected. The levels above do change: Let T_2 be the smallest subtree that contains both t_2 and t', it has height $h-1$ by construction. The set sizes on the path from t_2 to its subtree root r_2 of T_2 decrease by one. The set sizes on the path from t' to r_2 do not change. Likewise, the set sizes on the path from t_1 to the root r_1 of the containing subtree T_1 increase by one. Again, this means that these levels improve because of the logarithmic nature of the expected deletion time. The topmost level remains: In the original distribution W, both set sizes are always one. They both increase to two, hence we have an expected deletion time of $H_2 \cdot M = 1.5M$ on each of them in W'.

We now create a mixture distribution $\tilde{W}$: Choose the modified distribution W' with some probability $0 < p < 2(\ln 4 - 1) < 0.773$, and the original one W by Byrka et al. with probability $1-p$. The optimum choice for $h = 2$ is exactly $p = 6/13$ (see the proof of Proposition 17.2.5). For $h \geq 4$, the maximum expected set size for W is greater than $4/3$. Choosing, say, $p = 2/3$ results in better expected deletion times on all levels in $\tilde{W}$: the expected sizes on the topmost level become $2/3 \cdot H_2 + 1/3H_1 = 4/3$. (Note that this is significantly better than the limit upper bound $\ln 4$ of the maximum expected size in W.) The lower levels improve slightly because $p > 0$. For $h = 3$, the maximum expected size for W is $4/3$, and p can be chosen suitably to improve over it. $\square$

Unfortunately, the improvement vanishes as the tree height grows to infinity. Is there hope to get an improvement in the limit? We think the answer is no. A heuristic argument is as follows. Let $\alpha < \ln 4$ be arbitrarily close. There is a complete binary tree of some (minimal) height h such that the expected deletion times on the lowest level are at least αM in the Byrka distribution. Now increase the tree height further, say, to $h' = 2^{2^h}$. Now only the first h levels have an average expected deletion time less than αM, as the average set sizes increase with every level. This means that if we wish to modify W in order to get the average expected deletion time below αM on the lower levels, we would have to move the burden to the h topmost levels as in Propositions 17.2.5 and 17.2.6. However, there are only 2^h edges on these levels, and there are two edges in W of maximum degree 2^{2^h}. There are also two edges of degree $2^{2^h} - 1$, four of degree $2^{2^h} - 2$, eight of degree $2^{2^h} - 3$ etc. Hence, it seems out of the question that the h upper levels can increase their set sizes to allow for a (significant) decrease on the lower levels.

Of course, the argument would apply to any distribution with similar properties. The greedy recursive nature of the algorithm could lead to a proof that every distribution with such properties must be almost identical to the constructed one, such as our example in Proposition 17.2.6. This, however, seems much more difficult to us than the usual optimality proofs for greedy algorithms.

The argument is similar if we are not interested in the maximum expected deletion time, but the total expected deletion times, where the same approximation factor is approached in the limit. The total expected deletion times are of interest for unweighted instances, as we shall see in the following section.

17.3 quasi-bipartite and claw-free instances

Fortunately, it is possible to choose better distributions if S^* has a special structure, as is the case in quasi-bipartite and claw-free instances. For quasi-bipartite instances, Byrka et al. describe a witness tree distribution that yields a better approximation factor of $1.21\overline{6} + \epsilon$. Note the following equivalence.

Theorem 17.3.1 ([CKP10a; Goe+12]). *HYP and BCR are polyhedrally equivalent on quasi-bipartite instances.*

While Chakrabarty et al. [CKP10a] show the two relaxations have the same optimum objective value, they did not give a cost-preserving polynomial-time conversion from a *BCR* solution to a *HYP* solution. Therefore, their approach does not directly help in solving *DCR* on quasi-bipartite instances. Goemans et al. [Goe+12] and independently, Fung et al. [Fun+12], show that *DCR* can be solved exactly in polynomial time on quasi-bipartite instances by solving *BCR* and converting the solution to a *DCR* solution of the same value. This allows elimination of the ϵ in the approximation factor of the randomized algorithm.[4] However, an $\epsilon > 0$ is also introduced in the derandomization of the algorithm in [Byr+13]. It is unclear to us whether it can also be removed in the deterministic algorithm for quasi-bipartite instances.

Theorem 17.3.2 ([Byr+13] with [Goe+12; Fun+12]). *For quasi-bipartite instances there is a randomized polynomial-time approximation algorithm with expected approximation factor $73/60 = 1.21\overline{6}$. Moreover, there is a polynomial-time approximation algorithm with factor $73/60 + \epsilon$ for every fixed $\epsilon > 0$.*

Proof. Consider the components of an optimum Steiner tree S^* on a quasi-bipartite instance (we do not use the transformation procedure into a binary tree here). Unless a component consists of two terminals connected by a single edge, it is a star with a Steiner vertex at its center (see the leftmost component in Figure 16.3 on page 184). For the latter, let k denote the number of terminals in the star. Choose one terminal

4 This was not available to Byrka et al. in their conference paper [Byr+10], and it was apparently overlooked in the later journal version [Byr+13].

uniformly at random as a 'hub'. Connect it to all other terminals in the witness tree W. We now have $|W(e)| = k-1$ with probability $1/k$ and $|W(e)| = 1$ with probability $1 - 1/k$ for each edge e of the star. Hence,

$$\mathbf{E}[D(e)] \leq \left(\frac{1}{k}H_{k-1} + \left(1 - \frac{1}{k}\right)H_1\right)M = \left(\frac{1}{k}H_{k-1} + \frac{k-1}{k}\right)M,$$

which attains its maximum at $k = 4$. Therefore $\mathbf{E}[D(e)] \leq \frac{73}{60}M$.

If the component consists of two terminals t_1, t_2 (see the rightmost component in Figure 16.3 on page 184), we add the edge $e = (t_1, t_2)$ to W, hence $|W(e)| = 1$ and $\mathbf{E}[D(e)] = 1$. The proof by Byrka et al. contains a subtle mistake here, because it states for every component that all edges but one are selected for the set $\tilde{B}$, and by (17.5), the edge (t_1, t_2) would not be part of W. Instead, terminals from other components would connect to t_1 and t_2 in W, which would make the expected deletion times worse.

Again, the algorithm can be derandomized [Byr+13, Theorem 28], which introduces a loss in the approximation factor of $(1+\epsilon)$ for a choice of $\epsilon > 0$. □

Due to the simplicity of quasi-bipartite instances, it is evident that the chosen witness tree distribution is optimal in the cost-oblivious setting. The integrality gap result of Goemans et al. also applies for quasi-bipartite instances.

Theorem 17.3.3 ([Goe+12]). *BCR has an integrality gap of at most* $73/60$ *for quasi-bipartite instances.*

Prior to this theorem, Rajagopalan and Vazirani showed that the integrality gap in quasi-bipartite instances is at most $3/2$ [RV99]. Their algorithm achieves a $(3/2+\epsilon)$-approximation. Gröpl et al. [Grö+02] gave a combinatorial $73/60$-approximation algorithm for *uniformly* quasi-bipartite instances. Based on this algorithm, Chakrabarty et al. [CKP10a] show that in this scenario, the integrality gap of *HYP* (and hence also *BCR*) is at most $73/60$. The same authors [CKP10b] show an integrality gap bound of α for quasi-bipartite instances where $\alpha = 1 + e^{-\alpha}$, i.e., $\alpha < 1.279$. Fung et al. [Fun+12] give a simple sampling algorithm that achieves an expected approximation factor of α for quasi-bipartite instances.

Inspired by the previous theorem, let us turn to claw-free instances. We will show an approximation factor between $73/60$ and $\ln(4)$. Feldmann et al. prove a stronger version of Theorem 17.3.1.

Theorem 17.3.4 ([Fel+16]). *HYP and BCR are polyhedrally equivalent on instances without Steiner claws.*

The equivalence cannot be extended much further under reasonable assumptions because in the same paper it was proved that deciding

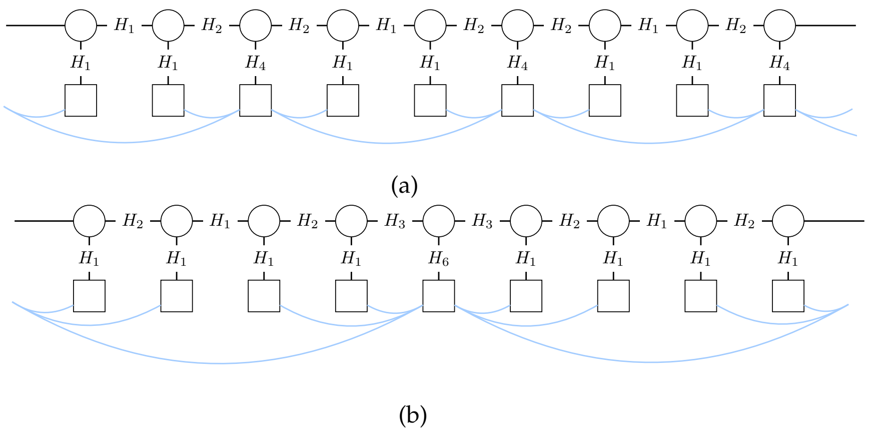

H_1 H_2 H_2 H_1 H_2 H_2 H_1 H_2
H_1 H_1 H_4 H_1 H_1 H_4 H_1 H_1 H_4
(a)
H_2 H_1 H_2 H_3 H_3 H_2 H_1 H_2
H_1 H_1 H_1 H_1 H_6 H_1 H_1 H_1 H_1
(b)

Figure 17.3: (a) A snippet of a periodic witness tree W (blue arcs) on the terminals (squares) of a Steiner tree in a claw-free instance. The expected deletion times divided by M are shown on each edge for this W. The witness tree is called a 4-hub because of the terminals with four emanating witness tree edges. (b) A snippet of a similarly defined periodic witness tree called a 6-hub.

equivalence of *HYP* and *BCR* for a given instance is NP-hard, even when the Steiner vertices form a single star.

The proof of Theorem 17.3.4 uses an efficiently computable cost-preserving map from feasible *BCR* solutions to feasible *HYP* solutions. Hence, we can again get rid of ϵ in the randomized algorithm by solving *HYP* exactly via *BCR*. We will next choose a different witness tree distribution tailored to claw-free instances. Sanità seems to remember[5] that an approximation factor smaller than $\ln 4$ can be obtained, but the choice of distribution and the factor have apparently been lost.

Theorem 17.3.5. *For claw-free instances, there is a randomized polynomial-time approximation algorithm with expected approximation factor* $991/732 < 1.354$. *Moreover, there is a polynomial-time approximation algorithm with approximation factor* $991/732 + \epsilon$ *for every* $\epsilon > 0$.

Proof. Use the algorithm of Byrka et al. with the modification that *HYP* is solved exactly. In order to achieve this, solve *BCR* in polynomial time and use the conversion by Feldmann et al. [Fel+16].

As a Steiner vertex has at most two Steiner neighbors, and S^* is acyclic, the Steiner vertices in each (undirected) component form a path (see Figure 16.3 on page 184). We modify the optimal Steiner tree S^* in a similar fashion as in Section 17.1. For every Steiner vertex with no adjacent terminal, delete it from the path by contracting it and its two edges into a single edge whose weight equals the sum of the two edges. (A Steiner end vertex without an adjacent terminal can be simply removed. This can only happen if it is connected to the path with zero weight as S^* is an optimal Steiner tree.) For every Steiner

5 Laura Sanità, personal communication, November 2017.

vertex on the path with more than one terminal, expand the Steiner vertex to a path of Steiner vertices, each having a single adjacent terminal. The edge weights between these Steiner vertices are set to zero. Now we have a path of Steiner vertices where each Steiner vertex has exactly one terminal adjacent to it (Figure 17.3a). (For components with exactly two terminals, we select the edge between them for W and obtain $\mathbf{E}[D(e)] = 1M$ as in the proof of Theorem 17.3.2.)

For the sake of a simpler analysis, let us imagine that the path is infinitely long. The edges near the ends of the path cannot have greater expected deletion times, hence this assumption will provide us with an upper bound.

We choose a probability distribution of witness trees W with the following idea. Pick a tree on the terminals that is periodic: If we shift it by its period along the path, we obtain the same tree. For period P, the probability for each of the trees shifted by $\{0, \ldots, P-1\}$ is chosen as $1/P$. This means that the expected deletion times of all edges between Steiner vertices are identical, and the expected deletion times of the edges between Steiner and terminal vertices are equal to each other as well.

Our first attempt at a distribution is as follows and can be seen in Figure 17.3a. Some terminals dangling from the path will be *hubs* with degree four in W. (Only near the end of the path, this degree may be smaller, the necessary modifications are straightforward.) A hub h has one edge to each of the terminals immediately to the left and right. Moreover, h has one edge to each of the hubs left and right, which are immediately left and right of the aforementioned 'neighboring' terminals.

It is evident that (roughly) one third of the vertices are hubs. Fix one Steiner vertex on the path. With probability $1/3$, turn its adjacent terminal into a hub as described above, and continue to declare terminals as hubs on the path, each with distance three (counting only edges between Steiner vertices) from the previous as described above.

For the next tree, shift the choice of hubs one to the right along the path. We choose this tree with probability $1/3$ as well, and then we shift another time, again we select this tree with probability $1/3$. We call this the 4-hub distribution after the number of edges emanating from each hub.

Let us now calculate the probabilities for the cardinalities $|W(e)|$. (The cardinalities are smaller near the ends of the path, hence there is no need for discussing them.) For the edges to the terminals, we have

$$\Pr[|W(e)| = 1] = \frac{2}{3}, \qquad \Pr[|W(e)| = 4] = \frac{1}{3}.$$

For the edges between Steiner vertices, we have

$$\Pr[|W(e)| = 1] = \frac{1}{3}, \qquad \Pr[|W(e)| = 2] = \frac{2}{3}.$$

We now apply Lemma 17.1.5. For edges incident to a terminal vertex, we have

$$\mathbf{E}[D(e)] = M\left(\frac{2}{3}H_1 + \frac{1}{3}H_4\right) = \frac{49}{36} \cdot M = 1.36\overline{1} \cdot M.$$

For edges between Steiner vertices, we have

$$\mathbf{E}[D(e)] = M\left(\frac{1}{3}H_1 + \frac{2}{3}H_2\right) = \frac{4}{3} \cdot M.$$

This would give us an approximation factor of $1.36\overline{1}$ by following the proof of Theorem 17.2.2. However, we choose another distribution of trees and turn the two distributions into a mixture distribution by a convex combination.

Our second '6-hub' distribution follows the same idea as the first, but here the hubs have six edges each. Each hub has edges to two terminals to its left and two to its right, and an edge to the hub left and the hub to the right. This is depicted in Figure 17.3b on page 201. We shift to obtain five different trees, each is selected with probability $1/5$. We again omit the discussion near the ends of a path. For the edges to the terminals, we have

$$\Pr[|W(e)| = 1] = \frac{4}{5}, \qquad \Pr[|W(e)| = 6] = \frac{1}{5},$$

and therefore

$$\mathbf{E}[D(e)] = M\left(\frac{4}{5}H_1 + \frac{1}{5}H_6\right) = \frac{129}{100} \cdot M.$$

For the edges between Steiner vertices, we have

$$\Pr[|W(e)| = 1] = \frac{1}{5}, \quad \Pr[|W(e)| = 2] = \frac{2}{5}, \quad \Pr[|W(e)| = 3] = \frac{2}{5},$$

and thus

$$\mathbf{E}[D(e)] = M\left(\frac{1}{5}H_1 + \frac{2}{5}H_2 + \frac{2}{5}H_3\right) = \frac{23}{15} \cdot M.$$

Note that $23/15 = 1.5\overline{3}$, which is worse than in the 4-hubs distribution.

Now let us choose with probability x the 4-hub distribution and with probability $(1-x)$ the latter 6-hub distribution. For terminal edges, we obtain

$$\frac{49}{36}x + \frac{129}{100}(1-x),$$

and for the Steiner edges, we have

$$\frac{4}{3}x + \frac{23}{15}(1-x).$$

As in the proof of Theorem 17.2.5, we equate the two. By solving this equality for x one obtains the optimum objective $991/732 < 1.354$ at $x = 219/244$.

The derandomization of Byrka et al. goes through unscathed, but a loss in the approximation factor of $(1+\epsilon)$ is introduced for a choice of $\epsilon > 0$. □

We were not able to prove that our choice of distribution in Theorem 17.3.5 is the best possible for claw-free instances. A linear program solver was used to compute the optimum choice for mixture distributions of k-hubs where $k \in \{2, 3, \dots, 100\}$ (i.e., also odd k, where we consider two equiprobable symmetrical trees), but this does not yield anything better than our manual choice of 4- and 6-hubs. Of course, this does not preclude the existence of a better distribution on the set of all spanning trees. We note that Cayley's formula states that the number of spanning trees on n terminals is n^{n-2}. Most spanning trees, however, are not useful because the pathlengths are long and thus cause large witness set sizes. One could try to prune the search space. For example, it appears that 'crossing' edges in W, i.e., two edges uv and xy with $u < x < v < y$, can be replaced by ux and vy, which contributes less to the witness set sizes.

We conjecture that the approximation factor in Theorem 17.3.5 carries over to the integrality gap and that this can be proved with the methods by Goemans et al. [Goe+12]. The authors mention that their results can be formulated in terms of linear progamming theory. However, their extended version is still in the making, and it is unclear whether a different analysis of Algorithm 17.1 automatically bounds the integrality gap as well[6]. Note that the authors did treat the case of quasi-bipartite instances separately from the general case, so possibly, it does not follow directly.

Let us now turn to unweighted instances in the claw-free case. Könemann[7] suggested that it might be possible to refine the analysis by taking edge costs into account: the maximum expected deletion time among the edges is used in all previous estimates (see (17.8)). For example, it could be possible to choose a witness tree distribution where the expected deletion time is small for costly edges and large for cheap edges. We will now show how uniform edge weights can be exploited.

Theorem 17.3.6. *There is a randomized polynomial-time approximation algorithm for the Steiner tree problem on unweighted, claw-free instances with expected approximation factor* 1.25.

Proof. In an instance with unit weights, the cost of the optimum Steiner tree S^* equals the number $|S^*|$ of its edges. As in the proof of Theorem 17.3.5, we transform each component of S^* such that each

6 Rico Zenklusen, personal communication, April 2019.
7 Jochen Könemann, personal communication, December 2017.

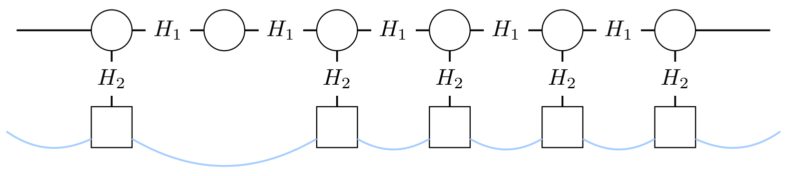


Figure 17.4: A witness tree (blue arcs) on the terminals (squares) of a Steiner tree in a claw-free instance. The expected deletion times (divided by M) are shown for this witness tree. At least half the edges have an expected deletion time of $H_1 M$, the others have a deletion time of $H_2 M$. The deterministic choice of witness tree shown is well-suited for unweighted claw-free instances.

Steiner vertex is adjacent to *at most one* terminal. This may introduce zero-weight edges, but this is not problematic because we will bound their weight with one in the analysis. We do not, however, contract paths of Steiner vertices that are not adjacent with terminals in order to obtain exactly one adjacent terminal: The edge weights between these Steiner vertices could add up to more than one. As before, we get rid of ϵ by solving *DCR* exactly via *BCR*.

The choice of witness tree W is deterministic and simply a path from the leftmost terminal to the rightmost terminal (Figure 17.4 on this page). Again, we imagine the path to be of infinite length. Each terminal edge has $\mathbf{E}[D(e)] = 3/2 \cdot M$ (except at the ends of the path where $\mathbf{E}[D(e)] = M$). The edges between Steiner vertices have $\mathbf{E}[D(e)] = M$, regardless of whether the vertex is adjacent to a terminal. We can conclude that edges with $\mathbf{E}[D(e)] = M$ account for at least half the edges in the transformed tree S^*.

By (17.2) (without the factor $(1+\epsilon/2)$) the expected cost satisfy

$$\begin{aligned}\mathbf{E}\left[\sum_{t\geq 1} c(C^t)\right] &\leq \frac{1}{M}\sum_{e\in S^*}\mathbf{E}[D(e)]c(e)\\ &\leq \frac{1}{M}\sum_{e\in S^*}\mathbf{E}[D(e)]\\ &\leq \frac{1}{M}(1.5\cdot M|S^*|/2 + M|S^*|/2) = 1.25\cdot OPT.\end{aligned}$$

□

We note that Berman, Karpinski, and Zelikovsky [BKZ09] give a combinatorial 1.25-approximation algorithm for graphs derived from a metric with distances $\{0,1,2\}$.[8] Previous results for complete graphs with weights 1 and 2 are a 4/3-approximation analysis by Bern and Plassmann [BP89] of the Rayward-Smith heuristic [Ray83], and an algorithm with approximation factor $1.279+\epsilon$ by Robins and Zelikovsky [RZ05].

8 The domain of the metric serves as the set of vertices. An edge (with unit weight) exists between a pair of elements that are at distance 1 from each other.

18 INTEGRALITY GAP LOWER BOUNDS

[T]he inside of a computer is as dumb as hell but it goes like mad!

— Richard P. Feynman, Feynman Lectures on Computation (1996)

As mentioned in the previous chapter, the integrality gap of *BCR* is at most two. This is conjectured to be tight [KPT11], but the currently best known lower bound is $36/31 \approx 1.161$ [Byr+13]. Instances that are hard with respect to *BCR*'s integrality gap are difficult to find. Some of the best known are instances derived from the set cover problem. In the following sections, we will review two techniques of constructing hard instances from the literature that can be described by 'keep distance and average' and 'iterated set cover construction'. We will add another that we call 'entanglement' of two set cover instances.

18.1 Goemans's instance family

Goemans (reported in [Sin00; AC04]) constructs a family of instances with weights 1 and 2 and $n+1$ terminals for $n \in \mathbb{N}$. The integrality gap of these instances approaches $8/7 \approx 1.143$ as n goes to infinity. Moreover, the case of $n = 3$ yields a lower bound of $12/11 = 1.\overline{09}$ for planar graphs. We note that the instances do contain Steiner claws for $n \geq 1$.

The idea behind Goemans's approach is to have a minimum distance of four between terminals by giving the edges a weight of 2. After adding some edges of weight 1, these distances shall not decrease. However, the edges of weight 1 can be used in a feasible fractional solution to lower the total cost.

Goemans's construction is as follows (see Figure 18.1a on the next page for $n = 3$). Create $n+1$ terminals $t_0, t_1, \dots, t_n$. The terminal t_0 is called the *root*, and it is convenient to choose it as the root terminal in the *BCR* relaxation. Create n Steiner vertices $s_1, \dots, s_n$. Each s_i has an edge to t_0 and an edge to t_i, both with weight two. We will now create another $2\binom{n}{2}$ Steiner vertices. Between each pair (t_i, t_j) of the n non-root terminals for $i \neq j$, put a Steiner vertex s_{ij} with edges $(t_i, s_{ij}), (s_{ij}, t_j)$ of weight two. Connect each such s_{ij} to a Steiner vertex s'_{ij} with an edge (s_{ij}, s'_{ij}) of weight one. Finally, connect each s'_{ij} to the Steiner vertices s_i and s_j with an edge of weight one.

An optimal feasible integral solution has cost $OPT = 4n$: Clearly, there is a feasible solution with objective $4n$, for example the union of the paths (t_0, s_i, t_i) for $i = 1, \dots, n$. (There are several other feasible solutions with this objective, see Figure 18.1a.) The minimum distance

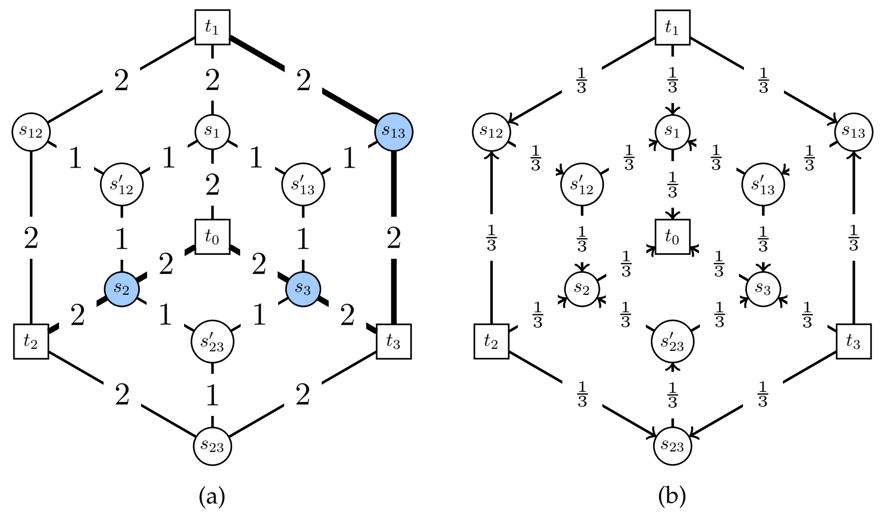


Figure 18.1: (a) Goemans's construction for $n = 3$. The graph has $n + 1$ terminals (squares). The numbers indicate the edge weights. Connecting the terminals has cost of at least twelve. One optimal solution is shown with blue vertices and bold edges. (b) Every directed edge in the fractional *BCR* solution is set to $1/3$ in one direction (from terminals towards the root), and zero in the other (not shown). The total cost is eleven, hence the gap is at least $12/11$.

between any two terminals is four. While the vertices s'_{ij} can serve as hubs that enable us to use less edges of weight 2, they do not allow for an improvement of the cost.

An optimal fractional *BCR* solution is as follows. (The case $n = 3$ can be seen in Figure 18.1b.) Every non-root terminal sends $1/n$ to its Steiner neighbors. Every Steiner vertex s'_{ij} sends $1/n$ to s_{ij}, and every Steiner vertex s_{ij} sends $1/n$ to each s_i and s_j. The Steiner vertices s_i send $1/n$ to the root terminal. The total cost are $OPT_f = 7\frac{n}{2} + \frac{1}{2}$. Hence, the ratio approaches

$$\frac{OPT}{OPT_f} = \frac{4n}{7\frac{n}{2} + \frac{1}{2}} \overset{n\to\infty}{\longrightarrow} \frac{8}{7} \approx 1.143.$$

18.2 INSTANCES BASED ON SET COVER

Another good way of constructing instances with a large gap is to use the reduction from (unweighted) set cover (see Theorem 16.3.4), which is so straightforward that we can directly think of a set cover instance as a Steiner tree instance. Recall that the transformed instance is always quasi-bipartite and has uniform weights. Let n denote the number of elements to be covered and m the number of sets. In this and the following section, we will only consider *uniform* instances where all sets have the same cardinality s, and the number of times

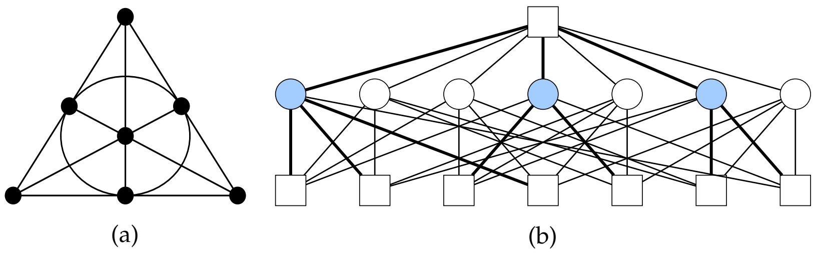


Figure 18.2: (a) The Fano plane, the finite projective plane of order two. It has seven points and seven lines (the circle being a line). (a) By considering the set of points not on a line as a set, an unweighted set cover instance is obtained with seven sets of each four elements. The corresponding Steiner tree instance with uniform weights is shown. An optimal integral feasible solution via a selection of three sets is indicated by blue vertices and bold edges.

an element is present in a set is the same for all elements, namely ms/n. We note that for a choice of parameters n, m, s, there may be several nonisomorphic set cover instances, potentially having different integrality gaps.

Some of these set cover instances, but not all, correspond to line configurations in the plane or block designs (in particular, Steiner systems), or complements thereof, which have interesting connections to other areas of combinatorics. A discussion of these topics would carry us too far afield, the reader may consult [CD06].

Skutella (reported in [KPT11]) constructs a single instance with a gap of exactly $8/7$ from the Fano plane, the finite projective plane of order two. The Fano plane is depicted in Figure 18.2a. It often serves as a counterexample in combinatorics. For example, the corresponding matroid appears frequently as a forbidden minor in matroid theory (see, e.g., [Oxl06]).

Every line in the Fano plane goes through three points, and every two lines intersect in exactly one point. The ground set is the set of points $\{1, 2, \ldots, 7\}$. As sets, Skutella chooses the *complements* of the seven lines, i.e., the set of four points not on a line. The resulting set cover instance is transformed into a Steiner tree instance that can be seen in Figure 18.2b. In order to cover the ground set, three sets have to be selected. Hence $OPT = 7 + 3 = 10$. There is an optimal feasible solution to the BCR relaxation where all directed edges pointing towards the root are selected with $1/4$, and the remaining with zero, and thus $OPT_f = 7 + 7/4$. Hence, the integrality gap of this instance is $8/7$. If one chooses the points on the lines as sets instead ($s = 3$), one obtains an instance with a worse gap (see Table 18.1 on page 211).

An analogous construction can be used for the finite projective plane of order three. By considering the points that are not on a line as a set, we obtain an instance with $n = 13$, $m = 13$, $s = 9$ and a gap of $72/65 \approx 1.108$.

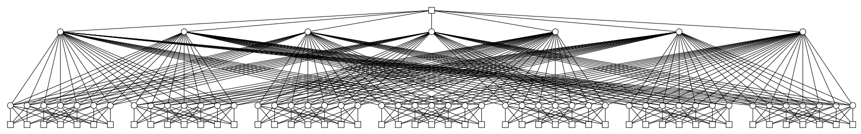


Figure 18.3: The iterated construction of Skutella's instance for $h = 2$. Three Steiner vertices (sets) have to be selected for each of the seven copies of the original instance on the bottom, and in order to select them, three Steiner vertices adjacent to the root have to be selected.

Table 18.1 includes all instances with a gap of at least 1.1 for parameters $1 \leq n \leq 7$, $m \leq 2n$, as well as $n \in \{8, 9\}$, $m \leq n$. They were found in an exhaustive search. In addition, it includes two instances with $m > n$ and six instances with $n \geq 10$ whose gap is at least 1.1. Most of these were constructed manually from combinatorial designs, some of which can be found at the La Jolla covering repository [Gor19]. Several instances whose gap is smaller than 1.1 are also included because we will use them in the following section to construct some interesting instances with $n \geq 10$. Note that the instance with parameters $n = 3$, $m = 3$, $s = 2$ is planar, and to the best of our knowledge, its gap of $1.\overline{1}$ is the best known lower bound for planar graphs.

Note that the combination of two set cover instances with the same parameters n, m, s results in a set cover instance with parameters $2n$, $2m$, s that has twice the optimum integral and optimum fractional objective value, and hence the same ratio. However, we chose to include the parameters $2n, 2m, s$ in Table 18.1 if there are instances with these parameters that are nonisomorphic to such combined instances. Instances with $n = 6$, $m = 6$, $s = 2$ and a ratio of $1.\overline{1}$ are always the combination of two $n = 3$, $m = 3$, $s = 2$ instances, hence they are not included.

Byrka et al. [Byr+13] give an iterated construction for set cover instances, which is as follows. The instances we shall construct have a height h. The case $h = 1$ is exactly the set cover instance. To create an instance of height $h \geq 2$, start from the set cover instance. Let t_i be a non-root terminal of this instance. Turn each t_i into a Steiner vertex, and add $m - 1$ copies of this vertex, i.e., these vertices are connected to the same original Steiner vertices as t_i. Create n instances I_i of height $h - 1$, and remove the root terminal including its incident edges in each of them. The m Steiner vertices on the top level of each I_i, i.e., the Steiner vertices that had been adjacent to the root terminal, are now identified with the m-fold copy of t_i.

Let k be the minimum number of sets that cover all elements in the set cover instance. Note that we have to select k Steiner vertices for each 'packet' of n terminals on the bottom level, and also k for

Table 18.1: *BCR* gaps of several instances based on (unweighted) set cover with n elements and m sets of uniform size s. The smallest number of sets that suffices to cover an instance equals $OPT - n$. The ratio OPT/OPT_f is rounded down on the third digit unless periodic.

n	m	s	OPT	OPT_f	OPT/OPT_f
3	3	3	5	4.5	$1.\overline{1}$
4	4	3	6	$5.\overline{3}$	1.125
5	5	2	8	7.5	$1.0\overline{6}$
5	5	3	7	$6.\overline{6}$	1.05
5	5	4	7	6.25	1.12
6	4	3	9	8	1.125
6	6	3	9	8	1.125
6	6	5	8	7.2	$1.\overline{1}$
6	9	4	8	7.5	$1.0\overline{6}$
6	10	3	9	8	1.125
7	7	3	10	$9.\overline{3}$	1.071
7	7	4	10	8.75	**1.142**
7	7	6	9	$8.1\overline{6}$	1.102
8	4	4	11	10	1.1
8	6	4	11	10	1.1
8	8	3	12	$10.\overline{6}$	1.125
8	8	4	11	10	1.1
9	6	3	13	12	$1.08\overline{3}$
9	6	6	11	10.5	1.047
9	9	5	12	10.8	$1.\overline{1}$
9	9	6	11	10.5	1.047
10	6	5	13	12	$1.08\overline{3}$
10	10	4	14	12.5	1.12
11	11	5	15	13.2	$1.1\overline{36}$
12	12	8	15	13.5	$1.\overline{1}$
13	13	6	17	$15.1\overline{6}$	1.120
13	13	9	16	$14.\overline{4}$	1.107

each packet of m Steiner vertices on the levels above. The value of the optimum integral solution is

$$OPT = n^h + \sum_{i=0}^{h-1} kn^i = n^h + k\frac{n^h - 1}{n-1}.$$

Now consider fractional solutions. We send $\frac{n}{ms}$ from every non-root terminal to its adjacent Steiner vertex. The Steiner vertices adjacent to the root terminal also each send $\frac{n}{ms}$ to the root. All other Steiner vertices send $\frac{n^2}{m^2s^2}$. The (optimal) feasible fractional solution has value

$$OPT_f = n^{h-1}\frac{n}{ms}ms + \frac{n}{ms}m + \sum_{i=0}^{h-2} n^i m^2 s \frac{n^2}{m^2 s^2} = n^h + \frac{n\left(n^h - 1\right)}{(n-1)s}.$$

When Skutella's instance is iterated (Figure 18.3 on page 210), the optimum integral solution has value $\frac{3}{2}7^h - \frac{1}{2}$. The optimum fractional solution has value $\frac{31}{24}7^h - \frac{7}{24}$. The gap thus approaches $36/31 \approx 1.161$ as $h \to \infty$ [Byr+13]. This is the currently best known gap lower bound for *BCR*. We note that the iterated construction generates Steiner claws for $h \geq 2$ in all non-trivial set cover instances, hence 36/31 is not necessarily a lower bound for *HYP*. Skutella's instance provides us with a *HYP* lower bound of 8/7 because it is quasi-bipartite.[1]

Using different instances with the same gap may yield different outcomes in the iterated construction. For example, the instance

$$\{1,2,3\}, \{1,2,4\}, \{1,3,4\}, \{2,3,4\}$$

with parameters $n = 4$, $m = 4$, $s = 3$ has integrality gap 9/8, just as the instance

$$\{1,2,3\}, \{1,4,5\}, \{2,4,6\}, \{3,5,6\}$$

with parameters $n = 6$, $m = 4$, $s = 3$. The gaps when iterating once ($h = 2$) are $39/34 \approx 1.147$ for the former and $57/50 = 1.14$ for the latter, i.e., the resulting gaps may be different. Hence, even if we found only one new instance with gap 8/7, this might result in a better gap than Skutella's instance when using the iterated construction!

The construction can be generalized by using different set cover instances per level, or even different instances on a single level. This, however, does not seem to be of advantage.

18.3 ENTANGLEMENT

In the following, we denote the integrality gap of an instance by g. Consider the sets

$$(\{1,2,3\}, \{1,2,5\}, \{1,4,5\}, \{2,3,4\}, \{3,4,5\})$$

1 Feldmann et al. [Fel+16] show, using an iterated construction of the instance $n = 3, m = 3, s = 2$ with an additional vertex, that the worst-case ratio of the integrality gaps of *HYP* and *BCR* is at least 8/7. The authors leave it as an open question if Skutella's instance yields a better lower bound.

for set $[5] = \{1, 2, 3, 4, 5\}$ in this arbitrary, but fixed order (Figure 18.4a on the following page). Two sets are necessary to cover, so we have

$$OPT = 2 + 5 = 7, \qquad OPT_f = 5 + 5/3, \qquad g = 1.05.$$

One observes the following: While two sets suffice to cover [5], e.g., $\{1, 2, 3\}$ and $\{3, 4, 5\}$, not every pair of sets achieves this, for example $\{1, 2, 3\}$ and $\{1, 2, 5\}$. There are $\binom{5}{2} = 10$ ways of selecting two sets. We call a pair of sets (and their corresponding Steiner vertices) *covering* if their union covers set [5]. One observes that five of the possible pairs cover [5] and five do not. We call the number of covering pairs (or more generally, covering k-tuples) the *covering number* of the instance. We now demonstrate how to use this to our advantage: Add another five terminals to the corresponding Steiner tree instance. If we can connect the five Steiner vertices to these new terminals such that

1. the Steiner vertices and the new terminals correspond to a set cover instance with $n = 5$, $m = 5$, $s = 3$,
2. every covering pair of Steiner vertices for the original five terminals of [5] is a non-covering pair for the new terminals,
3. every covering pair of Steiner vertices for the new five terminals is a non-covering pair for the original five terminals of [5],

then we force selecting three Steiner vertices (instead of two) in the integral solution.

We performed an exhaustive search with a computer program that considered all $\binom{5}{3} = 10$ possible sets of three elements, $\binom{10}{5} = 252$ ways of selecting five of these sets, and $5!$ ways of ordering each of them (i.e., set the mapping to the Steiner vertices). The program found ways of satisfying 1.-3., for example

$$(\{1, 2, 5\}, \{2, 3, 4\}, \{1, 3, 5\}, \{1, 3, 4\}, \{2, 4, 5\}).$$

The solution was verified manually to rule out an error in the program. We say that the resulting Steiner tree instance is *entangled*. It is depicted in Figure 18.4b on the next page. Let us look at the feasible integral and fractional solutions. We have to select three sets, but now there are ten terminals. We install $1/3$ on every edge in the *BCR* solution. We have

$$OPT = 3 + 10 = 13, \quad OPT_f = 10 + 5/3, \quad g = 39/35 \approx 1.114,$$

which is much better than in the original instance. An ILP solver was used to verify the integrality gap.

It is important to note that an instance constructed in this way is a set cover instance on twice the number of elements, and could have been found by other means such as brute force. However, by the above

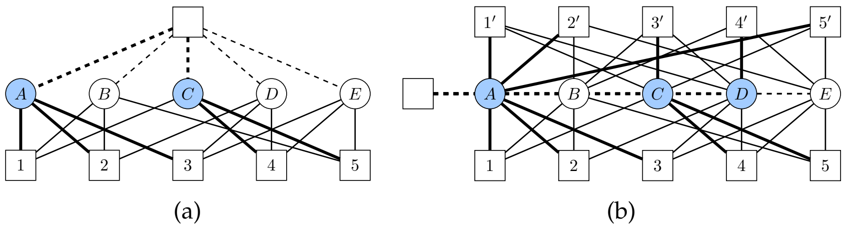


Figure 18.4: (a) An instance of the Steiner tree problem in graphs, derived from a set cover instance. The terminals 1-5 are to be covered by sets corresponding to the Steiner vertices A-E. Selecting two sets via the dashed edges from the root terminal suffices, e.g., A and C. Not every pair of sets is covering, e.g., A and B. (b) Five additional terminals $1'$-$5'$ are added and connected to the Steiner vertices A-E. The five edges from the unlabeled root terminal to the Steiner vertices are only hinted at by dashes in order to avoid obscuring the other edges. It is impossible to cover all terminals by selecting just two Steiner vertices. An optimal solution with three is shown.

discussion we now know *why* this instance is difficult with respect to *BCR*.

We wondered whether the same trick works for other instances as well, and found that this is sometimes the case. Note that the number C_1 of covering k-tuples in an instance need not be equal to the number $\binom{m}{k} - C_2$ of noncovering k-tuples in the other instance, $C_1 \leq \binom{m}{k} - C_2$ suffices to make a pair of instances candidates.

In the following, we will not report 'uninteresting' instances: If for some choice of parameters n, s, there is an instance where k sets must be selected, then we do not check whether entanglement is possible for instances with these parameters where less than k sets suffice, as these will have a worse gap.

Table 18.2 on page 216 shows all interesting instances that are candidates, i.e., their covering numbers make them candidates for entanglement, for parameters $n \leq 7$, $m \leq 2n$ and $8 \leq n \leq 9$, $m \leq n$. We also include two instances with $n = 10$ and an instance with $n = 6$, $m = 14$. The computational power required to perform experiments beyond $n = 7$ is enormous.

As indicated in Table 18.2, we found several instances where entanglement is possible. We only list a few that exhibit a rather large gap; we arbitrarily choose one if several nonisomorphic instances exist:

$$(\{1,2\},\{2,3\},\{3,4\},\{4,5\},\{1,5\});$$
$$(\{1,2\},\{4,5\},\{1,3\},\{2,4\},\{3,5\}),$$

for $n = 5$, $m = 5$, $s = 2$ for covering triplets, and

$$\begin{aligned}&(\{1,2,4,5\},\{1,2,4,6\},\{1,2,5,6\},\{1,3,4,5\},\{1,3,4,6\},\\&\ \{1,3,5,6\},\{2,3,4,5\},\{2,3,4,6\},\{2,3,5,6\});\\&(\{1,2,4,5\},\{1,2,4,6\},\{1,2,5,6\},\{1,3,4,5\},\{1,3,4,6\},\\&\ \{1,3,5,6\},\{2,3,4,5\},\{2,3,4,6\},\{2,3,5,6\}),\end{aligned}$$

for $n = 6$, $m = 9$, $s = 4$ for covering pairs, and

$$\begin{aligned}&(\{1,2,4,6,8,9\},\{1,2,5,6,7,9\},\{1,3,4,7,8,9\},\\&\ \{1,3,5,6,7,8\},\{2,3,4,5,6,9\},\{2,3,4,5,7,8\});\\&(\{1,2,5,6,7,9\},\{1,3,4,7,8,9\},\{1,3,4,6,8,9\},\\&\ \{2,3,4,5,6,7\},\{2,3,4,5,6,8\},\{1,2,5,7,8,9\}),\end{aligned}$$

for $n = 9$, $m = 6$, $s = 6$ for covering pairs, and

$$\begin{aligned}&(\{1,3,5,6\},\{1,3,5,7\},\{1,4,6,8\},\{1,4,7,8\},\\&\ \{2,3,6,8\},\{2,3,7,8\},\{2,4,5,6\},\{2,4,5,7\});\\&(\{1,3,5,6\},\{1,4,7,8\},\{1,4,6,8\},\{1,3,5,7\},\\&\ \{2,3,7,8\},\{2,4,5,6\},\{2,4,5,7\},\{2,3,6,8\}),\end{aligned}$$

for $n = 8$, $m = 8$, $s = 4$ for covering triplets, and

$$\begin{aligned}&(\{1,6,9\},\{1,7,8\},\{2,4,8\},\\&\ \{2,5,7\},\{3,4,9\},\{3,5,6\});\\&(\{1,6,9\},\{1,7,8\},\{3,4,9\},\\&\ \{3,5,6\},\{2,5,7\},\{2,4,8\}),\end{aligned}$$

for $n = 9$, $m = 6$, $s = 3$ for covering quadruplets.

Unfortunately, the instances on six elements that would yield a hypothetical gap of $8/7 \approx 1.143$ (the same as Skutella's instance!) do not admit entanglement.

The effect of forcing the selection of one additional set is counterbalanced by the increased number of terminals. It is thus evident that our technique will not lead to improvements for instances with even moderately large n. For example, there is an instance with $n = 8$, $m = 8$, $k = 3$ that has a gap of $9/8$. The hypothetical entangled instance on 16 elements (see Table 18.2) would also have a gap of $9/8$. Here, entanglement is not possible. We note that there are instances for these parameters that are non-isomorphic to the instance derived from the lines of the Kantor–Möbius configuration, which has 30 covering quadruplets out of 70.

In Skutella's instance ($n = 7$, $m = 7$, $s = 4$), 28 of the $\binom{35}{3} = 35$ triplets of sets are covering, and 7 are non-covering. There are instances with $n = 7$, $m = 7$, $s = 3$ that have 28 non-covering triplets and 7

Table 18.2: Uniform set cover instances that are candidates for entanglement. There can be several possible combinations of covering numbers that need checking, see Table 18.3. The number k of sets needed to cover n elements is $OPT - n$. The hypothetical gap (rounded down on the third digit unless periodic) is the gap of the instance with $2n$ elements constructed by entanglement, if this is possible. An asterisk indicates that not all experiments were fully performed due to time constraints.

k	n	m	s	Gap	Hypoth. gap	Possible (combinations)
2	4	4	2	1	1.1	yes (all)
2	4	6	2	1	1.1	yes (all)
2	5	5	3	1.05	1.114	yes (all)
2	6	8	3	1	1.071	yes (all)
2	6	9	4	$1.0\overline{6}$	$1.\overline{1}$	yes
2	6	12	3	1	1.071	yes (all)
2	6	12	4	$1.0\overline{6}$	$1.\overline{1}$	no
2	7	14	4	1.029	$1.\overline{1}$	yes (all)
2	8	8	5	1.041	1.079	yes (some)
2	9	6	6	1.047	1.076	yes (some)
2	9	9	6	1.047	1.076	yes (some*)
3	5	5	2	$1.0\overline{6}$	1.12	yes (all)
3	5	10	2	$1.0\overline{6}$	1.12	yes (all)
3	6	9	2	1	$1.0\overline{6}$	yes (all)
3	6	10	3	1.125	**1.142**	no
3	6	12	2	1	$1.0\overline{6}$	yes (all)
3	6	12	3	1.125	**1.142**	no
3	6	14	3	1.125	**1.142**	no
3	7	7	3	1.071	1.102	yes (some)
3	7	14	3	1.071	1.102	yes (all)
3	8	8	4	1.1	$1.\overline{1}$	yes (one)
3	9	9	4	$1.0\overline{6}$	1.086	yes (some)
3	10	5	4	1.04	$1.0\overline{6}$	yes (all)
3	10	6	5	$1.08\overline{3}$	$1.\overline{09}$	yes (all)
4	8	8	3	1.125	1.125	no
4	9	6	3	$1.08\overline{3}$	1.125	yes (some)

Table 18.3: Covering numbers for the instances of Table 18.2 that make them candidates for entanglement.

k	n	m	s	Relevant cov. numbers	$\binom{m}{k}$
2	4	4	2	2	6
2	4	6	2	3	15
2	5	5	3	5	10
2	6	8	3	1, 2, 4	28
2	6	9	4	18	36
2	6	12	3	3, 4, 6	66
2	6	12	4	30	66
2	7	14	4	7, 9, . . . , 18	91
2	8	8	5	4, . . . , 12, 16	28
2	9	6	6	6, 7, 8, 9	15
2	9	9	6	9, . . . , 18, 20	36
3	5	5	2	5	10
3	5	10	2	30	120
3	6	9	2	4, 6	84
3	6	10	3	60	120
3	6	12	2	8	220
3	6	12	3	106, 107, 108, 110, 112	220
3	6	14	3	166, 168, 171, 172	364
3	7	7	3	7, 9, . . . , 12, 14, 15	35
3	7	14	3	56, 60, . . . , 82, 84	364
3	8	8	4	24, . . . , 32	56
3	9	9	4	10, . . . , 33	84
3	10	5	4	4	10
3	10	6	5	10	20
4	8	8	3	30, . . . , 39	70
4	9	6	3	6, 7, 8, 9	15

covering triplets.[2] Our program also did not find an entanglement in this asymmetric case. Even if this were possible, we would not improve over $8/7 \approx 1.143$: One would have to select $1/3$ on the edges from the Steiner vertices to the root in the relaxation (instead of $1/4$ for Skutella's instance) because the second instance has $s = 3$. Hence, the gap would be $(14 + 4)/(14 + 7/3) = 54/49 \approx 1.102$.

Likewise, there is an instance with $n = 7$, $m = 7$, $s = 4$ that has 7 covering and 14 noncovering pairs, and an instance with $n = 7$, $m = 7$, $s = 5$ that has 14 covering and 7 noncovering pairs. As there already is an instance for $n = 7$, $m = 7$, $s = 4$ where selecting three sets is necessary, the entangled instance would have a smaller gap. We did not check whether entanglement is possible here.

There are also instances for $n = 9$, $m = 9$, $s = 4$ with covering triplet numbers $\{10, \dots, 33\}$ and for $n = 9$, $m = 9$, $s = 5$ with covering triplet numbers $\{48, \dots, 60\}$. The total number of triplets is 84, so candidates for entanglement would be the pairs of covering numbers

$$\begin{array}{l}(10,48), \dots, (10,60),\\ \quad\vdots\\ (24,48), \dots, (24,60),\\ (25,48), \dots, (25,59),\\ \quad\vdots\\ (33,48), \dots, (33,51).\end{array}$$

We did not check whether this is possible (which would be time-demanding) because one would have to select $1/4$ on the edges to the root in the relaxation. The gap would be $(18 + 4)/(18 + 9/4) = 88/81 \approx 1.086$, which is worse than the gap of the instance with parameters $n = 9$, $m = 9$, $s = 5$.

Judging from the findings of this section, and perhaps the notoriety of the Fano plane in combinatorics, it seems plausible that Skutella's instance has the maximum gap among all uniform unweighted set cover instances.[3] For larger instances, the gap will most likely approach one, simply because of the growing numbers in the numerator and the denominator. However, Table 18.2 shows that there exist instances with $n \in \{11, 13\}$ that have quite large gaps. Further experiments should be made for moderate n, although they pose an enormous computational effort.

To sum up, although we were unable to improve the gap lower bound, our technique gives an insight why some set cover instances

2 In fact, here the sets are the points that *do* lie on the lines of the Fano plane. Other possible covering numbers for $n = 7$, $m = 7$, $s = 3$ are 9, 10, 11, 12, 14, 15. However, for $n = 7$,$m = 7$,$s = 4$, the covering triplet numbers are $\{28, 29, 30\}$. Of these, there are no covering pairs only for 28.

3 We note that no instance exists with parameters $n = 8$, $m = 8$, $s = 5$ where three sets must be selected, which one may have deemed possible. The hypothetical gap is $1.1458\overline{3}$, just slightly exceeding $8/7$.

have a large gap: they consist of two entangled smaller set cover instances.

18.3.1 *Implementation*

A few remarks on our implementation are in order. Since there may be several nonisomorphic instances with the same parameters, we had to rely on brute-force enumeration even when some instance can be created from known combinatorial designs.

It is easily possible to generate all possible m-selections of s-sets with dynamic programming. However, storing them requires a lot of memory, hence we generated the selections one-by-one and added them to a pool of selections only when appropriate. Traversing all possible selections was done using an implicit bitstring representation due to Payne and Ives [PI79]. For $n = 9$, the runtime of our algorithm is still significant. In this case, we fixed without loss of generality two sets S_1, S_2 for different cardinalities $|S_1 \cap S_2|$ in order to reduce the number of combinations.

The generated instances were checked for uniformity; non-uniform instances were discarded.[4] Then, the instances are grouped by their numbers of covering pairs, triplets or quadruplets, depending on the optimum number of sets needed to cover the instance. Only instance pairs whose covering numbers C_1, C_2 fulfill the necessary condition $C_1 \leq \binom{m}{k} - C_2$ are of interest. For these covering numbers, the instances in each group can be distinguished into isomorphism classes, only one instance per class is used then. As isomorphism is an equivalence relation, computing its equivalence classes is often, while costly, computationally feasible. However, if entanglement is possible and the majority of instances allow it, it may be preferable not to compute isomorphism classes because we will likely stumble upon two instances that can be entangled. Hence, one should test for entanglement both with and without isomorphism reduction. We note that the number of isomorphism classes is typically very small, usually below ten.

Note that if two distinct covering numbers align, their number of instances may differ greatly. For example, for parameters $n = 9$, $m = 9$, $s = 6$, there are 1,655,640 instances with covering number 16 (for pairs), but only 1,512 instances with covering number 20.

An isomorphism reduction on 1,512 instances is easily possible and reduces the time of the pairwise check with the 1,655,640 other instances: Here, there is only one isomorphism class, hence the speedup is 1,512. However, isomorphism reduction on 1,655,640 instances is

4 Suppose we consider covering triplets. It may seem that we can discard instances that have covering pairs right away. However, this is dangerous, for it could be the case that the covering pair in one instance is not just a noncovering pair in the other, but also that it cannot be extended to a covering triplet in the other.

very costly and not worth the extra computation if the number of instances to check each of them against is small, as is the case for one isomorphism class. On the other hand, the class for covering number 16 is also checked against other classes, so it is difficult to estimate how to use isomorphism reduction for the best speedup.

19 FURTHER VARIANTS OF THE STEINER TREE PROBLEM

The wheel is come full circle[.]

— William Shakespeare, King Lear, Act 5, Scene 3 (1623)

19.1 THE PRIZE-COLLECTING STEINER TREE PROBLEM

In the Steiner tree problem, the terminal vertices must be spanned by the tree. In a well-known variant, a vertex has a *profit* (or *prize*), which contributes to the objective value if a vertex is selected.

Definition 19.1.1 (Prize-Collecting Steiner Tree (PCST))**.** Given a simple graph $G(V, E)$ with nonnegative vertex profits and edge weights

$$p : V \longrightarrow \mathbb{R}_0^+$$
$$w : E \longrightarrow \mathbb{R}_0^+,$$

find a tree $T = (V_T, E_T)$ in G that maximizes

$$w(T) = \sum_{v \in V_T} p(v) - \sum_{e \in E_T} w(e).$$

We will not discuss the problem by itself, but only use it for proving NP-completeness in the following section. That PCST is NP-complete follows from a simple reduction from the Steiner tree problem by choosing the vertex profits sufficiently large, modelling the terminals. We list some interesting results.

PCST is a Lagrangean relaxation of the k-cardinality tree problem [CRW04]. Bateni et al. [Bat+11] show that PCST can be solved in polynomial time for graphs of bounded treewidth. Building upon this, they show a PTAS for PCST on planar graphs.

19.2 THE MAXIMUM WEIGHT CONNECTED SUBGRAPH PROBLEM

In contrast to the problems considered in earlier chapters, the following problem only has vertex weights.

Definition 19.2.1. Given a graph (V, E, w) with weights $w : V \to \mathbb{R}$, find a connected (induced) subgraph of G of maximum total weight.

The NP-completeness of MWCS was established by Karp in supplementary material to Ideker et al. [Ide+02]. Dittrich et al. [Dit+08] give reductions from PCST to MWCS and vice versa, which also establishes the latter's NP-completeness.

Theorem 19.2.2 ([Ide+02],[Dit+08])**.** *MWCS is* NP*-complete.*

Proof. A given solution can be verified in polynomial time. For NP-hardness we reduce from PCST. Let (V, E, w, p) denote the PCST instance. We modify the graph as follows. For every edge $(u, v) \in E$, we create an intermediate vertex x and replace (u, v) with (u, x) and (x, v). The vertex x gets weight $-w(u, v)$. The original vertices $v \in V$ retain their weights. It is easy to see that this indeed constitutes a polynomial-time reduction. □

As noted by El-Kebir and Klau [EK14], the reduction from PCST to MWCS is approximation-preserving, and hence it is NP-hard to approximate MWCS within a constant factor, as approximating PCST within a constant factor is NP-hard [FPS01].

19.3 PREPROCESSING RULES FOR MWCS

We can process the connected components of the input graph independently of each other and return the best result, hence we assume connectivity in the following. If there is no vertex of positive weight, then a single vertex is an optimal solution, namely one with maximum weight. Hence, we now assume that there is a vertex of positive weight. As we can test all single-vertex solutions in linear time, we can assume that there is an optimal solution of at least two positive vertices.

We will use a *merge* operation for two adjacent vertices u, v that replaces u and v with a vertex x of weight $w(x) = w(u) + w(v)$, which becomes the endpoint of the edges incident to u and v. If there are edges $(u, y), (v, y)$, only one edge (x, y) is the replacement. The operation generalizes to a set of vertices that induce a connected subgraph. The following rules are described in [AB14], and also partly by El-Kebir and Klau [EK14]. They are applied exhaustively.

1. A vertex u with $w(u) \leq 0$ can be removed if its degree is one.

2. If there is a path of vertices of degree two, all of which have weight zero or less, we can merge these vertices. We may do so because one such a vertex is only selected if there is a positively weighted vertex that the path links to, so the whole path must be selected.

3. If there is an edge $(u, v) \in E$ with $w(u) \geq 0$ and $w(v) \geq 0$, then the two vertices can be merged: If u is in an optimal solution, we can select v without decreasing the objective.

4. If for a pair $(u, v) \in \binom{V}{2}$ of vertices, not necessarily adjacent, we have $N(u) \setminus \{v\} \subseteq N(v)$, and $w(u) \leq 0$ and $w(u) \leq w(v)$, then u and all its incident edges can be removed. This is because if $G[S]$ is connected for some $S \subseteq V$ with $u \in S$, then the graph

> $G[(S \setminus \{u\}) \cup \{v\}]$ is also connected. Thus there is an optimal solution that does not contain u.

In order to implement rule 4. in $\mathcal{O}(|V|^2\Delta(G))$ time, sort all adjacency lists in linear time in total (see the proof of Theorem 4.3.1). Then it is straightforward to check for each pair $(u,v) \in \binom{V}{2}$ of vertices if $N(u) \setminus \{v\} \subseteq N(v)$ in time $\mathcal{O}(\Delta)$ by advancing two pointers on the sorted adjacency lists and comparing entries.

If rule 4. is to be implemented only for adjacent pairs, the runtime can obviously be bounded by $\mathcal{O}(|E|\Delta(G))$. We can improve this using the following lemma.

Lemma 19.3.1 (Essentially [CN85]). *In a simple graph $G = (V,E)$, we have*

$$\sum_{uv \in E} \min(\deg(u), \deg(v)) \leq 2|E|p(G).$$

Proof. Consider partition of G into pseudoforests $P_1, \ldots, P_p$. Each pseudoforest can be 1-oriented (see the proof of Theorem 8.2.4), i.e., each vertex is assigned at most one edge. Let $v_i(e)$ be the unique vertex that $e \in P_i$ points to. We have

$$\begin{aligned}\sum_{uv \in E} \min(\deg(u), \deg(v)) &\leq \sum_{i=1}^{p} \sum_{e \in P_i} \deg(v_i(e)) \\ &\leq \sum_{i=1}^{p} \sum_{v \in V} \deg(v) \\ &= 2|E|p.\end{aligned}$$

□

Chiba and Nishizeki used the lemma (stated in terms of forests and Γ) to list the triangles of a graph in $\mathcal{O}(|E|\,\Gamma)$ time. Modifying their algorithm slightly yields the following theorem.

Theorem 19.3.2 (Re-phrased from [CN85]). *Rule 4. can be implemented for adjacent vertices in $\mathcal{O}(|E|p(G))$ time.*

Proof. Let $(u,v) \in E$. Observe that if $\deg(u) > \deg(v)$, then we have $N(u) \setminus \{v\} \nsubseteq N(v)$.

Sort the vertices by their degree in $\mathcal{O}(|V|)$ time by placing them into $|V|$ buckets for every possible degree in $\{0, \ldots, |V|-1\}$. Concatenating the buckets yields the desired sorted list. Create a Boolean array of length $|V|$, initialized with 'false'. This will be used as the row of the adjacency matrix. Go through the list of vertices in descending order of their degree. Let v be the current vertex. Mark all of v's neighbors in the array. Then, for every neighbor u of v with $\deg(u) \leq \deg(v)$, check whether all of u's neighbors except v are marked in the array. If this is the case, then $N(u) \setminus \{v\} \subseteq N(v)$. Once all neighbors u of v have been tested, unmark them in the array, and proceed with the next vertex in the sorted list.

Table 19.1: Reduction of MWCS instances in the ACTMOD dataset used in the 11th DIMACS Implementation Challenge.

	Original instance		After Preprocessing	
Instance	Vertices	Edges	Vertices	Edges
drosophila001	5,226	93,394	2,514	41,825
drosophila005	5,226	93,394	2,486	40,234
drosophila0075	5,226	93,394	2,421	36,328
HCMV	3,863	29,293	974	4,000
lymphoma	2,034	7,756	1,316	6,503
metabol_expr_mice_1	3,523	4,345	1,331	1,858
metabol_expr_mice_2	3,514	4,332	1,293	1,803
metabol_expr_mice_3	2,853	3,335	846	1,178

The algorithm performs $\mathcal{O}(|E|)$ steps for marking and unmarking in the array. For every pair $(u, v) \in E$, we perform $\min(\deg(u), \deg(v))$ lookups in the array (if the degrees are equal, this actually happens twice). By Lemma 19.3.1, this amounts to $\mathcal{O}(|E|p)$ in total. □

The preprocessing rules 1.-4. were applied exhaustively to the ACTMOD dataset of the 11th DIMACS Implementation Challenge. The results for the dataset are shown in Table 19.1. Using the neighborhood reduction also for non-adjacent vertices yields better results than in [AB14]. The latter, in turn, produced better results than the reduction by El-Kebir and Klau [EK14] that only checks whether neighborhoods are equal.

Recently, new reduction techniques (some being NP-hard to compute) were introduced by Rehfeldt and Koch [RK19] and Rehfeldt et al. [RKM19]. See also [RKM19] for reduction techniques for PCST.

19.4 ALGORITHMS FOR MWCS AND ITS VARIANTS

The (M)ILP formulations of Chapters 14 and 16 can be easily adapted for the MWCS problem. If we have edge weights and are interested in the total weight of induced edges, we can model these with additional variables $\tilde{x}_{uv}$ by adding additional constraints (see, e.g., [Alt+14]):

$$\begin{aligned} \tilde{x}_{uv} &\leq y_u, y_v, && uv \in E, \\ \tilde{x}_{uv} &\geq y_u + y_v - 1, && uv \in E. \end{aligned}$$

Another variant was investigated by Backes et al. [Bac+11]. They consider the problem of finding a connected subgraph of k vertices in a directed graph that maximizes the total vertex weight. Here, connected means there is a root node r in the subgraph such that there is a path from r to every other vertex in the subgraph. If we drop the k-cardinality requirement and consider an undirected graph, we have the MWCS problem.

The special feature of the formulation by Backes et al. is that it uses vertex-variables only: For every vertex, there is a binary variable y_v that determines whether v is selected, and a root variable r_v that determines whether v is the root. Let $\text{in}(v) = \{u \in V \mid (u,v) \in E\}$. The constraints

$$y_v - r_v - \sum_{u \in \text{in}(v)} y_u \leq 0, \qquad v \in V, \tag{19.1}$$

require every non-root vertex to have at least one predecessor. However, directed cycles not containing the root also satisfy (19.1), hence the solution could contain several disconnected cycles. Therefore, the constraints are extended. Let $C \subseteq V$ be the set of vertices in a directed cycle, and let $\text{in}(C) = \bigcup_{v \in C} \text{in}(v) \setminus C$. The full ILP is

$$\sum_{v \in V} y_v = k, \tag{19.2}$$

$$\sum_{v \in V} r_v = 1, \tag{19.3}$$

$$y_v - r_v \geq 0, \qquad v \in V, \tag{19.4}$$

$$\sum_{v \in C} (y_v - r_v) - \sum_{v \in \text{in}(C)} y_v \leq |C| - 1, \qquad C \subseteq V : C \text{ cycle}, \tag{19.5}$$

$$y_v \in \{0,1\}, \qquad v \in V, \tag{19.6}$$

$$r_v \in \{0,1\}, \qquad v \in V. \tag{19.7}$$

Constraint (19.3) states that there is exactly one root vertex, and Constraint (19.4) ensures the root vertex must be selected. Constraint (19.5) requires every vertex set of a cycle that does not contain the root vertex to have at least one ingoing edge. This ensures these vertices can be reached from the root as well. Álvarez-Miranda et al. [ÁLM13] show that the LP relaxation of (19.1)-(19.7) is strictly weaker than the generalized subtour elimination constraints. They also gave formulations based on vertex variables only that are polyhedrally equivalent to *GSEC*.

El-Kebir and Klau [EK14] use such a vertex-based ILP. In addition, they divide the graph into smaller components. We give a rough overview. The first layer is obvious: The connected components are processed separately. The second layer is to compute the biconnected components of the graph. Of course, the solution may involve vertices that are in several adjacent biconnected components. Therefore, in addition to solving instances of MWCS in each biconnected component, instances of a rooted variant of the MWCS problem are solved where the cut vertices are chosen as the root that must be selected. Solutions of two adjacent biconnected components that are rooted in the same cut vertex can be combined.

The third layer is to compute the triconnected components of the biconnected components, and proceed in a similar fashion. We omit the details.

20 CONCLUSION AND OUTLOOK

On two occasions, I have been asked [by members of Parliament], "Pray, Mr. Babbage, if you put into the machine wrong figures, will the right answers come out?" I am not able to rightly apprehend the kind of confusion of ideas that could provoke such a question.

— Charles Babbage, Passages from the Life of a Philosopher (1864)

20.1 PROBLEMS SOLVABLE IN POLYNOMIAL TIME

In this thesis, we saw that the densest subgraph problem and the related problems of smallest maximum indegree orientations (equivalently, pseudoarboricity) and arboricity can be attacked with a multitude of algorithmic techniques such as linear programming, flows, matroid theory, and greedy algorithms. Based on these, it is possible to devise both exact and approximative algorithms.

We were able to improve the runtime of an exact pseudoarboricity algorithm by shrinking the search interval with Kowalik's approximation scheme [Kow06] in Chapter 6. In fact, the pseudoarboricity problem seems to hail from the best of all possible worlds: Not only can the search interval be shrunk, it is in addition possible to use the balanced binary search technique by Gabow and Westermann [Wes88; GW92] on the interval. We are even able to repeatedly shrink the search interval because the approximation scheme also uses a binary search, which leads to a $\log^*$ in the runtime bound under certain conditions. We wonder whether more problems exist that these techniques can be applied to.

While we were not able to improve the runtime for the densest subgraph problem asymptotically, our new runtime bounds for the pseudoarboricity also hold for finding an 'almost-densest subgraph' of density greater $\lceil d^* \rceil - 1$ with Dinitz's algorithm [Din70]. The algorithms have the practical advantage that no complicated data structures are required and no intermediate steps have to be performed, which is the case in the Goldberg–Rao flow algorithm [GR98]. In our experiments in Chapter 12, Dinitz's algorithm showed much better performance than push-relabel algorithms and the solving of LPs with the state-of-the-art solver Gurobi.

The pseudoarboricity has previously been used as a stepping stone for the arboricity by Gabow and Westermann [Wes88; GW92]. We took up this idea in Chapter 10 in order to devise a constructive approximation scheme for the arboricity by using Kowalik's approximation scheme. Converting k pseudoforests into $k+1$ forests is possible in near-linear time for every fixed k using several data structures. In Chapter 9, we showed how to achieve linear time for $k = 3$.

Whether our ideas can be developed further to find a faster exact arboricity algorithm remains to be investigated: After computing an optimal pseudoforest partition and converting it into $p+1$ forests with our conversion based on perfect hashing, how fast can forest F_{p+1} be inserted into the other p forests, provided this is feasible? If this step could be performed in time $\mathcal{O}(|E|^{3/2})$, then we would have an exact (randomized) algorithm that is faster than Gabow's [Gab98]. An exact deterministic algorithm with a runtime of $\tilde{\mathcal{O}}(m\Gamma)$ could also be within reach by computing a partition of $p+1$ pseudoforests first.

Recently, Kopelowitz et al. [KPP16] showed that the triangle listing algorithm of Chiba and Nishizeki with runtime $\tilde{\mathcal{O}}(m\Gamma)$ is essentially time-optimal unless the 3SUM conjecture[1] fails. Whether such a conditional lower bound can be shown for the arboricity problem should be investigated.

Another related question is whether the approximation scheme can be made constructive for the densest subgraph problem as well. It would be preferable to the approximation scheme by Toko Worou and Galtier [TG16] because its dependence on ϵ is inversely linear, while the latter's dependence is inversely quadratic. Furthermore, Kowalik's approximation scheme is simple to implement and has excellent practical performance, which can be seen from our experiments for the exact re-orientation algorithm with Dinitz's algorithm.

In addition to these results, we turned the tables and used the arboricity as a stepping stone for the pseudoarboricity in Chapter 11 when it is asymptotically maximal. To this end, Gabow's arboricity algorithm [Gab98] was used to shrink the search interval to constant size after a fast preprocessing that preserves the maximum density. The preprocessing is of independent practical interest because it can reduce the graph size considerably, which was also demonstrated by our experiments in Chapter 12.

20.2 NP-COMPLETE PROBLEMS

We then investigated problems involving connected subgraphs that are NP-complete. We proposed an ILP for spanning subtrees based on the maximum density in Chapter 14. We showed that this formulation is strictly weaker than the well-known generalized subtour elimination constraints (*GSEC*) [Fis+94; Chi+10]. If an upper bound on the number of vertices to select is specified, it becomes incomparable to the latter. We showed that an MILP formulation of Althaus et al. [Alt+14], which generalizes Cohen's idea of using the smallest maximum fractional

1 Given a set A of n integers, are there three distinct elements $(x, y, z) \in A^3$ such that $x + y = z$? It is conjectured that this requires $\Omega(n^{2-o(1)})$ time.

indegree, implies a subset of the constraints of the ILP. The intersection of the polyhedra of the MILP and *GSEC* is in fact a subset of the polyhedron of the ILP based on the maximum density.

The formulation of Althaus et al. only has a linear number of constraints and can thus be easily integrated into ILPs whenever connectivity is a requirement. In our experiments in Chapter 15, Gurobi produced unexpectedly good results with this MILP on default settings. Most instances could be solved to optimality within one hour, significantly beating the *GSEC* ILP with the cutting-plane method and, less significantly, the combination of the two programs. For a fair comparison, we also ran the tests with Gurobi's built-in strategies deactivated. Here, the combined MILP was slightly better than either of the two alone.

Our formulations are tightest when exactly k vertices are to be selected. While they become less tight with growing k, it could be possible that a $2(1 - 1/k)$-approximation algorithm for the k-cardinality problem exists; the currently best known factor is two [Gar05]. Whether our formulations can be used to this end is an interesting question for further research.

In the related Steiner tree problem, we tried to shed some light on the idea behind the analysis of Byrka et al. [Byr+13] in Chapter 17. We showed that there is a better witness tree distribution than the one of Byrka et al. for every tree height. However, we also gave a heuristic argument why both distributions should be asymptotically optimal when the tree height approaches infinity. We also presented a new witness tree distribution for claw-free instances that improves the expected approximation factor from $\ln(4) \approx 1.386$ to roughly 1.354. Whether this is the optimal choice could not be ascertained. We also showed an expected approximation factor of 1.25 for claw-free instances with uniform weights. It would be interesting to try if the techniques of Goemans et al. [Goe+12] can be applied to also obtain these factors as upper bounds on the integrality gap of the bidirected cut relaxation.

Finding good lower bounds to the integrality gap has proven difficult. Our new idea of entangling two instances derived from the set cover problem (Chapter 18) did work in some cases, but no better lower bound than the one achieved by Skutella's instance [KPT11] could be found. We also noted in Section 16.3 that the inapproximability result by Chlebík and Chlebíková [CC08] holds for claw-free instances. One could try using different gadgets in the reduction that contain Steiner claws in order to find a better lower bound for the general case.

[Aar17] Scott Aaronson. “P $\overset{?}{=}$ NP”. In: *Electronic Colloquium on Computational Complexity (ECCC)* 24 (2017), p. 4. URL: https://eccc.weizmann.ac.il/report/2017/004.

[Ach+16] Tobias Achterberg, Robert E. Bixby, Zonghao Gu, Edward Rothberg, and Dieter Weninger. *Presolve Reductions in Mixed Integer Programming*. Tech. rep. 16-44. Zuse-Institut Berlin, 2016.

[Ack28] Wilhelm Ackermann. “Zum Hilbertschen Aufbau der reellen Zahlen”. In: *Mathematische Annalen* 99.1 (1928), pp. 118–133. DOI: 10.1007/BF01459088.

[AC04] A. Agarwal and M. Charikar. “On the advantage of network coding for improving network throughput”. In: *Information Theory Workshop*. 2004, pp. 247–249. DOI: 10.1109/ITW.2004.1405308.

[AHU74] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

[AAR95] Oswin Aichholzer, Franz Aurenhammer, and Günter Rote. *Optimal Graph Orientation with Storage Applications*. Tech. rep. F003-51. SFB ‘Optimierung und Kontrolle’, TU Graz, Austria, 1995. URL: http://www.ist.tugraz.at/files/publications/geometry/aar-ogosa-95.ps.gz.

[AT94] Martin Aigner and Eberhard Triesch. “Realizability and uniqueness in graphs”. In: *Discrete Mathematics* 136.1 (1994), pp. 3–20. DOI: 10.1016/0012-365X(94)00104-Q.

[AG08] Noga Alon and Shai Gutner. “Linear Time Algorithms for Finding a Dominating Set of Fixed Size in Degenerated Graphs”. In: *Algorithmica* 54.4 (July 2008), p. 544. DOI: 10.1007/s00453-008-9204-0.

[Alt+91] Helmut Alt, Norbert Blum, Kurt Mehlhorn, and Markus Paul. “Computing a maximum cardinality matching in a bipartite graph in time $\mathcal{O}(n^{1.5}\sqrt{m/\log n})$”. In: *Information Processing Letters* 37.4 (1991), pp. 237–240. DOI: 10.1016/0020-0190(91)90195-N.

[AB14] Ernst Althaus and Markus Blumenstock. “Algorithms for the Maximum Weight Connected Subgraph and Prize-collecting Steiner Tree Problems”. In: *11th DIMACS Implementation Challenge on Steiner Tree Problems, Providence, Rhode Island, USA, December 4-5*. 2014. eprint: http:

//dimacs11.zib.de/workshop/AlthausBlumenstock.pdf.

[Alt+14] Ernst Althaus, Markus Blumenstock, Alexej Disterhoft, Andreas Hildebrandt, and Markus Krupp. "Algorithms for the Maximum Weight Connected k-Induced Subgraph Problem". In: *Combinatorial Optimization and Applications – 8th International Conference, COCOA 2014, Wailea, Maui, HI, USA, December 19-21, 2014, Proceedings*. 2014, pp. 268–282. DOI: 10.1007/978-3-319-12691-3_21.

[ÁLM13] Eduardo Álvarez-Miranda, Ivana Ljubić, and Petra Mutzel. "The Maximum Weight Connected Subgraph Problem". In: *Facets of Combinatorial Optimization: Festschrift for Martin Grötschel*. Ed. by Michael Jünger and Gerhard Reinelt. Springer Berlin Heidelberg, 2013, pp. 245–270. DOI: 10.1007/978-3-642-38189-8_11.

[AC09] Reid Andersen and Kumar Chellapilla. "Finding Dense Subgraphs with Size Bounds". In: *Algorithms and Models for the Web-Graph, 6th International Workshop, WAW 2009, Barcelona, Spain, February 12-13, 2009. Proceedings*. 2009, pp. 25–37. DOI: 10.1007/978-3-540-95995-3_3.

[Ang+14] Albert Angel, Nick Koudas, Nikos Sarkas, Divesh Srivastava, Michael Svendsen, and Srikanta Tirthapura. "Dense subgraph maintenance under streaming edge weight updates for real-time story identification". In: *The VLDB Journal* 23.2 (2014), pp. 175–199. DOI: 10.1007/s00778-013-0340-z.

[AMZ97] Srinivasa Rao Arikati, Anil Maheshwari, and Christos D. Zaroliagis. "Efficient computation of implicit representations of sparse graphs". In: *Discrete Applied Mathematics* 78.1-3 (1997), pp. 1–16. DOI: 10.1016/S0166-218X(97)00007-3.

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. 1st ed. New York, NY, USA: Cambridge University Press, 2009.

[AK06] Sanjeev Arora and George Karakostas. "A $2+\epsilon$ approximation algorithm for the k-MST problem". In: *Mathematical Programming* 107.3 (2006), pp. 491–504. DOI: 10.1007/s10107-005-0693-1.

[AR98] Sunil Arya and H. Ramesh. "A 2.5-Factor Approximation Algorithm for the k-MST Problem". In: *Information Processing Letters* 65.3 (1998), pp. 117–118. DOI: 10.1016/S0020-0190(98)00010-6.

[Asa+00] Yuichi Asahiro, Kazuo Iwama, Hisao Tamaki, and Takeshi Tokuyama. “Greedily Finding a Dense Subgraph”. In: *Journal of Algorithms* 34.2 (2000), pp. 203–221. DOI: 10.1006/jagm.1999.1062.

[Asa+07] Yuichi Asahiro, Eiji Miyano, Hirotaka Ono, and Kouhei Zenmyo. “Graph Orientation Algorithms to Minimize the Maximum Outdegree”. In: *International Journal of Foundations of Computer Science* 18.02 (2007), pp. 197–215. DOI: 10.1142/S0129054107004644.

[Awe+95] Baruch Awerbuch, Yossi Azar, Avrim Blum, and Santosh Vempala. “Improved approximation guarantees for minimum-weight k-trees and prize-collecting salesmen”. In: *Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing, 29 May-1 June 1995, Las Vegas, Nevada, USA*. 1995, pp. 277–283. DOI: 10.1145/225058.225139.

[Bac+11] Christina Backes et al. “An integer linear programming approach for finding deregulated subgraphs in regulatory networks”. In: *Nucleic Acids Research* (2011). DOI: 10.1093/nar/gkr1227.

[BKV12] Bahman Bahmani, Ravi Kumar, and Sergei Vassilvitskii. “Densest Subgraph in Streaming and MapReduce”. In: *The Proceedings of the VLDB Endowment* 5.5 (2012), pp. 454–465. DOI: 10.14778/2140436.2140442.

[Băl+15] Oana Denisa Bălălău, Francesco Bonchi, T.-H. Hubert Chan, Francesco Gullo, and Mauro Sozio. “Finding Subgraphs with Maximum Total Density and Limited Overlap”. In: *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining, WSDM 2015, Shanghai, China, February 2-6, 2015*. 2015, pp. 379–388. DOI: 10.1145/2684822.2685298.

[Bal70] Michel L. Balinski. “On a Selection Problem”. In: *Management Science* 17.3 (1970), pp. 230–231. DOI: 10.1287/mnsc.17.3.230.

[Ban80] Lech Banachowski. “A complement to Tarjan’s result about the lower bound on the complexity of the set union problem”. In: *Information Processing Letters* 11.2 (1980), pp. 59–65. DOI: 10.1016/0020-0190(80)90001-0.

[BU17] Nikhil Bansal and Seeun William Umboh. “Tight approximation bounds for dominating set on graphs of bounded arboricity”. In: *Information Processing Letters* 122 (2017), pp. 21–24. DOI: 10.1016/j.ipl.2017.01.011.

[Bap14] Ravindra B. Bapat. *Graphs and Matrices*. 2nd ed. Universitext. Springer, 2014. DOI: 10.1007/978-1-4471-6569-9.

[BA99] Albert-László Barabási and Réka Albert. "Emergence of Scaling in Random Networks". In: *Science* 286.5439 (1999), pp. 509–512. DOI: 10.1126/science.286.5439.509.

[BE10] Leonid Barenboim and Michael Elkin. "Sublogarithmic distributed MIS algorithm for sparse graphs using Nash-Williams decomposition". In: *Distributed Computing* 22.5 (2010), pp. 363–379. DOI: 10.1007/s00446-009-0088-2.

[Bat+11] MohammadHossein Bateni, Chandra Chekuri, Alina Ene, Mohammad T. Hajiaghayi, Nitish Korula, and Dániel Marx. "Prize-collecting Steiner Problems on Planar Graphs". In: *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*. 2011, pp. 1028–1049. DOI: 10.1137/1.9781611973082.79.

[Bea90] John E. Beasley. "OR-Library: Distributing Test Problems by Electronic Mail". In: *The Journal of the Operational Research Society* 41.11 (1990), pp. 1069–1072. DOI: 10.2307/2582903.

[BKZ09] Piotr Berman, Marek Karpinski, and Alexander Zelikovsky. "1.25-Approximation Algorithm for Steiner Tree Problem with Distances 1 and 2". In: *Algorithms and Data Structures*. Ed. by Frank Dehne, Marina Gavrilova, Jörg-Rüdiger Sack, and Csaba D. Tóth. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 86–97.

[BP89] Marshall Bern and Paul Plassmann. "The Steiner problem with edge lengths 1 and 2". In: *Information Processing Letters* 32.4 (1989), pp. 171–176. DOI: 10.1016/0020-0190(89)90039-2.

[BMS81] Alberto Bertoni, Giancarlo Mauri, and Nicoletta Sabadini. "A Characterization of the Class of Functions Computable in Polynomial Time on Random Access Machines". In: *Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computing*. Milwaukee, Wisconsin, USA: ACM, 1981, pp. 168–176. DOI: 10.1145/800076.802470.

[BT97] Dimitris Bertsimas and John Tsitsiklis. *Introduction to Linear Optimization*. 1st ed. Athena Scientific, 1997.

[BC14] Stephan Beyer and Markus Chimani. "Steiner Tree 1.39-Approximation in Practice". In: *Mathematical and Engineering Methods in Computer Science*. Ed. by Petr Hliněný, Zdeněk Dvořák, Jiří Jaroš, Jan Kofroň, Jan Kořenek, Petr Matula, and Karel Pala. Cham: Springer International

Publishing, 2014, pp. 60–72. DOI: 10.1007/978-3-319-14896-0_6.

[Bez00] Ivona Bezáková. "Compact Representations of Graphs and Adjacency Testing". http://people.cs.uchicago.edu/~ivona/PAPERS/GraphRepr.ps. MA thesis. Slovakia: Comenius University, Bratislava, 2000.

[Bha+10] Aditya Bhaskara, Moses Charikar, Eden Chlamtáč, Uriel Feige, and Aravindan Vijayaraghavan. "Detecting High Log-Densities – an $\mathcal{O}(n^{1/4})$ Approximation for Densest k-Subgraph". In: *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*. 2010, pp. 201–210. DOI: 10.1145/1806689.1806719.

[Bha+15a] Sayan Bhattacharya, Monika Henzinger, Danupon Nanongkai, and Charalampos Tsourakakis. "Space- and Time-Efficient Algorithm for Maintaining Dense Subgraphs on One-Pass Dynamic Streams". In: *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*. Portland, Oregon, USA: ACM, 2015, pp. 173–182. DOI: 10.1145/2746539.2746592.

[Bha+15b] Sayan Bhattacharya, Monika Henzinger, Danupon Nanongkai, and Charalampos E. Tsourakakis. "Space- and Time-Efficient Algorithm for Maintaining Dense Subgraphs on One-Pass Dynamic Streams". In: *CoRR* abs/1504.02268 (2015). arXiv: 1504.02268.

[Bie+04] Therese Biedl, Erik D. Demaine, Christian A. Duncan, Rudolf Fleischer, and Stephen G. Kobourov. "Tight bounds on maximal and maximum matchings". In: *Discrete Mathematics* 285.1 (2004), pp. 7–15. DOI: 10.1016/j.disc.2004.05.003.

[BX00] Maria J. Blesa and Fatos Xhafa. "A C++ implementation of tabu search for k-cardinality tree problem based on generic programming and component reuse". In: *Net.ObjectDays 2000 Tagungsband*. Net.ObjectDays-Forum. Erfurt, Germany, 2000, pp. 648–652.

[BRV96] Avrim Blum, R. Ravi, and Santosh Vempala. "A Constant-factor Approximation Algorithm for the k-MST Problem". In: *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996*. Ed. by Gary L. Miller. ACM, 1996, pp. 442–448. DOI: 10.1145/237814.237992.

[BRV99] Avrim Blum, R. Ravi, and Santosh Vempala. "A Constant-Factor Approximation Algorithm for the k-MST Problem". In: *Journal of Computer and System Sciences* 58.1 (1999), pp. 101–108. DOI: 10.1006/jcss.1997.1542.

[BB05] Christian Blum and Maria J. Blesa. "New metaheuristic approaches for the edge-weighted k-cardinality tree problem". In: *Computers & Operations Research* 32.6 (2005), pp. 1355–1377. DOI: 10.1016/j.cor.2003.11.007.

[Blu16] Markus Blumenstock. "Fast Algorithms for Pseudoarboricity". In: *Proceedings of the Eighteenth Workshop on Algorithm Engineering and Experiments, ALENEX 2016, Arlington, Virginia, USA, January 10, 2016*. 2016, pp. 113–126. DOI: 10.1137/1.9781611974317.10.

[BD97] Al Borchers and Ding-Zhu Du. "The k-Steiner Ratio in Graphs". In: *SIAM Journal on Computing* 26.3 (1997), pp. 857–869. DOI: 10.1137/S0097539795281086.

[Bor+17] Glencora Borradaile, Jennifer Iglesias, Theresa Migler, Antonio Ochoa, Gordon Wilfong, and Lisa Zhang. "Egalitarian Graph Orientations". In: *Journal of Graph Algorithms and Applications* 21.4 (2017), pp. 687–708. DOI: 10.7155/jgaa.00435.

[BKK07] Glencora Borradaile, Claire Kenyon-Mathieu, and Philip N. Klein. "A polynomial-time approximation scheme for Steiner tree in planar graphs". In: *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2007, New Orleans, Louisiana, USA, January 7-9, 2007*. 2007, pp. 1285–1294. URL: http://dl.acm.org/citation.cfm?id=1283383.1283521.

[Bor26] Otakar Borůvka. "O jistém problému minimálním [On a certain minimal problem, in Czech]". In: *Práce Moravské přírodovědecká společnost*. Práce Moravské přírodovědecké společnosti 3.3 (1926), pp. 37–58.

[Bra+14] Marcus Brazil, Ronald L. Graham, Doreen A. Thomas, and Martin Zachariasen. "On the history of the Euclidean Steiner tree problem". In: *Archive for History of Exact Sciences* 68.3 (2014), pp. 327–354. DOI: 10.1007/s00407-013-0127-z.

[BF99] Gerth S. Brodal and Rolf Fagerberg. "Dynamic Representation of Sparse Graphs". In: *Proceedings of the 6th International Workshop on Algorithms and Data Structures*. WADS '99. London, UK: Springer-Verlag, 1999, pp. 342–351. URL: http://dl.acm.org/citation.cfm?id=645932.673191.

[BLT12] Gerth S. Brodal, George Lagogiannis, and Robert E. Tarjan. "Strict Fibonacci heaps". In: *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*. 2012, pp. 1177–1184. DOI: 10.1145/2213977.2214082.

[BS04] Thang Nguyen Bui and Gnanasekaran Sundarraj. "Ant System for the k-Cardinality Tree Problem". In: *Genetic and Evolutionary Computation - GECCO 2004, Genetic and Evolutionary Computation Conference, Seattle, WA, USA, June 26-30, 2004, Proceedings, Part I*. Ed. by Kalyanmoy Deb et al. Vol. 3102. Lecture Notes in Computer Science. Springer, 2004, pp. 36–47. DOI: 10.1007/978-3-540-24854-5_4.

[Byr+10] Jarosław Byrka, Fabrizio Grandoni, Thomas Rothvoß, and Laura Sanità. "An Improved LP-based Approximation for Steiner Tree". In: *Proceedings of the Forty-second ACM Symposium on Theory of Computing*. Cambridge, Massachusetts, USA: ACM, 2010, pp. 583–592. DOI: 10.1145/1806689.1806769.

[Byr+13] Jarosław Byrka, Fabrizio Grandoni, Thomas Rothvoß, and Laura Sanità. "Steiner Tree Approximation via Iterative Randomized Rounding". In: *Journal of the ACM* 60.1 (2013), 6:1–6:33. DOI: 10.1145/2432622.2432628.

[Cat+92] Paul A. Catlin, Jerrold W. Grossman, Arthur M. Hobbs, and Hong-Jian Lai. "Fractional Arboricity, Strength, and Principal Partitions in Graphs and Matroids". In: *Discrete Applied Mathematics* 40.3 (1992), pp. 285–302. DOI: 10.1016/0166-218X(92)90002-R.

[CKP10a] Deeparnab Chakrabarty, Jochen Könemann, and David Pritchard. "Hypergraphic LP Relaxations for Steiner Trees". In: *Integer Programming and Combinatorial Optimization: 14th International Conference, IPCO 2010, Lausanne, Switzerland, June 9-11, 2010. Proceedings*. Ed. by Friedrich Eisenbrand and F. Bruce Shepherd. Springer Berlin Heidelberg, 2010, pp. 383–396. DOI: 10.1007/978-3-642-13036-6_29.

[CKP10b] Deeparnab Chakrabarty, Jochen Könemann, and David Pritchard. "Integrality gap of the hypergraphic relaxation of Steiner trees: A short proof of a 1.55 upper bound". In: *Operations Research Letters* 38.6 (2010), pp. 567–570. DOI: 10.1016/j.orl.2010.09.004.

[Cha00a] Moses Charikar. "Greedy Approximation Algorithms for Finding Dense Components in a Graph". In: *Proceedings of the Third International Workshop on Approximation Algorithms for Combinatorial Optimization*. APPROX

'00. London, UK: Springer-Verlag, 2000, pp. 84–95. URL: http://dl.acm.org/citation.cfm?id=646688.702972.

[Chaoob] Bernard Chazelle. "A Minimum Spanning Tree Algorithm with inverse-Ackermann Type Complexity". In: *Journal of the ACM* 47.6 (2000), pp. 1028–1047. DOI: 10.1145/355541.355562.

[Che+94] Boliong Chen, Makoto Matsumoto, Jianfang Wang, Zhongfu Zhang, and Jianxun Zhang. "A short proof of Nash-Williams' theorem for the arboricity of a graph". In: *Graphs and Combinatorics* 10.1 (1994), pp. 27–28.

[Che+17] Min Chen, Seog-Jin Kim, Alexandr V. Kostochka, Douglas B. West, and Xuding Zhu. "Decomposition of sparse graphs into forests: The Nine Dragon Tree Conjecture for $k \leq 2$". In: *Journal of Combinatorial Theory, Series B* 122 (2017), pp. 741–756. DOI: 10.1016/j.jctb.2016.09.004.

[CZZ19] Xujin Chen, Wenan Zang, and Qiulan Zhao. "Densities, Matchings, and Fractional Edge-Colorings". In: *SIAM Journal on Optimization* 29.1 (2019), pp. 240–261. DOI: 10.1137/17M1147676.

[Che95] Y. L. Chen. "A parametric maximum flow algorithm for bipartite graphs with applications". In: *European Journal of Operational Research* 80.1 (1995), pp. 226–235. DOI: 10.1016/0377-2217(93)E0161-P.

[CM88] J. Cheriyan and S. N. Maheshwari. "Analysis of preflow push algorithms for maximum network flow". English. In: *Foundations of Software Technology and Theoretical Computer Science*. Ed. by Kesav V. Nori and Sanjeev Kumar. Vol. 338. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1988, pp. 30–48. DOI: 10.1007/3-540-50517-2_69.

[CHM96] Joseph Cheriyan, Torben Hagerup, and Kurt Mehlhorn. "An $o(n^3)$-Time Maximum-Flow Algorithm". In: *SIAM Journal on Computing* 25.6 (1996), pp. 1144–1170. DOI: 10.1137/S0097539791278376.

[CG97] Boris V. Cherkassky and Andrew V. Goldberg. "On Implementing the Push-Relabel Method for the Maximum Flow Problem". In: *Algorithmica* 19.4 (1997), pp. 390–410. DOI: 10.1007/PL00009180.

[CN85] Norishige Chiba and Takao Nishizeki. "Arboricity and Subgraph Listing Algorithms". In: *SIAM Journal on Computing* 14.1 (1985), pp. 210–223. DOI: 10.1137/0214017.

[Chi+10] Markus Chimani, Maria Kandyba, Ivana Ljubić, and Petra Mutzel. “Obtaining Optimal *k*-Cardinality Trees Fast”. In: *Journal of Experimental Algorithmics* 14 (2010), 5:2.5–5:2.23. DOI: 10.1145/1498698.1537600.

[CKM07] Markus Chimani, Maria Kandyba, and Petra Mutzel. “A New ILP Formulation for 2-Root-Connected Prize-Collecting Steiner Networks”. In: *Algorithms - ESA 2007, 15th Annual European Symposium, Eilat, Israel, October 8-10, 2007, Proceedings*. Ed. by Lars Arge, Michael Hoffmann, and Emo Welzl. Vol. 4698. Lecture Notes in Computer Science. Springer, 2007, pp. 681–692. DOI: 10.1007/978-3-540-75520-3_60.

[CMZ12] Markus Chimani, Petra Mutzel, and Bernd Zey. “Improved Steiner tree algorithms for bounded treewidth”. In: *Journal of Discrete Algorithms* 16 (2012). Selected papers from the 22nd International Workshop on Combinatorial Algorithms (IWOCA 2011), pp. 67–78. DOI: 10.1016/j.jda.2012.04.016.

[CC08] Miroslav Chlebík and Janka Chlebíková. “The Steiner tree problem on graphs: Inapproximability results”. In: *Theoretical Computer Science* 406.3 (2008). Algorithmic Aspects of Global Computing, pp. 207–214. DOI: 10.1016/j.tcs.2008.06.046.

[CR94] Sunil Chopra and Mendu R. Rao. “The Steiner tree problem I: Formulations, compositions and extension of facets”. In: *Mathematical Programming* 64.1 (1994), pp. 209–229. DOI: 10.1007/BF01582573.

[CE91] Marek Chrobak and David Eppstein. “Planar orientations with low out-degree and compaction of adjacency matrices”. In: *Theoretical Computer Science* 86.2 (1991), pp. 243–266. DOI: 10.1016/0304-3975(91)90020-3.

[CRW04] Fabián A. Chudak, Tim Roughgarden, and David P. Williamson. “Approximate *k*-MSTs and *k*-Steiner trees via the primal-dual method and Lagrangean relaxation”. In: *Mathematical Programming* 100.2 (2004), pp. 411–421. DOI: 10.1007/s10107-003-0479-2.

[Cob65] Alan Cobham. “The Intrinsic Computational Difficulty of Functions”. In: *Logic, Methodology and Philosophy of Science: Proceedings of the 1964 International Congress (Studies in Logic and the Foundations of Mathematics)*. Ed. by Yehoshua Bar-Hillel. North-Holland Publishing, 1965, pp. 24–30.

[Coh10] Nathann Cohen. *Several Graph problems and their Linear Program formulations*. French Institute for Research in Computer Science and Automation (INRIA). 2010. URL: https://hal.inria.fr/inria-00504914.

[CD06] Charles J. Colbourn and Jeffrey H. Dinitz. *Handbook of Combinatorial Designs, Second Edition*. Discrete Mathematics and Its Applications. Chapman & Hall/CRC, 2006.

[Coo71] Stephen A. Cook. "The Complexity of Theorem-proving Procedures". In: *Proceedings of the Third Annual ACM Symposium on Theory of Computing*. Shaker Heights, Ohio, USA: ACM, 1971, pp. 151–158. DOI: 10.1145/800157.805047.

[CR73] Stephen A. Cook and Robert A. Reckhow. "Time Bounded Random Access Machines". In: *Journal of Computer and System Sciences* 7.4 (1973), pp. 354–375. DOI: 10.1016/S0022-0000(73)80029-7.

[Cor+01] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. 2nd. McGraw-Hill Higher Education, 2001.

[CGW91] Collette R. Coullard, John G. del Greco, and Donald K. Wagner. "Representations of bicircular matroids". In: *Discrete Applied Mathematics* 32.3 (1991), pp. 223–240. DOI: 10.1016/0166-218X(91)90001-D.

[Dan49] George B. Dantzig. "Programming of Interdependent Activities: II Mathematical Model". In: *Econometrica* 17.3/4 (1949), pp. 200–211. DOI: 10.2307/1905523.

[Dan51] George B. Dantzig. "Maximization of a Linear Function of Variables Subject to Linear Inequalities, in Activity Analysis of Production and Allocation". In: New York: Wiley, 1951. Chap. XXI.

[Dan63] George B. Dantzig. *Linear programming and extensions*. Rand Corporation Research Study. Princeton, NJ: Princeton Univ. Press, 1963. XVI, 625.

[DFJ54] George B. Dantzig, Delbert R. Fulkerson, and Selmer M. Johnson. "Solution of a Large-Scale Traveling-Salesman Problem". In: *Operations Research* 2.4 (1954), pp. 393–410. DOI: 10.1287/opre.2.4.393.

[Das+12] Atish Das Sarma, Ashwin Lall, Danupon Nanongkai, and Amitabh Trehan. "Dense Subgraphs on Dynamic Networks". In: *Distributed Computing*. Ed. by Marcos K. Aguilera. Springer Berlin Heidelberg, 2012, pp. 151–165.

[DHS91] Alice M. Dean, Joan P. Hutchinson, and Edward R. Scheinerman. “On the Thickness and Arboricity of a Graph”. In: *Journal of Combinatorial Theory, Series B* 52.1 (1991), pp. 147–151. DOI: 10.1016/0095-8956(91)90100-X.

[Din70] E. A. Dinic. “Algorithm for Solution of a Problem of Maximum Flow in a Network with Power Estimation”. In: *Soviet Mathematics Doklady* 11 (1970), pp. 1277–1280.

[Din06] Yefim Dinitz. “Dinitz’ Algorithm: The Original Version and Even’s Version”. In: *Theoretical Computer Science: Essays in Memory of Shimon Even*. Ed. by Oded Goldreich, Arnold L. Rosenberg, and Alan L. Selman. Springer Berlin Heidelberg, 2006, pp. 218–240. DOI: 10.1007/11685654_10.

[Din+08] Irina Dinu, John D. Potter, Thomas Mueller, Qi Liu, Adeniyi J. Adewale, Gian S. Jhangri, Gunilla Einecke, Konrad S. Famulski, Philip Halloran, and Yutaka Yasui. “Gene-set analysis and reduction”. In: *Briefings in Bioinformatics* 10.1 (2008), pp. 24–34. DOI: 10.1093/bib/bbn042.

[DS13] Irit Dinur and David Steurer. “Analytical Approach to Parallel Repetition”. In: *CoRR* abs/1305.1979 (2013). arXiv: 1305.1979.

[DS14] Irit Dinur and David Steurer. “Analytical Approach to Parallel Repetition”. In: *Proceedings of the Forty-sixth Annual ACM Symposium on Theory of Computing*. New York, NY: ACM, 2014, pp. 624–633. DOI: 10.1145/2591796.2591884.

[Dit+08] Marcus T. Dittrich, Gunnar W. Klau, Andreas Rosenwald, Thomas Dandekar, and Tobias Müller. “Identifying functional modules in protein-protein interaction networks: an integrated exact approach”. In: *Bioinformatics* 24.13 (2008), pp. i223–i231. DOI: 10.1093/bioinformatics/btn161.

[Don+18] Riccardo Dondi, Mohammad M. Hosseinzadeh, Giancarlo Mauri, and Italo Zoppis. “Top-*k* Overlapping Densest Subgraphs: Approximation and Complexity”. In: *CoRR* abs/1809.02434 (2018). arXiv: 1809.02434.

[DGP09] Yon Dourisboure, Filippo Geraci, and Marco Pellegrini. “Extraction and Classification of Dense Implicit Communities in the Web Graph”. In: *ACM Transactions on the Web* 3.2 (2009), 7:1–7:36. DOI: 10.1145/1513876.1513879.

[DW71] Stuart E. Dreyfus and Robert A. Wagner. "The Steiner problem in graphs". In: *Networks* 1.3 (1971), pp. 195–207. DOI: 10.1002/net.3230010302.

[DP14] Ran Duan and Seth Pettie. "Linear-Time Approximation for Maximum Weight Matching". In: *Journal of the ACM* 61.1 (2014), 1:1–1:23. DOI: 10.1145/2529989.

[DEK04] Christian A. Duncan, David Eppstein, and Stephen G. Kobourov. "The Geometric Thickness of Low Degree Graphs". In: *Proceedings of the Twentieth Annual Symposium on Computational Geometry*. SCG '04. Brooklyn, New York, USA: ACM, 2004, pp. 340–346. DOI: 10.1145/997817.997868.

[Edm65] Jack Edmonds. "Minimum Partition of a Matroid Into Independent Subsets". In: *Journal of Research of the National Bureau of Standards, B* 69 (1965), pp. 67–72.

[Edm67] Jack Edmonds. "Optimum branchings". In: *Journal of Research of the National Bureau of Standards, B* 71 (1967), pp. 233–240.

[EK72] Jack Edmonds and Richard M. Karp. "Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems". In: *Journal of the ACM* 19.2 (1972), pp. 248–264. DOI: 10.1145/321694.321699.

[Ehr+97] Matthias Ehrgott, Jörg Freitag, Horst W. Hamacher, and Francesco Maffioli. "Heuristics for the K-Cardinality Tree and Subgraph Problems". In: *Asia-Pacific Journal of Operational Research* 14.1 (1997), pp. 87–114.

[EK14] Mohammed El-Kebir and Gunnar W. Klau. "Solving the Maximum-Weight Connected Subgraph Problem to Optimality". In: *CoRR* abs/1409.5308 (2014). arXiv: 1409.5308.

[EFS56] Peter Elias, Amiel Feinstein, and Claude E. Shannon. "A note on the maximum flow through a network". In: *IRE Transactions on Information Theory* 2.4 (1956), pp. 117–119. DOI: 10.1109/TIT.1956.1056816.

[ELS15] Alessandro Epasto, Silvio Lattanzi, and Mauro Sozio. "Efficient Densest Subgraph Computation in Evolving Graphs". In: *Proceedings of the 24th International Conference on World Wide Web, WWW 2015, Florence, Italy, May 18-22, 2015*. 2015, pp. 300–310. DOI: 10.1145/2736277.2741638.

[Epp94] David Eppstein. "Arboricity and Bipartite Subgraph Listing Algorithms". In: *Information Processing Letters* 51.4 (1994), pp. 207–211. DOI: 10.1016/0020-0190(94)90121-X.

[ELS13] David Eppstein, Maarten Löffler, and Darren Strash. "Listing All Maximal Cliques in Large Sparse Real-World Graphs". In: *ACM Journal of Experimental Algorithmics* 18 (2013). DOI: 10.1145/2543629.

[Eul36] Leonhard Euler. "Solutio problematis ad geometriam situs pertinentis [The solution of a problem relating to the geometry of position, in Latin]". In: *Commentarii Academiae Scientiarum Imperialis Petropolitanae* 8 (1736), pp. 128–140.

[Eve11] Shimon Even. *Graph Algorithms*. 2nd ed. New York, NY, USA: Cambridge University Press, 2011.

[ET75] Shimon Even and Robert E. Tarjan. "Network Flow and Testing Graph Connectivity". In: *SIAM Journal on Computing* 4.4 (1975), pp. 507–518. DOI: 10.1137/0204043.

[Fan+15] Genghua Fan, Yan Li, Ning Song, and Daqing Yang. "Decomposing a graph into pseudoforests with one having bounded degree". In: *Journal of Combinatorial Theory, Series B* 115 (2015), pp. 72–95. DOI: 10.1016/j.jctb.2015.05.003.

[Far02] Julius Farkas. "Theorie der einfachen Ungleichungen [Theory of simple inequalities, in German]". In: *Journal für die reine und angewandte Mathematik* 124 (1902), pp. 1–27. URL: http://eudml.org/doc/149129.

[FM95] Tomás Feder and Rajeev Motwani. "Clique Partitions, Graph Compression and Speeding-Up Algorithms". In: *Journal of Computer and System Sciences* 51.2 (1995), pp. 261–272. DOI: 10.1006/jcss.1995.1065.

[FPS01] Joan Feigenbaum, Christos H. Papadimitriou, and Scott Shenker. "Sharing the Cost of Multicast Transmissions". In: *Journal of Computer and System Sciences* 63.1 (2001), pp. 21–41. DOI: 10.1006/jcss.2001.1754.

[Fel+16] Andreas Emil Feldmann, Jochen Könemann, Neil Olver, and Laura Sanità. "On the equivalence of the bidirected and hypergraphic relaxations for Steiner tree". In: *Mathematical Programming* 160.1 (2016), pp. 379–406. DOI: 10.1007/s10107-016-0987-5.

[Fis+94] Matteo Fischetti, Horst W. Hamacher, Kurt Jørnsten, and Francesco Maffioli. "Weighted k-Cardinality Trees: Complexity and Polyhedral Structure". In: *Networks* 24.1 (1994), pp. 11–21. DOI: 10.1002/net.3230240103.

[FF56] Lester R. Ford and Delbert R. Fulkerson. "Maximal Flow through a Network". In: *Canadian Journal of Mathematics* 8 (1956), pp. 399–404. URL: http://www.rand.org/pubs/papers/P605/.

[FF57] Lester R. Ford and Delbert R. Fulkerson. "A simple algorithm for finding maximal network flows and an application to the Hitchcock problem". In: *Canadian Journal of Mathematics* (1957), pp. 210–218.

[FF10] Lester R. Ford and Delbert R. Fulkerson. *Flows in Networks*. Princeton, NJ, USA: Princeton University Press, 2010.

[FR83] L. R. Foulds and V. J. Rayward-Smith. "Steiner Problems in Graphs: Algorithms and Applications". In: *Engineering Optimization* 7.1 (1983), pp. 7–16. DOI: 10.1080/03052158308960625.

[FG78] András Frank and András Gyárfás. "How to orient the edges of a graph?" In: *COMBINATORICS: 5th Hungarian Colloquium, Keszthely, June/July 1976, Proceedings*. Ed. by András Hajnal and Vera T. Sós. Colloquia Mathematica Societatis János Bolyai 2. Amsterdam: North Holland Publishing Company, 1978, pp. 353–364.

[FKS84] Michael L. Fredman, János Komlós, and Endre Szemerédi. "Storing a Sparse Table with $\mathcal{O}(1)$ Worst Case Access Time". In: *Journal of the ACM* 31.3 (1984), pp. 538–544. DOI: 10.1145/828.1884.

[FS89] Michael L. Fredman and Michael E. Saks. "The Cell Probe Complexity of Dynamic Data Structures". In: *Proceedings of the Twenty-first Annual ACM Symposium on Theory of Computing*. Seattle, Washington, USA: ACM, 1989, pp. 345–354. DOI: 10.1145/73007.73040.

[FT87] Michael L. Fredman and Robert E. Tarjan. "Fibonacci Heaps and Their Uses in Improved Network Optimization Algorithms". In: *Journal of the ACM* 34.3 (1987), pp. 596–615. DOI: 10.1145/28869.28874.

[FW93] Michael L. Fredman and Dan E. Willard. "Surpassing the information theoretic bound with fusion trees". In: *Journal of Computer and System Sciences* 47.3 (1993), pp. 424–436. DOI: 10.1016/0022-0000(93)90040-4.

[Fun+12] Isaac Fung, Konstantinos Georgiou, Jochen Könemann, and Malcolm Sharpe. "Efficient Algorithms for Solving Hypergraphic Steiner Tree Relaxations in Quasi-Bipartite Instances". In: *CoRR* abs/1202.5049 (2012). arXiv: 1202.5049.

[Gab98] Harold N. Gabow. "Algorithms for Graphic Polymatroids and Parametric $\bar{s}$-Sets". In: *Journal of Algorithms* 26.1 (1998), pp. 48–86. DOI: 10.1006/jagm.1997.0904.

[GS85] Harold N. Gabow and Matthias Stallmann. “Efficient algorithms for graphic matroid intersection and parity”. In: *Automata, Languages and Programming*. Ed. by Wilfried Brauer. Springer Berlin Heidelberg, 1985, pp. 210–220.

[GT88a] Harold N. Gabow and Robert E. Tarjan. “Algorithms for Two Bottleneck Optimization Problems”. In: *Journal of Algorithms* 9.3 (1988), pp. 411–417. DOI: 10.1016/0196-6774(88)90031-4.

[GT91] Harold N. Gabow and Robert E. Tarjan. “Faster Scaling Algorithms for General Graph Matching Problems”. In: *Journal of the ACM* 38.4 (1991), pp. 815–853. DOI: 10.1145/115234.115366.

[GW92] Harold N. Gabow and Herbert H. Westermann. “Forests, Frames, and Games: Algorithms for Matroid Sums and Applications”. English. In: *Algorithmica* 7.1-6 (1992), pp. 465–497. DOI: 10.1007/BF01758774.

[GGT16] Esther Galbrun, Aristides Gionis, and Nikolaj Tatti. “Top-*k* overlapping densest subgraphs”. In: *Data Mining and Knowledge Discovery* 30.5 (2016), pp. 1134–1165. DOI: 10.1007/s10618-016-0464-z.

[Gal79] David Gale. “The Game of Hex and the Brouwer Fixed-Point Theorem”. In: *The American Mathematical Monthly* 86.10 (1979), pp. 818–827. DOI: 10.2307/2320146.

[GKT51] David Gale, Harold William Kuhn, and Albert William Tucker. “Linear Programming and the Theory of Games”. In: *Activity Analysis of Production and Allocation*. Ed. by Tjalling C. Koopmans. Wiley, New York, 1951, pp. 317–329.

[GGT89] Giorgio Gallo, Michael D. Grigoriadis, and Robert E. Tarjan. “A Fast Parametric Maximum Flow Algorithm and Applications”. In: *SIAM J. Comput.* 18.1 (1989), pp. 30–55. DOI: 10.1137/0218003.

[GGJ77] Michael R. Garey, Ronald L. Graham, and David S. Johnson. “The Complexity of Computing Steiner Minimal Trees”. In: *SIAM Journal on Applied Mathematics* 32.4 (1977), pp. 835–859. DOI: 10.1137/0132072.

[GJ77] Michael R. Garey and David S. Johnson. “The Rectilinear Steiner Tree Problem is NP-Complete”. In: *SIAM Journal on Applied Mathematics* 32.4 (1977), pp. 826–834. DOI: 10.1137/0132071.

[GJS76] Michael R. Garey, David S. Johnson, and Larry J. Stockmeyer. “Some simplified NP-complete graph problems”. In: *Theoretical Computer Science* 1.3 (1976), pp. 237–267. DOI: 10.1016/0304-3975(76)90059-1.

[Gar96] Naveen Garg. “A 3-Approximation for the Minimum Tree Spanning *k* Vertices”. In: *37th Annual Symposium on Foundations of Computer Science, FOCS '96, Burlington, Vermont, USA, 14-16 October, 1996*. 1996, pp. 302–309. DOI: 10.1109/SFCS.1996.548489.

[Gar05] Naveen Garg. “Saving an Epsilon: A 2-approximation for the *k*-MST Problem in Graphs”. In: *Proceedings of the Thirty-seventh Annual ACM Symposium on Theory of Computing*. Baltimore, MD, USA: ACM, 2005, pp. 396–402. DOI: 10.1145/1060590.1060650.

[Gei+11] Ludwig Geistlinger, Gergely Csaba, Robert Küffner, Nicola Mulder, and Ralf Zimmer. “From sets to graphs: towards a realistic enrichment analysis of transcriptomic systems.” In: *Bioinformatics* 27.13 (2011), pp. 366–373. DOI: 10.1093/bioinformatics/btr228.

[GP07] George F. Georgakopoulos and Kostas Politopoulos. “MAX-DENSITY Revisited: a Generalization and a More Efficient Algorithm”. In: *The Computer Journal* 50.3 (2007), pp. 348–356. DOI: 10.1093/comjnl/bxl082.

[GP68] Edgar N. Gilbert and Henry O. Pollak. “Steiner Minimal Trees”. In: *SIAM Journal on Applied Mathematics* 16.1 (1968), pp. 1–29. URL: http://www.jstor.org/stable/2099400.

[GM93] Michel X. Goemans and Young-Soo Myung. “A Catalog of Steiner Tree Formulations”. In: *Networks* 23.1 (1993), pp. 19–28. DOI: 10.1002/net.3230230104.

[Goe+12] Michel X. Goemans, Neil Olver, Thomas Rothvoß, and Rico Zenklusen. “Matroids and Integrality Gaps for Hypergraphic Steiner Tree Relaxations”. In: *Proceedings of the Forty-fourth Annual ACM Symposium on Theory of Computing*. New York, NY, USA: ACM, 2012, pp. 1161–1176. DOI: 10.1145/2213977.2214081.

[GW95] Michel X. Goemans and David P. Williamson. “A General Approximation Technique for Constrained Forest Problems”. In: *SIAM Journal on Computing* 24.2 (1995), pp. 296–317. DOI: 10.1137/S0097539793242618.

[Gol84] Andrew V. Goldberg. *Finding a Maximum Density Subgraph*. Tech. rep. Berkeley, CA, USA: University of California at Berkeley, 1984.

[GK04] Andrew V. Goldberg and Alexander V. Karzanov. “Maximum Skew-symmetric Flows and Matchings”. In: *Mathematical Programming* 100.3 (2004), pp. 537–568. DOI: 10.1007/s10107-004-0505-z.

[GK97] Andrew V. Goldberg and Robert Kennedy. “Global Price Updates Help”. In: *SIAM Journal on Discrete Mathematics* 10.4 (1997), pp. 551–572. DOI: 10.1137/S0895480194281185.

[GR98] Andrew V. Goldberg and Satish Rao. “Beyond the Flow Decomposition Barrier”. In: *Journal of the ACM* 45.5 (1998), pp. 783–797. DOI: 10.1145/290179.290181.

[GT88b] Andrew V. Goldberg and Robert E. Tarjan. “A New Approach to the Maximum-flow Problem”. In: *Journal of the ACM* 35.4 (1988), pp. 921–940. DOI: 10.1145/48014.61051.

[GT90] Andrew V. Goldberg and Robert E. Tarjan. “Finding Minimum-Cost Circulations by Successive Approximation”. In: *Mathematics of Operations Research* 15.3 (1990), pp. 430–466. URL: http://www.jstor.org/stable/3689990.

[GT14] Andrew V. Goldberg and Robert E. Tarjan. “Efficient Maximum Flow Algorithms”. In: *Communications of the ACM* 57.8 (2014), pp. 82–89. DOI: 10.1145/2628036.

[GV08] Petr A. Golovach and Yngve Villanger. “Parameterized Complexity for Domination Problems on Degenerate Graphs”. In: *Graph-Theoretic Concepts in Computer Science*. Ed. by Hajo Broersma, Thomas Erlebach, Tom Friedetzky, and Daniel Paulusma. Springer Berlin Heidelberg, 2008, pp. 195–205.

[Gon09] Daniel Gonçalves. “Covering planar graphs with forests, one having a bounded maximum degree”. In: *Journal of Combinatorial Theory, Series B* 99.2 (2009), pp. 314–322. DOI: 10.1016/j.endm.2008.06.033.

[Gor19] Dan M. Gordon. *La Jolla Covering Repository*. Institute for Defense Analyses. 2019. URL: https://ljcr.dmgordon.org/cover.html.

[Grö+02] Clemens Gröpl, Stefan Hougardy, Till Nierhoff, and Hans Jügen Prömel. “Steiner trees in uniformly quasi-bipartite graphs”. In: *Information Processing Letters* 83.4 (2002), pp. 195–200. DOI: 10.1016/S0020-0190(01)00335-0.

[GL98] Roberto Grossi and Elena Lodi. "Simple planar graph partition into three forests". In: *Discrete Applied Mathematics* 84.1 (1998), pp. 121–132. DOI: 10.1016/S0166-218X(98)00007-9.

[Gur19] Gurobi Optimization, LLC. *Gurobi Optimizer Reference Manual*. 2019. URL: http://www.gurobi.com.

[GT94] Dan Gusfield and Éva Tardos. "A Faster Parametric Minimum-Cut Algorithm". English. In: *Algorithmica* 11.3 (1994), pp. 278–290. DOI: 10.1007/BF01240737.

[Hak65] Seifollah L. Hakimi. "On the degrees of the vertices of a directed graph". In: *Journal of the Franklin Institute* 279.4 (1965), pp. 290–308. DOI: 10.1016/0016-0032(65)90340-6.

[Hal35] Philip Hall. "On Representatives of Subsets". In: *Journal of the London Mathematical Society* s1-10.1 (1935), pp. 26–30. DOI: 10.1112/jlms/s1-10.37.26.

[HO92] Jianxiu Hao and James B. Orlin. "A Faster Algorithm for Finding the Minimum Cut in a Graph". In: *Proceedings of the Third Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms, 27-29 January 1992, Orlando, Florida, USA*. Ed. by Greg N. Frederickson. ACM/SIAM, 1992, pp. 165–174. URL: http://dl.acm.org/citation.cfm?id=139404.139439.

[HW08] Godfrey H. Hardy and Edward M. Wright. *An introduction to the theory of numbers*. 6th ed. Revised by D. R. Heath-Brown and J. H. Silverman, With a foreword by Andrew Wiles. Oxford University Press, Oxford, 2008.

[HS65] Juris Hartmanis and Richard E. Stearns. "On the Computational Complexity of Algorithms". In: *Transactions of the American Mathematical Society* 117 (1965), pp. 285–306. DOI: 10.2307/1994208.

[Hås01] Johan Håstad. "Some Optimal Inapproximability Results". In: *Journal of the ACM* 48.4 (2001), pp. 798–859. DOI: 10.1145/502090.502098.

[HW73] Carl Hierholzer and Christian Wiener. "Ueber die Möglichkeit, einen Linienzug ohne Wiederholung und ohne Unterbrechung zu umfahren". In: *Mathematische Annalen* 6.1 (1873), pp. 30–32. DOI: 10.1007/BF01442866.

[HK56] Alan J. Hoffman and Joseph B. Kruskal. "Integral Boundary Points of Convex Polyhedra". In: *Linear Inequalities and Related Systems*. Ed. by Harold W. Kuhn and Albert W. Tucker. Princeton University Press, NJ, 1956, pp. 223–246.

[HK73] John E. Hopcroft and Richard M. Karp. "An $n^{5/2}$ Algorithm for Maximum Matchings in Bipartite Graphs". In: *SIAM Journal on Computing* 2.4 (1973), pp. 225–231. DOI: 10.1137/0202019.

[HT73] John E. Hopcroft and Robert E. Tarjan. "Dividing a Graph into Triconnected Components". In: *SIAM Journal on Computing* 2.3 (1973), pp. 135–158. DOI: 10.1137/0202012.

[HT74] John E. Hopcroft and Robert E. Tarjan. "Efficient Planarity Testing". In: *Journal of the ACM* 21.4 (1974), pp. 549–568. DOI: 10.1145/321850.321852.

[HP99] Stefan Hougardy and Hans Jürgen Prömel. "A 1.598 Approximation Algorithm for the Steiner Problem in Graphs". In: *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms*. Baltimore, Maryland, USA: Society for Industrial and Applied Mathematics, 1999, pp. 448–453. URL: http://dl.acm.org/citation.cfm?id=314500.314599.

[Ide+02] Trey Ideker, Owen Ozier, Benno Schwikowski, and Andrew F. Siegel. "Discovering regulatory and signalling circuits in molecular interaction networks". In: *Proceedings of the Tenth International Conference on Intelligent Systems for Molecular Biology, August 3-7, 2002, Edmonton, Alberta, Canada*. 2002, pp. 233–240.

[Jai01] Kamal Jain. "A Factor 2 Approximation Algorithm for the Generalized Steiner Network Problem". In: *Combinatorica* 21.1 (2001), pp. 39–60. DOI: 10.1007/s004930170004.

[JY17] Hongbi Jiang and Daqing Yang. "Decomposing a Graph into Forests: The Nine Dragon Tree Conjecture is True". In: *Combinatorica* 37.6 (2017), pp. 1125–1137. DOI: 10.1007/s00493-016-3390-1.

[Joh74] David S. Johnson. "Approximation Algorithms for Combinatorial Problems". In: *Journal of Computer and System Sciences* 9.3 (1974), pp. 256–278. DOI: 10.1016/S0022-0000(74)80044-9.

[JA97] Kurt Jörnsten and Løkketangen Arne. "Tabu search for weighted k-cardinality trees". In: *Asia-Pacific Journal of Operational Research* 14.2 (1997), pp. 9–26.

[Juh+18] Daniel Juhl, David M. Warme, Pawel Winter, and Martin Zachariasen. "The GeoSteiner software package for computing Steiner trees in the plane: an updated computational study". In: *Mathematical Programming Computation* (2018). DOI: 10.1007/s12532-018-0135-8.

[KNR92] Sampath Kannan, Moni Naor, and Steven Rudich. "Implicit Representation of Graphs". In: *SIAM Journal on Discrete Mathematics* 5.4 (1992), pp. 596–603. DOI: 10.1137/0405049.

[KKT95] David R. Karger, Philip N. Klein, and Robert E. Tarjan. "A Randomized Linear-time Algorithm to Find Minimum Spanning Trees". In: *Journal of the ACM* 42.2 (1995), pp. 321–328. DOI: 10.1145/201019.201022.

[Kar84] Narendra Karmarkar. "A new polynomial-time algorithm for linear programming". In: *Combinatorica* 4.4 (1984), pp. 373–396. DOI: 10.1007/BF02579150.

[Kar72] Richard M. Karp. "Reducibility Among Combinatorial Problems". In: *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York.* 1972, pp. 85–103.

[KZ97] Marek Karpinski and Alexander Zelikovsky. "New Approximation Algorithms for the Steiner Tree Problems". In: *Journal of Combinatorial Optimization* 1.1 (1997), pp. 47–65. DOI: 10.1023/A:1009758919736.

[Kar74] Alexander Karzanov. "Determining the maximal flow in a network by the method of preflows". In: *Soviet Mathematics Doklady* 15.2 (1974), pp. 434–437.

[Kar73] Alexander V. Karzanov. "O nakhozhdenii maksimal'nogo potoka v setyakh spetsial'nogo vida i nekotorykh prilozheniyakh [On finding a maximum flow in a network with special structure and some applications, in Russian]". In: *Matematicheskie Voprosy Upravleniya Proizvodstvom*. Ed. by L. A. Lyusternik. Vol. 15. Moscow State University Press, 1973, pp. 81–94.

[Kha79] Leonid G. Khachiyan. "A polynomial algorithm in linear programming". In: *Soviet Mathematics Doklady* 20 (1979), pp. 191–194.

[Kho06] Subhash Khot. "Ruling Out PTAS for Graph Min-Bisection, Dense k-Subgraph, and Bipartite Clique". In: *SIAM Journal on Computing* 36.4 (2006), pp. 1025–1071. DOI: 10.1137/S0097539705447037.

[KS09a] Samir Khuller and Barna Saha. "On Finding Dense Subgraphs". In: *Automata, Languages and Programming, 36th International Colloquium, ICALP 2009, Rhodes, Greece, July 5-12, 2009, Proceedings, Part I.* 2009, pp. 597–608. DOI: 10.1007/978-3-642-02927-1_50.

[KS09b] Samir Khuller and Barna Saha. "On Finding Dense Subgraphs". Unpublished. 2009. URL: http://www.cs.umd.edu/users/samir/grant/journal-combinatorica.pdf.

[Kim+13] Seog-Jin Kim, Alexandr V. Kostochka, Douglas B. West, Hehui Wu, and Xuding Zhu. "Decomposition of Sparse Graphs into Forests and a Graph with Bounded Degree". In: *Journal of Graph Theory* 74.4 (2013), pp. 369–391. DOI: 10.1002/jgt.21711.

[KM72] Victor Klee and George J. Minty. "How good is the simplex algorithm?" In: *Inequalities, III (Proc. Third Sympos., Univ. California, Los Angeles, Calif., 1969; dedicated to the memory of Theodore S. Motzkin)*. Ed. by Oved Shisha. Academic Press, New York, 1972, pp. 159–175.

[KM98] Thorsten Koch and Alexander Martin. "Solving Steiner tree problems in graphs to optimality". In: *Networks* 32.3 (1998), pp. 207–232. DOI: 10.1002/(SICI)1097-0037(199810)32:3<207::AID-NET5>3.0.CO;2-O.

[KPT11] Jochen Könemann, David Pritchard, and Kunlun Tan. "A partition-based relaxation for Steiner trees". In: *Mathematical Programming* 127.2 (2011), pp. 345–370. DOI: 10.1007/s10107-009-0289-2.

[KPP16] Tsvi Kopelowitz, Seth Pettie, and Ely Porat. "Higher Lower Bounds from the 3SUM Conjecture". In: *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*. Ed. by Robert Krauthgamer. SIAM, 2016, pp. 1272–1287. DOI: 10.1137/1.9781611974331.ch89. URL: https://doi.org/10.1137/1.9781611974331.ch89.

[KV18] Bernhard Korte and Jens Vygen. *Combinatorial Optimization: Theory and Algorithms*. 6th. Springer Publishing Company, Incorporated, 2018.

[KP94] Guy Kortsarz and David Peleg. "Generating Sparse 2-spanners". In: *Journal of Algorithms* 17.2 (1994), pp. 222–236. DOI: 10.1006/jagm.1994.1032.

[Kow06] Łukasz Kowalik. "Approximation Scheme for Lowest Outdegree Orientation and Graph Density Measures". English. In: *Algorithms and Computation*. Ed. by Tetsuo Asano. Vol. 4288. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2006, pp. 557–566. DOI: 10.1007/11940128_56.

[Law76] Eugene L. Lawler. *Combinatorial Optimization: Networks and Matroids*. Holt, Rinehart and Winston, 1976. Chap. 4, pp. 125–127.

[LS14] Yin T. Lee and Aaron Sidford. "Path Finding Methods for Linear Programming: Solving Linear Programs in $\tilde{O}(\sqrt{rank})$ Iterations and Faster Algorithms for Maximum Flow". In: *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*. 2014, pp. 424–433. DOI: 10.1109/FOCS.2014.52.

[LW10] Christoph Lenzen and Roger Wattenhofer. "Minimum Dominating Set Approximation in Graphs of Bounded Arboricity". In: *Distributed Computing, 24th International Symposium, DISC 2010, Cambridge, MA, USA, September 13-15, 2010. Proceedings*. Ed. by Nancy A. Lynch and Alexander A. Shvartsman. Vol. 6343. Lecture Notes in Computer Science. Springer, 2010, pp. 510–524. DOI: 10.1007/978-3-642-15763-9_48.

[LK14] Jure Leskovec and Andrej Krevl. *SNAP Datasets: Stanford Large Letwork Dataset Collection*. http://snap.stanford.edu/data. 2014.

[Lev73] Leonid A. Levin. "Universal sequential search problems [in Russian]". In: *Problems of Information Transmission* 9.3 (1973), pp. 265–266.

[Lew78] John M. Lewis. "On the Complexity of the Maximum Subgraph Problem". In: *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing*. San Diego, California, USA: ACM, 1978, pp. 265–274. DOI: 10.1145/800133.804356.

[Lic82] David Lichtenstein. "Planar Formulae and Their Uses". In: *SIAM Journal on Computing* 11.2 (1982), pp. 329–343. DOI: 10.1137/0211025.

[Lip10] Richard J. Lipton. *The P=NP question and Gödel's lost letter*. New York: Springer, 2010.

[Lju04] Ivana Ljubić. "Exact and memetic algorithms for two network design problems". PhD thesis. Technische Universität Wien, 2004. URL: https://www.ac.tuwien.ac.at/files/archive/www.ads.tuwien.ac.at/publications/bib/pdf/ljubicPhD.pdf.

[Lju+06] Ivana Ljubić, René Weiskircher, Ulrich Pferschy, Gunnar W. Klau, Petra Mutzel, and Matteo Fischetti. "An Algorithmic Framework for the Exact Solution of the Prize-Collecting Steiner Tree Problem". In: *Mathematical Programming* 105.2-3 (2006), pp. 427–449. DOI: 10.1007/s10107-005-0660-x.

[Lov75] László Lovász. "On the ratio of optimal integral and fractional covers". In: *Discrete Mathematics* 13.4 (1975), pp. 383–390. DOI: 10.1016/0012-365X(75)90058-8.

[LZ93] Dmitrii D. Lozovanu and Alexander Zelikovsky. "Minimal and bounded trees". In: Tezele Congresului XVIII al Academiei Romano-Americane. Chişinău, Moldova: American-Romanian Academy of Arts and Sciences, 1993, pp. 25–26.

[Luc82] Édouard Lucas. Récréations mathématiques [Mathematical Recreations, in French] 1. Gauthier-Villars, 1882.

[LR01] Abilio Lucena and Mauricio G. C. Resende. "Generating lower bounds for the prize collecting Steiner problem in graphs". In: *Electronic Notes in Discrete Mathematics* 7 (2001). Brazilian Symposium on Graphs, Algorithms and Combinatorics, pp. 70–73. DOI: 10.1016/S1571-0653(04)00227-6.

[Ma+17] Xiuli Ma, Guangyu Zhou, Jingbo Shang, Jingjing Wang, Jian Peng, and Jiawei Han. "Detection of Complexes in Biological Networks Through Diversified Dense Subgraph Mining". In: *Journal of Computational Biology* 24.9 (2017), pp. 923–941. DOI: 10.1089/cmb.2017.0037.

[Mąd13] Aleksander Mądry. "Navigating Central Path with Electrical Flows: From Flows to Matchings, and Back". In: *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*. 2013, pp. 253–262. DOI: 10.1109/FOCS.2013.35.

[Mąd16] Aleksander Mądry. "Computing Maximum Flow with Augmenting Electrical Flows". In: *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, New Brunswick, New Jersey, USA*. 2016, pp. 593–602. DOI: 10.1109/FOCS.2016.70.

[MB83] David W. Matula and Leland L. Beck. "Smallest-last Ordering and Clustering and Graph Coloring Algorithms". In: *J. ACM* 30.3 (1983), pp. 417–427. DOI: 10.1145/2402.322385.

[McG+15a] Andrew McGregor, David Tench, Sofya Vorotnikova, and Hoa T. Vu. "Densest Subgraph in Dynamic Graph Streams". In: *Mathematical Foundations of Computer Science 2015 - 40th International Symposium, MFCS 2015, Milan, Italy, August 24-28, 2015, Proceedings, Part II*. 2015, pp. 472–482. DOI: 10.1007/978-3-662-48054-0_39.

[McG+15b] Andrew McGregor, David Tench, Sofya Vorotnikova, and Hoa T. Vu. "Densest Subgraph in Dynamic Graph Streams". In: *CoRR* abs/1506.04417 (2015). arXiv: 1506.04417.

[MV80] Silvio Micali and Vijay V. Vazirani. "An $\mathcal{O}(\sqrt{|V|}|E|)$ algorithm for finding maximum matching in general graphs". In: *21st Annual Symposium on Foundations of Computer Science (sfcs 1980)*. 1980, pp. 17–27. DOI: 10.1109/SFCS.1980.12.

[Mil+11] Tijana Milenković, Vesna Memišević, Anthony Bonato, and Nataša Pržulj. "Dominating Biological Networks". In: *PLOS ONE* 6.8 (2011), pp. 1–12. DOI: 10.1371/journal.pone.0023016.

[MU05] Michael Mitzenmacher and Eli Upfal. *Probability and computing – randomized algorithms and probabilistic analysis*. Cambridge University Press, 2005.

[Mon+12] Mickael Montassier, Patrice Ossona de Mendez, André Raspaud, and Xuding Zhu. "Decomposing a graph into forests". In: *Journal of Combinatorial Theory, Series B* 102.1 (2012), pp. 38–52. DOI: 10.1016/j.jctb.2011.04.001.

[Nas61] Crispin St. J. A. Nash-Williams. "Edge-Disjoint Spanning Trees of Finite Graphs". In: *Journal of the London Mathematical Society* 36.1 (1961), pp. 445–450. DOI: 10.1112/jlms/s1-36.1.445.

[Nas64] Crispin St. J. A. Nash-Williams. "Decomposition of Finite Graphs Into Forests". In: *Journal of the London Mathematical Society* 39.1 (1964), pp. 12–12. DOI: 10.1112/jlms/s1-39.1.12.

[Nas+17] Muhammad A. U. Nasir, Aristides Gionis, Gianmarco De Francisci Morales, and Sarunas Girdzijauskas. "Fully Dynamic Algorithm for Top-k Densest Subgraphs". In: *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, CIKM 2017, Singapore, November 06 - 10, 2017*. 2017, pp. 1817–1826. DOI: 10.1145/3132847.3132966.

[Naz+16] Maryam Nazarieh, Andreas Wiese, Thorsten Will, Mohamed Hamed, and Volkhard Helms. "Identification of key player genes in gene regulatory networks". In: *BMC Systems Biology* 10.1 (2016), p. 88. DOI: 10.1186/s12918-016-0329-5.

[Neu38] John von Neumann. "Über ein ökonomisches Gleichungssystem und eine Verallgemeinerung des Brouwerschen Fixpunktsatzes [On an economic system of equations and a generalization of the Brouwer fixed-point theorem, in German]". In: *Ergebnisse eines Mathematischen Seminars*. Ed. by Karl Menger. Vienna, 1938.

[Neu63] John von Neumann. “Discussion of a Maximum Problem”. In: *John von Neumann, Collected Works*. Ed. by A. H. Taub. Vol. VI. Pergamon Press, Oxford, 1963, pp. 89–95.

[Orl13] James B. Orlin. “Max Flows in $\mathcal{O}(nm)$ Time, or Better”. In: *Proceedings of the Forty-fifth Annual ACM Symposium on Theory of Computing*. Palo Alto, California, USA: ACM, 2013, pp. 765–774. DOI: 10.1145/2488608.2488705.

[Oxl06] James G. Oxley. *Matroid Theory (Oxford Graduate Texts in Mathematics)*. New York, NY, USA: Oxford University Press, Inc., 2006.

[PD06] Mihai Pătrașcu and Erik D. Demaine. “Logarithmic Lower Bounds in the Cell-Probe Model”. In: *SIAM Journal on Computing* 35.4 (2006), pp. 932–963. DOI: 10.1137/S0097539705447256.

[Pay86] Charles Payan. “Graphes Équilibrés et Arboricité Rationnelle [Balanced graphs and fractional arboricity, in French]”. In: *European Journal of Combinatorics* 7.3 (1986), pp. 263–270. DOI: 10.1016/S0195-6698(86)80032-4.

[PI79] William H. Payne and Frederick M. Ives. “Combination Generators”. In: *ACM Transactions on Mathematical Software* 5.2 (1979), pp. 163–172. DOI: 10.1145/355826.355830.

[Pét35] Rózsa Péter. “Konstruktion nichtrekursiver Funktionen [Construction of nonrecursive functions, in German]”. In: *Mathematische Annalen* 111.1 (1935), pp. 42–60. DOI: 10.1007/BF01472200.

[PR02] Seth Pettie and Vijaya Ramachandran. “An Optimal Minimum Spanning Tree Algorithm”. In: *Journal of the ACM* 49.1 (2002), pp. 16–34. DOI: 10.1145/505241.505243.

[PQ82] Jean-Claude Picard and Maurice Queyranne. “A Network Flow Solution to Some Nonlinear 0-1 Programming Problems, with Applications to Graph Theory”. In: *Networks* 12.2 (1982), pp. 141–159. DOI: 10.1002/net.3230120206.

[PST91] Serge A. Plotkin, David B. Shmoys, and Éva Tardos. “Fast approximation algorithms for fractional packing and covering problems”. In: *Proceedings 32nd Annual Symposium of Foundations of Computer Science*. 1991, pp. 495–504. DOI: 10.1109/SFCS.1991.185411.

[PD01] Tobias Polzin and Siavash Vahdati Daneshmand. “A comparison of Steiner tree relaxations”. In: *Discrete Applied Mathematics* 112.1 (2001). Combinatorial Optimization Symposium, Selected Papers, pp. 241–261. DOI: 10.1016/S0166-218X(00)00318-8.

[PD03] Tobias Polzin and Siavash Vahdati Daneshmand. "On Steiner trees and minimum spanning trees in hypergraphs". In: *Operations Research Letters* 31.1 (2003), pp. 12–20. DOI: 10.1016/S0167-6377(02)00185-2.

[Pou90] Johannes A. La Poutré. "New Techniques for the Union-Find Problems". In: *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms, 22-24 January 1990, San Francisco, California, USA.* Ed. by David S. Johnson. SIAM, 1990, pp. 54–63. URL: http://dl.acm.org/citation.cfm?id=320176.320182.

[PS02] Hans Jürgen Prömel and Angelika Steger. *The Steiner Tree Problem: A Tour through Graphs, Algorithms, and Complexity*. Advanced Lectures in Mathematics. Friedr. Vieweg & Sohn Verlagsgesellschaft, 2002. DOI: 10.1007/978-3-322-80291-0.

[RV95] Sridhar Rajagopalan and Vijay V. Vazirani. "Logarithmic approximation of minimum weight k trees". Unpublished manuscript. 1995.

[RV99] Sridhar Rajagopalan and Vijay V. Vazirani. "On the Bidirected Cut Relaxation for the Metric Steiner Tree Problem". In: *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms*. Baltimore, Maryland, USA: Society for Industrial and Applied Mathematics, 1999, pp. 742–751. URL: http://dl.acm.org/citation.cfm?id=314500.314909.

[Rav+96] R. Ravi, Ravi Sundaram, Madhav V. Marathe, Daniel J. Rosenkrantz, and Sekharipuram S. Ravi. "Spanning Trees – Short or Small". In: *SIAM Journal on Discrete Mathematics* 9.2 (1996), pp. 178–200. DOI: 10.1137/S0895480194266331.

[Ray83] Victor J. Rayward-Smith. "The computation of nearly minimal Steiner trees in graphs". In: *International Journal of Mathematical Education in Science and Technology* 14.1 (1983), pp. 15–23. DOI: 10.1080/0020739830140103.

[RK19] Daniel Rehfeldt and Thorsten Koch. "Combining NP-Hard Reduction Techniques and Strong Heuristics in an Exact Algorithm for the Maximum-Weight Connected Subgraph Problem". In: *SIAM Journal on Optimization* 29.1 (2019), pp. 369–398. DOI: 10.1137/17M1145963.

[RKM19] Daniel Rehfeldt, Thorsten Koch, and Stephen J. Maher. "Reduction techniques for the prize collecting Steiner tree problem and the maximum-weight connected subgraph problem". In: *Networks* 73.2 (2019), pp. 206–233. DOI: 10.1002/net.21857.

[RS14] Christian Reiher and Lisa Sauermann. “Nash-Williams’ theorem on decomposing graphs into forests”. In: *Mathematika* 60.1 (2014), pp. 32–36. DOI: 10.1112/S0025579313000119.

[Rhy70] John M. W. Rhys. “A Selection Problem of Shared Fixed Costs and Network Flows”. In: *Management Science* 17.3 (1970), pp. 200–207. DOI: 10.1287/mnsc.17.3.200.

[RZ00] Gabriel Robins and Alexander Zelikovsky. “Improved Steiner Tree Approximation in Graphs”. In: *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*. San Francisco, California, USA: Society for Industrial and Applied Mathematics, 2000, pp. 770–779. URL: http://dl.acm.org/citation.cfm?id=338219.338638.

[RZ05] Gabriel Robins and Alexander Zelikovsky. “Tighter Bounds for Graph Steiner Tree Approximation”. In: *SIAM Journal on Discrete Mathematics* 19.1 (2005), pp. 122–134. DOI: 10.1137/S0895480101393155.

[Sah+10] Barna Saha, Allison Hoch, Samir Khuller, Louiqa Raschid, and Xiao-Ning Zhang. “Dense Subgraphs with Restrictions and Applications to Gene Annotation Graphs”. In: *Research in Computational Molecular Biology, 14th Annual International Conference, RECOMB 2010, Lisbon, Portugal, April 25-28, 2010. Proceedings*. 2010, pp. 456–472. DOI: 10.1007/978-3-642-12683-3_30.

[SU13] Edward R. Scheinerman and Daniel H. Ullman. *Fractional Graph Theory: A Rational Approach to the Theory of Graphs*. Minola, N.Y.: Dover Publications, 2013.

[Sch90] Walter Schnyder. “Embedding Planar Graphs on the Grid”. In: *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms*. San Francisco, California, USA: Society for Industrial and Applied Mathematics, 1990, pp. 138–148. URL: http://dl.acm.org/citation.cfm?id=320176.320191.

[Sch99] Alexander Schrijver. *Theory of Linear and Integer programming*. Wiley-Interscience series in discrete mathematics and optimization. Wiley, 1999.

[SW11] Robert Sedgewick and Kevin Wayne. *Algorithms*. 4th ed. Addison-Wesley, 2011.

[Sey79] Paul D. Seymour. “On Multi-Colourings of Cubic Graphs, and Conjectures of Fulkerson and Tutte”. In: *Proceedings of the London Mathematical Society* s3-38.3 (1979), pp. 423–460. DOI: 10.1112/plms/s3-38.3.423.

[Sho77] Naum Z. Shor. “Cut-Off Method with Space Extension in Convex Programming Problems”. In: *Cybernetics* 13 (1977), pp. 94–65.

[Sim72] José M. S. Simões-Pereira. “On subgraphs as matroid cells”. In: *Mathematische Zeitschrift* 127.4 (1972), pp. 315–322. DOI: 10.1007/BF01111390.

[Sin00] Amitabh Sinha. “Steiner Trees: First Summer Paper for the PhD program at GSIA”. 2000.

[ST81] Daniel D. Sleator and Robert E. Tarjan. “A Data Structure for Dynamic Trees”. In: *Proceedings of the 13th Annual ACM Symposium on Theory of Computing, May 11-13, 1981, Milwaukee, Wisconsin, USA*. 1981, pp. 114–122. DOI: 10.1145/800076.802464.

[ST83] Daniel D. Sleator and Robert Endre Tarjan. “A Data Structure for Dynamic Trees”. In: *J. Comput. Syst. Sci.* 26.3 (1983), pp. 362–391. DOI: 10.1016/0022-0000(83)90006-5.

[Soa16] Robert I. Soare. *Turing Computability – Theory and Applications*. Theory and Applications of Computability. Springer, 2016. DOI: 10.1007/978-3-642-31933-4.

[SG10] Mauro Sozio and Aristides Gionis. “The community-search problem and how to plan a successful cocktail party”. In: *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, July 25-28, 2010*. 2010, pp. 939–948. DOI: 10.1145/1835804.1835923.

[Sub+05] Aravind Subramanian et al. “Gene set enrichment analysis: A knowledge-based approach for interpreting genome-wide expression profiles”. In: *Proceedings of the National Academy of Sciences* 102.43 (2005), pp. 15545–15550. DOI: 10.1073/pnas.0506580102.

[Sud27] Gabriel Sudan. “Sur le nombre transfini ω^ω [On the transfinite number ω^ω, in French]”. In: *Bulletin mathématique de la Société Roumaine des Sciences* 30.1 (1927), pp. 11–30. URL: http://www.jstor.org/stable/43769875.

[SW68] George Szekeres and Herbert S. Wilf. “An inequality for the chromatic number of a graph”. In: *Journal of Combinatorial Theory* 4.1 (1968), pp. 1–3. DOI: 10.1016/S0021-9800(68)80081-X.

[Tar75] Robert E. Tarjan. “Efficiency of a Good But Not Linear Set Union Algorithm”. In: *Journal of the ACM* 22.2 (1975), pp. 215–225. DOI: 10.1145/321879.321884.

[Tar79] Robert E. Tarjan. “A class of algorithms which require nonlinear time to maintain disjoint sets”. In: *Journal of Computer and System Sciences* 18.2 (1979), pp. 110–127. DOI: 10.1016/0022-0000(79)90042-4.

[TW07] Robert E. Tarjan and Renato F. Werneck. “Dynamic Trees in Practice”. In: *Experimental Algorithms*. Ed. by Camil Demetrescu. Springer Berlin Heidelberg, 2007, pp. 80–93.

[Thi03] Martin Thimm. “On the approximability of the Steiner tree problem”. In: *Theoretical Computer Science* 295.1 (2003). Mathematical Foundations of Computer Science, pp. 387–402. DOI: 10.1016/S0304-3975(02)00414-0.

[TG16] Bio Mikaila Toko Worou and Jérôme Galtier. “Fast approximation for computing the fractional arboricity and extraction of communities of a graph”. In: *Discrete Applied Mathematics* 213 (2016), pp. 179–195. DOI: 10.1016/j.dam.2014.10.023.

[Tom76] Nobuaki Tomizawa. “Strongly irreducible matroids and principal partition of a matroid (in Japanese)”. In: *Transactions of the Institute of Electronics and Communications Engineers of Japan*. Vol. 59A. 1976, pp. 83–91.

[Tri93] Eberhard Triesch. *Halving graphs is NP-complete*. Working paper 93 790. Research Institute for Discrete Mathematics, University of Bonn, 1993.

[Tso+13] Charalampos E. Tsourakakis, Francesco Bonchi, Aristides Gionis, Francesco Gullo, and Maria A. Tsiarli. “Denser than the Densest Subgraph: Extracting Optimal Quasi-Cliques with Quality Guarantees”. In: *The 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2013, Chicago, IL, USA, August 11-14, 2013*. 2013, pp. 104–112. DOI: 10.1145/2487575.2487645.

[Tur36] Alan M. Turing. “On Computable Numbers, with an Application to the Entscheidungsproblem”. In: *Proceedings of the London Mathematical Society* 2.42 (1936), pp. 230–265.

[Tut65] William T. Tutte. “Lectures on matroids”. In: *Journal of Research of the National Bureau of Standards, B* 69 (1965), pp. 1–47.

[VKP12] Elena Valari, Maria Kontaki, and Apostolos N. Papadopoulos. “Discovery of Top-k Dense Subgraphs in Dynamic Graph Collections”. In: *Scientific and Statistical Database Management - 24th International Conference,* *SSDBM 2012, Chania, Crete, Greece, June 25-27, 2012. Proceedings*. 2012, pp. 213–230. DOI: 10.1007/978-3-642-31235-9_14.

[Vaz01] Vijay V. Vazirani. *Approximation algorithms*. Springer, 2001. DOI: 10.1007/978-3-662-04565-7.

[Ven04] Venkat Venkateswaran. "Minimizing maximum indegree". In: *Discrete Applied Mathematics* 143.1–3 (2004), pp. 374–378. DOI: 10.1016/j.dam.2003.07.007.

[War98] David M. Warme. "Spanning Trees in Hypergraphs with Applications to Steiner Trees". AAI9840474. PhD thesis. Charlottesville, VA, USA: University of Virginia, 1998. DOI: 10.18130/V3ZG4B.

[Wer06] Renato F. Werneck. "Design and Analysis of Data Structures for Dynamic Trees". PhD thesis. Princeton University, 2006. URL: https://www.cs.princeton.edu/research/techreps/TR-750-06.

[Wes88] Herbert H. Westermann. "Efficient Algorithms For Matroid Sums". PhD thesis. USA: University of Colorado Boulder, 1988.

[Whi88] Walter Whiteley. "The Union of Matroids and the Rigidity of Frameworks". In: *SIAM Journal on Discrete Mathematics* 1.2 (1988), pp. 237–255. DOI: 10.1137/0401025.

[Whi35] Hassler Whitney. "On the Abstract Properties of Linear Dependence". In: *American Journal of Mathematics* 57.3 (1935), pp. 509–533. DOI: 10.2307/2371182.

[Won84] Richard T. Wong. "A dual ascent approach for Steiner tree problems on a directed graph". In: *Mathematical Programming* 28.3 (1984), pp. 271–287. DOI: 10.1007/BF02612335.

[Yan78] Mihalis Yannakakis. "Node-and Edge-deletion NP-complete Problems". In: *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing*. San Diego, California, USA: ACM, 1978, pp. 253–264. DOI: 10.1145/800133.804355.

[Zas82] Thomas Zaslavsky. "Bicircular Geometry and the Lattice of Forests of a Graph". In: *The Quarterly Journal of Mathematics* 33.4 (1982), pp. 493–511. DOI: 10.1093/qmath/33.4.493.

[Zha+08] Xing-Ming Zhao, Rui-Sheng Wang, Luonan Chen, and Kazuyuki Aihara. "Uncovering signal transduction networks from high-throughput data by integer linear programming". In: *Nucleic Acids Research* 36.9 (2008), e48–e48. DOI: 10.1093/nar/gkn145.

[Zus72] Konrad Zuse. *Der Plankalkül [in German]*. Vol. 63. Berichte der Gesellschaft für Mathematik und Datenverarbeitung. Gesellschaft für Mathematik und Datenverarbeitung, 1972.