PARAMETERIZED APPROACHES FOR LARGE-SCALE OPTIMIZATION PROBLEMS

A Dissertation

by

AUSTIN LOYD BUCHANAN

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Chair of Committee,	Sergiy Butenko
Committee Members,	Jianer Chen
	Kiavash Kianfar
	Erick Moreno-Centeno
Head of Department,	César O. Malavé

August 2015

Major Subject: Industrial and Systems Engineering

ABSTRACT

In this dissertation, we study challenging discrete optimization problems from the perspective of parameterized complexity. The usefulness of this type of analysis is twofold. First, it can lead to efficient algorithms for large-scale problem instances. Second, the analysis can provide a rigorous explanation for why challenging problems might appear relatively easy in practice. We illustrate the approach on several different problems, including: the maximum clique problem in sparse graphs; 0-1 programs with many conflicts; and the node-weighted Steiner tree problem with few terminal nodes. We also study polyhedral counterparts to fixed-parameter tractable algorithms. Specifically, we provide fixed-parameter tractable extended formulations for independent set in tree-like graphs and for cardinality-constrained vertex covers.

DEDICATION

To my parents.

ACKNOWLEDGEMENTS

First, I am extremely grateful to have Dr. Sergiy Butenko as my dissertation advisor. Since day one, he has treated me with the utmost respect and made me feel like a colleague instead of a student. He has always held a positive attitude towards me. Many times I have knocked on his door asking—"Do you have a few minutes?"—to which he responds—"For you, Austin, always." He has always given me the freedom to pick my own research topics, which, given my independent nature, has been crucial to my happiness and success.

I have had much help from within Texas A&M. My committee members Drs. Jianer Chen, Kiavash Kianfar, and Erick Moreno-Centeno have been very supportive. Drs. Chen and Kianfar have been some of the best teachers I have ever had. (Unfortunately, I have not taken any of Dr. Moreno's courses.) As department head, Dr. Cesar Malave makes everyone feel welcome and is always smiling. I also want to thank Dr. Guy Curry and Judy Meeks for helping to secure my Doctoral Merit Fellowship, which was a significant financial boost and factored into my decision to attend Texas A&M.

There are several faculty outside of Texas A&M that have been instrumental in earning my Ph.D. I want to thank Dr. Tieming Liu at Oklahoma State University for hiring me as an undergraduate researcher and for being my senior design project advisor. Without his encouragement, I may have never pursued a Ph.D. Professors Vladimir Boginski, Panos Pardalos, and Oleg Prokopyev have been great to work with, and I thank all of them for writing reference letters for me over the years for scholarships and faculty positions. Dr. Eduardo Pasiliao, my summer mentor at the

AFRL MMOI, has also been extremely helpful.

There are many Ph.D. students, both inside and outside of Texas A&M, that have helped me along. My collaborations with Je Sang Sung, Anurag Verma, Jose Walteros, and Yiming Wang have been very fruitful and I thank all of them for their friendship as well. My colleagues Nannan, Sasha, Su, Xin, Yu, Zimo, and others have helped to make my time at Texas A&M more enjoyable. I also fondly remember pastichio, poker, and/or white-sand beaches with Behdad, Chrys, Foad, Gabe, Juan, Oleg S., Sasha, Serdar, Vika, and Vova and the rest of the MMOI crew.

I have been fortunate to have had my Ph.D. studies funded from a variety of sources. This includes fellowships, scholarships, and instructor positions from Texas A&M, the Dwight Look College of Engineering, and the Department of Industrial and Systems Engineering. Other scholarships and travel funds have come from the Institute of Industrial Engineers, the Material Handling Education Foundation, the George Bush Presidential Library Foundation, and the Texas Engineering Foundation. My summer research has been supported primarily by the AFRL Mathematical Modeling and Optimization Institute. Partial support by AFOSR under grants FA9550-12-1-0103 and FA8651-12-2-0011 is also gratefully acknowledged.

Finally, I would like to thank my family for all of their support over the years.

TABLE OF CONTENTS

Page

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

TABLE Page

1. INTRODUCTION

Consider the Facebook graph. Each Facebook account is represented by a vertex, and two vertices are connected by an edge when the two people are friends. A *clique* in the Facebook graph (a group of people where everyone knows everyone else in the group) represents a tightly knit cluster. Or, consider a protein-protein interaction network from the field of bioinformatics. Each vertex represents a protein, and when two proteins are noticed to interact with each other, then the corresponding vertices are connected by an edge. The detection of a large clique in this protein-protein interaction network might lead to the discovery of a new protein complex [28]. Or, consider the stock market graph [18]. Here, there is a vertex for each stock, and two stocks' vertices are connected by an edge if their prices over time are negatively correlated. A large clique in this graph represents a diversified portfolio. These are just some of the many applications of clique finding.

Unfortunately, the problem of finding a largest clique in an arbitrary graph is challenging. Still, due to the numerous applications of clique-finding in social network analysis, bioinformatics, finance, and elsewhere, one may want to solve the problem anyway. Fortunately, for many real-life instances, the problem seems to be relatively easy. Indeed, in [107], we were able to solve million-vertex instances of this problem in just a few seconds. The computational results were nice, but we were unable to prove any nontrivial bounds on the runtime of our approach. Later on, we realized that there is a nice algorithm with a provable worst-case runtime for the maximum clique problem [26]. Its runtime depends polynomially on the number of vertices, but exponentially in the *degeneracy* of the graph—a common measure of a graph's sparsity. This is nice because many real-life graphs are sparse. In the case

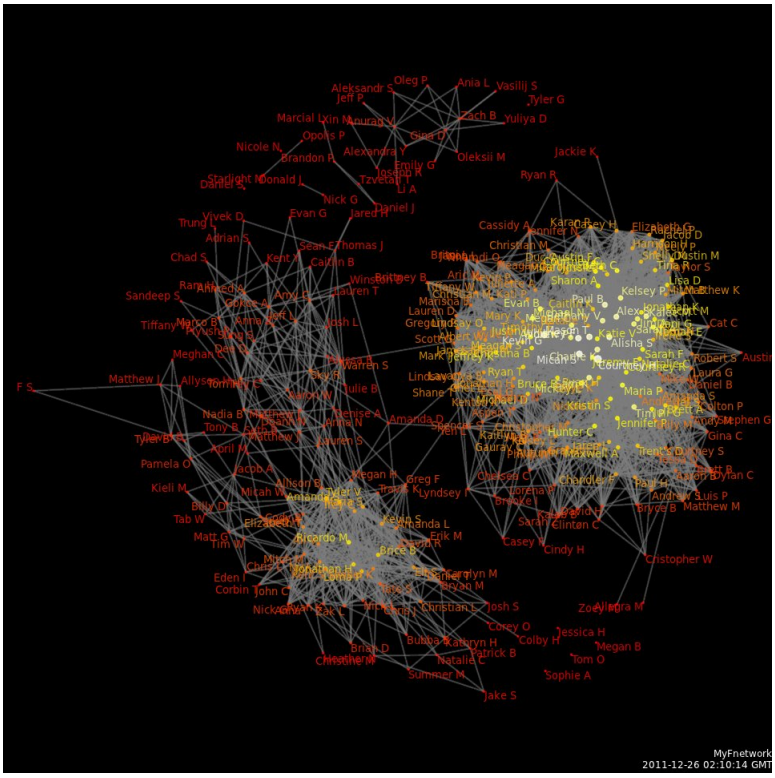

Figure 1.1: The author's Facebook friendship graph in 2011.

of Facebook-style graphs (where users are limited to 5000 friends), this results in an efficient algorithm. See Figure 1.1 for a subgraph of the Facebook graph.

In the era of Big Data, there is an increased need to solve *very* large instances of problems. However, it has been noted that "most interesting problems are NP-hard." This leads many people to give up on exact methods and instead rely upon heuristics that often guarantee little in terms of solution quality, or, if we are lucky, approximation algorithms whose output can be several times optimal. As demonstrated by our success with the maximum clique problem, this need not be the default reaction. Many problems have natural associated *parameters* (such as degeneracy for clique), such that when the parameter is relatively small, then the problem can be solved

quickly. This is the *fixed-parameter tractable* approach which has been spreading throughout computer science. Due to the relative youth of the field of *parameterized complexity*, this type of analysis has not been applied as frequently in the operations research (OR) and mathematical programming (MP) communities. This new perspective raises many questions that have scarcely been studied. In this dissertation, we answer some of these questions, but there is much left to do.

1.1 Preliminaries

In Section 1.2 we detail the contributions of this dissertation. However, we will first need some terminology and notation. This section serves to introduce the reader to basic concepts like graphs, computational complexity, and extended formulations. More details will be given later in the dissertation as they are needed.

1.1.1 Graph terminology

Most of the problems discussed in this dissertation are defined with respect to a simple graph. A simple graph $G = (V, E)$ is a pair, where V is a finite set of *vertices*, and $E \subseteq \binom{V}{2}$ is a collection of unordered pairs of vertices. Here, $\binom{V}{2} := \{\{u, v\} \mid u, v \in V, u \neq v\}$. An object $\{u, v\} \in E$ is called an *edge*, and its *endpoints* u and v are said to be *neighbors* or *adjacent*. The (open) neighborhood of a vertex u in a graph G is denoted $N_G(u) := \{v \in V \mid \{u, v\} \in E\}$. When the graph in question is clear, we simply write $N(u)$. The number $|N(u)|$ of neighbors of a vertex u is the *degree* of u. The closed neighborhood of u is denoted $N[u] := N(u) \cup \{u\}$. A graph $G' = (V', E')$ is said to be a *subgraph* of $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$. The subgraph *induced* by a vertex subset $S \subseteq V$ is denoted by $G[S] := (S, E \cap \binom{S}{2})$. A sequence $v_1, \ldots, v_k$ of vertices is a path in G if each pair (v_i, v_{i+1}) of consecutive vertices satisfies $\{v_i, v_{i+1}\} \in E$. If s is the first vertex and t is the last vertex in the path, then the path is said to be an s-t path. A graph is said to be connected if

for every pair (s, t) of distinct vertices there is an s-t path. Otherwise, the graph is said to be disconnected[1]. Other graph terminology (e.g., degeneracy, treewidth, hypergraph) will be defined later. For more information about graphs, we refer the reader to [40].

1.1.2 Combinatorial optimization

Most of the problems considered in this dissertation are combinatorial optimization problems. As noted by Schrijver [101],

> *Combinatorial optimization searches for an optimum object in a finite collection of objects. Typically, the collection has a concise representation (like a graph), while the number of objects is huge—more precisely, grows exponentially in the size of the representation (like all matchings or all Hamiltonian circuits). So scanning all objects one by one and selecting the best one is not an option. More efficient methods should be found.*

Sometimes it is not so clear as to what constitutes a combinatorial optimization problem. For example, consider (bounded) linear programming—the problem of optimizing a linear objective function over a feasible region that is bounded and defined by a set of linear inequalities. This seems to be a continuous optimization problem. However, it also fits into Schrijver's framework. Here the finite collection of objects is the set of extreme points of the feasible region, and we are tasked with finding an extreme point with optimal objective value. Is linear programming a combinatorial optimization problem, a continuous optimization problem, or both?

[1]By this definition of connectivity, the *trivial* graphs $(\emptyset, \emptyset)$ and $(\{v\}, \emptyset)$ are connected. However, it is sometimes convenient to define the trivial graphs as disconnected. As Diestel [40] notes, "[s]ometimes, . . . , trivial graphs can be useful; at other times they form silly counterexamples and become a nuisance. To avoid cluttering the text with non-triviality conditions, we shall mostly treat the trivial graphs. . . with generous disregard."

In the words of Lawler [71], "[p]erhaps the best way to convey the nature of combinatorial optimization problems is to give some specific examples." So, we mention some problems considered in this dissertation below.

In Chapter 2, we consider the Maximum-Weight Connected Subgraph (MWCS) and Node-Weighted Steiner Tree (NWST) problems.

Problem: Maximum-Weight Connected Subgraph (MWCS).
Input: a graph $G = (V, E)$, a weight $w(v)$ for each vertex $v \in V$.
Output: a maximum-weight subset S of vertices such that $G[S]$ is connected.

In the MWCS problem, the weight of a subset of vertices is the sum of its vertices' weights. In the NWST problem, the weight of a subgraph is the sum of its vertices' and edges' weights.

Problem: Node-Weighted Steiner Tree (NWST).
Input: a graph $G = (V, E)$, a set $D \subseteq V$ of terminal vertices, a nonnegative weight $w(i)$ for each vertex/edge $i \in E \cup V$.
Output: a minimum weight subgraph that connects the terminals.

In Chapter 3, we consider the Maximum Clique problem. A subset $S \subseteq V$ of vertices is said to be a clique in a graph $G = (V, E)$ if its induced subgraph $G[S]$ has all possible edges, i.e., $\binom{|S|}{2}$ edges.

Problem: Maximum Clique.
Input: a graph $G = (V, E)$.
Output: a largest clique of G.

In Chapter 4, we consider the Minimum Vertex Cover and Maximum Independent Set problems. A subset S of vertices is said to be a vertex cover for a graph $G = (V, E)$

if each edge $\{u, v\} \in E$ has an endpoint in S, i.e., $|S \cap \{u, v\}| \geq 1$. An independent set is a subset S of vertices such that the induced subgraph $G[S]$ has no edges.

Problem: Maximum Independent Set.

Input: a graph $G = (V, E)$.

Output: a largest independent set of G.

It is easy to see that a subset $S \subseteq V$ of vertices is an independent set of $G = (V, E)$ if and only if S is a clique in $\overline{G}$, where $\overline{G} := \left(V, \binom{V}{2} \setminus E\right)$ is the *complement* of G.

Problem: Minimum Vertex Cover.

Input: a graph $G = (V, E)$.

Output: a smallest vertex cover of G.

Note that a subset $S \subseteq V$ of vertices is a vertex cover for G if and only if $V \setminus S$ is an independent set of G. Hence, these two problems are equivalent with respect to optimization.

1.1.3 Integer programming

Combinatorial optimization problems are often formulated as integer programs. An integer program is an optimization problem in which the decision variables are required to take integer values. We will only discuss integer linear programs, where the objective function and constraints are linear.

$$\text{(integer program)} \qquad \sup_{x \in \mathbb{Z}_+^n} \left\{ c^T x \;\middle|\; Ax \leq b \right\}.$$

Here, n denotes the number of variables. Let m denote the number of rows of A. Denote by

$$P := \{x \in \mathbb{R}^n_+ \mid Ax \leq b\}.$$

This polyhedron P is a commonly used relaxation for the integer program's feasible region. (A polyhedron is the intersection of a finite number of halfspaces.) The actual feasible region for the integer program is

$$\mathcal{S} := \{x \in \mathbb{Z}^n_+ \mid Ax \leq b\}.$$

One important observation is that we can solve the integer program by solving

$$\sup_x \left\{ c^T x \;\middle|\; x \in P_I \right\},$$

where $P_I = \text{conv.hull}(\mathcal{S})$ is the convex hull of $\mathcal{S}$ (or the integer hull of P). If $\mathcal{S}$ is finite or if the problem data is rational, then this is actually a linear program [81]. The converse does not hold in general, since P_I may have an infinite number of facets. This is possible even when $n = 2$ and $m = 1$ [98]. A facet of a polyhedron P is an inclusion-wise maximal face of P that is distinct from P. The set F is a face of a polyhedron P if $F = \{x \in P \mid \pi x = \pi_0\}$ for some valid inequality $\pi x \leq \pi_0$ of P. An inequality $\pi x \leq \pi_0$ is said to be valid for a polyhedron P if every $x^* \in P$ satisfies $\pi x^* \leq \pi_0$.

For many combinatorial optimization problems, the variables represent yes/no decisions, e.g., should a particular vertex be included in the solution? This leads to the special class of integer programs called 0-1 programs. In this case, the supremum

is achieved (when feasible), so we can write max instead of sup.

$$\text{(0-1 program)} \qquad \max_{x \in \{0,1\}^n} \left\{ c^T x \;\middle|\; Ax \leq b \right\}.$$

In this case, the sets

$$P' = \{x \in [0,1]^n \mid Ax \leq b\}, \text{ and}$$
$$P'_I = \text{conv.hull}\{x \in \{0,1\}^n \mid Ax \leq b\}$$

are bounded polyhedra, a.k.a. polytopes, and the 0-1 program can be solved via the linear program

$$\max_x \left\{ c^T x \;\middle|\; x \in P'_I \right\}.$$

Of course, P'_I can have many more facets than P', so even though this is a linear program, it is not clear if it can be solved quickly.

In some cases, a polyhedron that has many facets can be represented in a higher-dimensional space by another polyhedron that has fewer facets. This is called an extension and is illustrated in Figure 1.2. More formally, a polyhedron $F = \{(x,y) \in \mathbb{R}^{n+t} \mid Cx + Dy \leq d\}$ is an extension for a polyhedron $P = \{x \in \mathbb{R}^n \mid Ax \leq b\}$ if $P = \text{proj}_x F$, where $\text{proj}_x F := \{x \mid \exists y : (x,y) \in F\}$. In this case, the inequalities $Cx + Dy \leq d$ provide an extended formulation for P. The size of an extension (resp. extended formulation) is the number of its facets (resp. inequalities). Often, the hope is to find a polynomial-size extended formulation for a problem whose representation in the original space of variables has exponential size. This is the case for the spanning tree polytope on n vertices, which has $\Theta(2^n)$ facets in the original space of variables [45], but admits an extended formulation of size $O(n^3)$ [78].

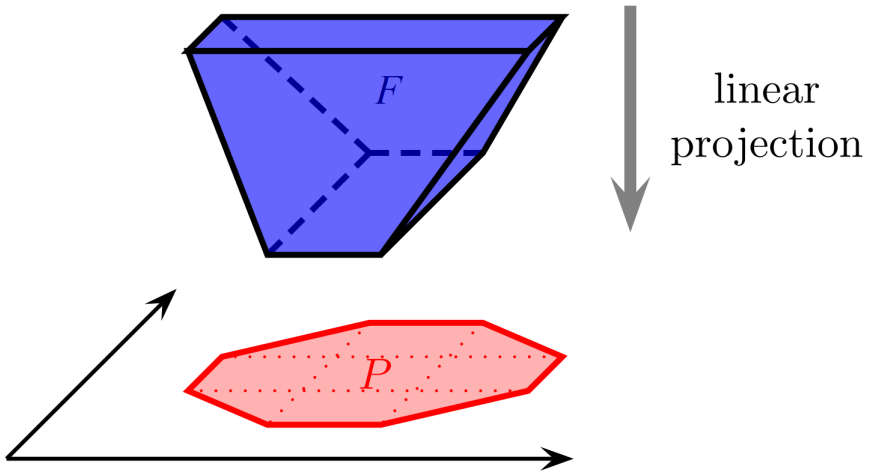

Figure 1.2: The extension F for P has 2 fewer facets. Original image by Thomas Rothvoss, modified and used with permission.

For more information on integer programming, consult (in order of increasing mathematical maturity) [110, 86, 100]. For more information about extended formulations, consult the surveys of [34, 66] or the text of [35].

1.1.4 Parameterized complexity

In computational complexity theory, the goal is to understand how efficiently a class of problems can be solved. This may include an analysis of how quick algorithms can be or how much space is necessary. In this dissertation, however, we are primarily concerned with runtime.

It is reasonable to expect that the runtime should depend on the problem input, with larger problems taking longer to solve. As such, we typically analyze the complexity of a problem as a function of the problem instance size. A natural question to ask is—How slow-growing must this function be for an algorithm to be practical? Is there a systematic way to categorize which problems are solvable in a reasonable amount of time on a computer? It is the Cobham-Edmonds thesis that this class of

reasonable problems coincides with those that can be solved in polynomial time [55]. This rule-of-thumb is imperfect (as Edmonds has noted [46]), but it has nevertheless proven to be "a good place to start" [46].

A large number of important problems have evaded this type of analysis. For example, no one has been able to solve the maximum clique problem in polynomial time, but no one has proven that this cannot be done. However, there is a consensus among researchers that no such algorithm exists. This is due to the fact that the maximum clique problem is NP-hard, and the widely-believed conjecture that P≠NP.

It is natural, then, to consider parameterized versions of the maximum clique problem, say the k-clique problem[2].

Problem: k-Clique.

Input: A simple graph G.

Parameter: A positive integer k.

Question: Does G have a clique of size k?

As far as the author knows, it is consistent with P≠NP that k-clique can be solved in time $O(2^k n)$. However, this is not suspected to be the case. Indeed, it is conjectured that this problem is not fixed-parameter tractable [42].

Definition 1 (fpt)**.** *A problem, parameterized by an integer k, is said to be fixed-parameter tractable (fpt) if it admits an algorithm running in time $f(k)n^{O(1)}$, where f is a computable function that does not depend on the input size n.*

Even worse, there are reasons [30] to think that k-clique cannot be solved in time $f(k)n^{o(k)}$. This follows either from the conjecture that not all SNP problems

[2]Often, researchers refer to a clique of k vertices as as a k-clique and the associated decision problem as k-Clique. We will typically refer to both as k-clique, and it will be clear what we mean based on the context. The same conventions will be used for k-coloring, k-vertex cover, k-independent set, etc.

(defined by [90]) can be solved in subexponential time [30], or that the exponential time hypothesis of Impagliazzo et al. [63] holds [76]. In contrast, the exhaustive search algorithm solves k-clique in time $O(f(k)n^k)$. More complicated algorithms based on matrix multiplication solve $(3k)$-clique in time $O(n^{\omega k})$, where $\omega < 2.373$ is the exponent of matrix multiplication [87]. While faster than the exhaustive search algorithm, this is still $n^{\Omega(k)}$.

While the minimum vertex cover problem and the maximum clique problem are equivalent, their parameterized versions seems to be much different. Indeed, k-vertex cover is fixed-parameter tractable, as there is a simple bounded search tree algorithm that runs in time $O(2^k n)$ [42]. Other algorithms improve this bound to $O(1.2738^k + kn)$ [32].

Definition 2 (ETH)**.** *The* exponential time hypothesis *(ETH) asserts that there is a constant $s > 0$ such that 3-CNF-SAT on n variables and m clauses cannot be solved in time $2^{sn}(n+m)^{O(1)}$.*

Thus, ETH states that 3-CNF-SAT requires time exponential in n (if the runtime must be polynomial in m). It has been shown [63] that if ETH is true, then 3-CNF-SAT also requires exponential time in the number m of clauses (if the runtime should be polynomial in n). There is also a strong version of ETH.

Definition 3 (SETH)**.** *The strong ETH (SETH) asserts that for every $\epsilon > 0$, there is a k such that k-CNF-SAT cannot be solved in time $(2-\epsilon)^n(n+m)^{O(1)}$.*

SETH would then imply that the exhaustive search algorithm is essentially best-possible for SAT (and, consequently, for solving 0-1 programs).

While the belief in ETH and SETH is not as strong as that of P≠NP, they have not been disproved over the past fifteen years. As the plausibility of these

conjectures has increased, others have used them to provide conditional lower bounds for other problems [93, 76, 75, 38]. Even if ETH or SETH is false, these conditional lower bounds can be useful, providing new ways to disprove ETH or SETH. Indeed, Williams [109] states:

> *The author can't help but confess his belief here that SETH is false. Many of his papers were conceived by finding an approach to refute SETH which ultimately failed, but was applicable to another problem instead (Max-2-SAT, ACC-SAT, All-Pairs Shortest Paths, etc.)*

Thus we are not here to argue that SETH is true. Rather, we will use it to compare our results to the state of current knowledge. Paraphrasing Garey and Johnson [54],

> *I can't find a faster algorithm, but neither can all these famous people.*

There is much background information about algorithm design, computational complexity, parameterized complexity, and fixed-parameter tractable algorithms that we simply cannot mention here. For more information, consult [36, 92, 91, 55, 42, 88, 50, 52, 43].

1.2 Summary of Contributions

In Chapter 2, we provide parameterized algorithms for the node-weighted Steiner tree (NWST) problem and for the maximum-weight connected subgraph (MWCS) problem. The NWST algorithm generalizes the well-known Dreyfus-Wagner algorithm and runs in time $O(n^3)$ when the number of terminals is bounded. We then show that the MWCS problem is polynomial-time solvable in graphs with no 3-vertex independent set via polyhedral arguments and the machinery of the ellipsoid method.

Next, two combinatorial algorithms for MWCS are described, which run in polynomial time for instances with few positive- or negative-weight vertices. Together, they imply that MWCS can be solved in time $O(1.5875^n)$, which is the first improvement in literature over the exhaustive search algorithm.

In Chapter 3, we provide parameterized algorithms for the maximum clique problem and show how they can be generalized to solve arbitrary 0-1 programs. One challenge is that the natural parameter–the size of the clique–seemingly leads to a dead end. This motivates the search for other parameters that are small on real-life instances. One such parameter is the *degeneracy* of the graph, which is a measure of the graph's sparsity. It turns out that the maximum clique problem is fixed-parameter tractable when parameterized by degeneracy. Then, we extend the ideas to solve arbitrary 0-1 programs. The parameter in this case depends on properties of an associated *conflict graph*. Conflict graphs are used to model pairwise dependencies between the 0-1 variables. Roughly speaking, the algorithms that we develop are quick when the conflict graph is dense.

In Chapter 4, we apply ideas from parameterized algorithms to the study of extended formulations. When constructing an extended formulation for an optimization problem, an important thing to keep in mind is its size. However, due to the computational intractability of many problems, one cannot always expect to find polynomial-size extended formulations. To deal with this, one approach is to introduce a parameter, and look for extended formulations whose size grows polynomially in the problem size, but exponentially (or worse) in the parameter. We illustrate this approach on the independent set and vertex cover problems and consider parameters such as the number of maximal independent sets (or minimal vertex covers), the treewidth of the graph, and the size of the vertex cover.

Finally, we conclude and offer ideas for future research in Chapter 5.

2. ALGORITHMS AND POLYHEDRA FOR CONNECTIVITY PROBLEMS

This chapter is based on work with Yiming Wang and Sergiy Butenko [27, 108]. We provide parameterized algorithms for the node-weighted Steiner tree (NWST) problem and for the maximum-weight connected subgraph (MWCS) problem. The NWST algorithm generalizes the well-known Dreyfus-Wagner algorithm and runs in time $O(n^3)$ when the number of terminals is bounded. We then show that the MWCS problem is polynomial-time solvable in graphs with no 3-vertex independent set via polyhedral arguments and the machinery of the ellipsoid method. Next, two combinatorial algorithms for MWCS are described, which run in polynomial time for instances with few positive- or negative-weight vertices. Together, they imply that MWCS can be solved in time $O(1.5875^n)$, which is the first improvement in literature over the exhaustive search algorithm.

The Steiner tree problem is well-studied in literature [60] and has applications in the design of networks. Although it is NP-hard [68], it is fixed-parameter tractable with respect to the number k of terminals, as demonstrated by the Dreyfus-Wagner algorithm [44], which runs in time $O(3^k n + 2^k n^2 + n^3)$.

Problem: Steiner Tree.
Input: a graph $G = (V, E)$, a set $D \subseteq V$ of terminal vertices, a nonnegative weight $w(e)$ for each edge $e \in E$.
Output: a minimum weight subset of edges that connects the terminals.

The Steiner tree problem has weights associated only with the edges of the network. In practice, however, there may be costs associated with the vertices as well. This leads to the so-called *node-weighted Steiner tree problem* (NWST) [102], which

has weights associated with both vertices and edges.

Problem: Node-Weighted Steiner Tree (NWST).

Input: a graph $G = (V, E)$, a set $D \subseteq V$ of terminal vertices, a nonnegative weight $w(i)$ for each vertex/edge $i \in E \cup V$.

Output: a minimum weight subgraph that connects the terminals.

While it is easy to transform an edge-weighted instance of the Steiner tree problem into an instance with weights only on nodes (by subdividing edges, i.e., 'placing' nodes on edges), we are not aware of any nice reductions in the other direction. It surprised us then, that we could find no attempts in literature to generalize the Dreyfus-Wagner algorithm for the node-weighted case. In Section 2.1, we fill this void by generalizing the Dreyfus-Wagner algorithm to handle both edge and vertex weights at no extra cost in runtime.

We also consider the related maximum-weight connected subgraph (MWCS) problem, which has applications in cluster detection in bioinformatics [41] and many other areas [2]. This problem is also NP-hard, even when restricted [106, 64] to planar graphs of maximum degree three with all weights either $+1$ or -1. The problem has no set of terminal vertices, but its difficulty arises due to the possibility for negative-weight vertices.

Problem: Maximum-Weight Connected Subgraph (MWCS).

Input: a graph $G = (V, E)$, a weight $w(v)$ for each vertex $v \in V$.

Output: a maximum-weight subset S of vertices s.t. $G[S]$ is connected.

We are aware of no previous algorithms for MWCS that achieve a nontrivial worst-case runtime. However, in Section 2.1 we show that MWCS can be solved in time $O(1.5875^n)$. This algorithm relies upon two different fixed-parameter tractable

subroutines. The first solves MWCS in time $O(2^q(m+n))$, where q denotes the number of negative-weight vertices. Surprisingly, we show that this is best-possible under the *strong exponential time hypothesis* of [63]. The second, which uses the NWST algorithm as a subroutine, runs in time $O(4^p n^3)$ for instances of MWCS with p positive-weight vertices.

We first discovered that MWCS is polytime solvable whenever the graph G has independence number $\alpha(G)$ at most two. When $\alpha(G) \leq 2$, we show that the *connected subgraph polytope* is characterized by the following inequalities [108].

$$x_a + x_b - \sum_{i \in C} x_i \leq 1, \ \forall a,b\text{-separator } C \subseteq V, \ \forall \text{ nonadjacent } a,b \in V \tag{2.1}$$

$$0 \leq x_i \leq 1, \ \forall i \in V. \tag{2.2}$$

While this formulation has exponentially many constraints, the separation problem for the separator inequalities (2.1) can be solved in polynomial time, so one can optimize a linear function over this feasible region in polynomial time via the ellipsoid method [56]. We also found a polynomial-size extended formulation for this separator-based LP relaxation (defined by constraints (2.1) and (2.2)), so one need not rely on the ellipsoid method to prove polynomiality [108].

It has been noted that many problems transition from being easy at 2 to hard at 3. This is the "mystical power of twoness" [73] exhibited by, for example, k-dimensional matching, k-colorability, and k-SAT. A natural question is—Does the same 'twoness' phenomenon occur with MWCS? If not, is there ever a sharp transition in the problem's complexity as a function of the graph's independence number $\alpha(G)$? We show that this is not the case, as MWCS is solvable in time $O(4^{\alpha(G)} n^3)$.

2.1 Node-Weighted Steiner Tree

Perhaps the most well-known algorithm to solve the Steiner tree problem is a dynamic programming algorithm due to Dreyfus and Wagner [44]. It runs in time $O(3^k n + 2^k n^2 + n^3)$, where k denotes the number of terminals. We show that this algorithm can be generalized to solve NWST in the same time.

To prove the correctness of the NWST algorithm, we will need some notation and an optimal substructure lemma. The notation that we use largely follows [44]. Consider a tree T connecting a set $S \subseteq V$ of terminals. Recall that $N_G(v)$ denotes the neighborhood of vertex v in graph G. For a vertex $x \in V(T)$ and one of its neighbors $y \in N_T(x)$ in T, let $S_y(x) \subseteq S$ be the set of terminal vertices reachable from x via a path in T that first crosses y. Further, define $S_Y(x) := \cup_{y \in Y} S_y(x)$ and $S_Y[x] := S_Y(x) \cup \{x\}$. Finally, let $T^Y(x)$ be the subtree of T that connects $S_Y[x]$. Note that Y is superscript for tree $T^Y(x)$, but is subscript for terminal subset $S_Y(x)$.

Lemma 1 (Optimal substructure)**.** *Let T be a minimum NWST connecting terminals $S \subseteq V$. Consider a vertex $x \in V(T)$ and a subset $Y \subseteq N_T(x)$ of its neighbors in T. Then the subtree $T^Y(x)$ is a minimum NWST connecting $S_Y[x]$.*

Proof. By the contrapositive. Let T_2 be the subgraph of T such that $T_2 \cup T^Y(x) = T$ and $T_2 \cap T^Y(x) = (\{x\}, \emptyset)$, Then,

$$\begin{aligned} & \sum_{v \in V(T)} w(v) + \sum_{e \in E(T)} w(e) \\ = & \sum_{v \in V(T^Y(x))} w(v) + \sum_{e \in E(T^Y(x))} w(e) + \sum_{v \in V(T_2)} w(v) + \sum_{e \in E(T_2)} w(e) - w(x). \end{aligned}$$

If $T^Y(x)$ is not a minimum NWST connecting $S_Y[x]$, then there is a connected

subgraph T_1 that connects $S_Y[x]$ and

$$\sum_{v \in V(T_1)} w(v) + \sum_{e \in E(T_1)} w(e) < \sum_{v \in V(T^Y(x))} w(v) + \sum_{e \in E(T^Y(x))} w(e).$$

Then, consider $T' = T_1 \cup T_2$, which is connected and has weight

$$\begin{aligned} & \sum_{v \in V(T')} w(v) + \sum_{e \in E(T')} w(e) \\ \leq & \sum_{v \in V(T_1)} w(v) + \sum_{e \in E(T_1)} w(e) + \sum_{v \in V(T_2)} w(v) + \sum_{e \in E(T_2)} w(e) - w(x) \\ < & \sum_{v \in V(T)} w(v) + \sum_{e \in E(T)} w(e). \end{aligned}$$

This shows that if $T^Y(x)$ is not a minimum NWST connecting $S_Y[x]$, then $w(T') < w(T)$, implying that T cannot be a minimum NWST. □

Theorem 1. *Algorithm 1 is correct and solves the NWST problem in time* $O(3^k n + 2^k n^2 + n^3)$*, where* k *denotes the number of terminals.*

Proof. The proof of the algorithm's correctness is based on the claim that, for each vertex v and each nonempty vertex subset D, the term $q(v, D)$ is the weight of a minimum NWST connecting $D \cup \{v\}$. Thus, the returned value $q(s, C)$ will be the weight of a minimum NWST connecting $S = C \cup \{s\}$. The claim about $q(\cdot, \cdot)$ is proven by induction on $|D|$.

If $|D| = 1$, then this is a shortest path problem and $q(v, D) = d_{vu}$ for some u, so the statement is true. Now suppose the statement is true for $|D| < i$. When $|D| = i$, consider a minimum NWST T_0 connecting v and D. Denote the weight of a subgraph H by $w(H)$. We consider three cases regarding the neighborhood of vertex v in T_0.

Algorithm 1 An algorithm for NWST

If terminals are not reachable from each other, return ∞;
Compute length d_{uv} of shortest path between all $u, v \in V$, where we define $d_{uu} = w(u)$. A path's length is the sum of weights of all of its edges and vertices (including endpoint vertices);
For every pair $u, v \in V$ of vertices, let $q(u, \{v\}) = d_{uv}$;
Select $s \in S$ and let $C = S \setminus \{s\}$;
for $i \leftarrow 2$ to $k - 1$ **do**
  for all $D \subseteq C$ with $|D| = i$ **do**
    for all $v \in V$ **do**
      $p(v, D) = \min_{A:0<|A|<|D|} \{q(v, A) + q(v, D \setminus A) - w(v)\}$;
    end for
    for all $v \in V$ **do**
      $q(v, D) = \min_{u \in V} \{d_{vu} + p(u, D) - w(u)\}$;
    end for
  end for
end for
return $q(s, C)$;

1. $|N_{T_0}(v)| \geq 2$, then there exist nonempty subsets $Q \subsetneq N_{T_0}(v)$ and $Q' := N_{T_0}(v) \setminus Q$. Since T_0 is a minimum NWST connecting D, we can assume, without loss of generality, that every leaf of T_0 belongs to D. This implies that[1] $|D_Q(v)| \geq 1$ and $|D_{Q'}(v)| \geq 1$. Also $D_Q(v) \cap D_{Q'}(v) = \emptyset$ and $D_Q(v) \cup D_{Q'}(v) = D$. By Lemma 1, $T^Q(v)$ is a minimum NWST connecting $D_Q[v]$, and $T^{Q'}(v)$ is a minimum NWST connecting $D_{Q'}[v]$. Since $|D_Q(v)| < |D|$ and $|D_{Q'}(v)| < |D|$

[1]For the definition of $D_Q(v)$, recall the notation $S_Y(x)$ introduced prior to Lemma 1.

(and by the induction assumption), the following holds:

$$\begin{aligned} w(T_0) &= w(T^Q(v)) + w(T^{Q'}(v)) - w(v) \\ &= q(v, D_Q(v)) + q(v, D_{Q'}(v)) - w(v) \\ &\geq p(v, D) \\ &= p(v, D) + d_{vv} - w(v) \\ &\geq q(v, D). \end{aligned}$$

Now we show that a contradiction arises if $w(T_0) > q(v, D)$, thus showing that $w(T_0) = q(v, D)$. By the algorithm, there exists a vertex u and a nonempty subset $A \subsetneq D$ of vertices such that

$$\begin{aligned} p(u, D) &= q(u, A) + q(u, D \setminus A) - w(u) \\ q(v, D) &= d_{vu} + p(u, D) - w(u). \end{aligned}$$

Let $H = T_1 \cup T_2 \cup P_{vu}$ where T_1 is a minimum NWST connecting u and A, T_2 is a minimum NWST connecting u and $D \setminus A$, and P_{vu} is a shortest path from v to u. This results in the contradiction that

$$\begin{aligned} q(v, D) &= d_{vu} + p(u, D) - w(u) \\ &= d_{vu} + q(u, A) + q(u, D \setminus A) - w(u) - w(u) \\ &= w(H) \\ &\geq w(T_0) \\ &> q(v, D). \end{aligned}$$

2. $|N_{T_0}(v)| = 1$ and the branch of T_0 touching v 'divides' before touching another terminal. Let the dividing vertex be u. Then no terminal is an interior vertex on the path P_{vu} from v to u in T_0. Let $T_1 = T_0 \setminus P_{vu} \cup \{u\}$, then $|N_{T_1}(u)| \geq 2$. By Lemma 1, T_1 is a minimum NWST connecting u and $D \setminus \{v\}$, and P_{vu} is the shortest path from v to u, so $w(T_1) = q(u, D \setminus \{v\}) = p(u, D \setminus \{v\})$ as it is in Case 1. Now,

$$\begin{aligned} w(T_0) &= d_{vu} + w(T_1) - w(u) \\ &= d_{vu} + q(u, D \setminus \{v\}) - w(u) \\ &= d_{vu} + p(u, D \setminus \{v\}) - w(u) \\ &\geq q(v, D \setminus \{v\}) \\ &= q(v, D). \end{aligned}$$

   Then, $w(T_0) = q(v, D)$ holds by the same analysis as in Case 1.

3. $|N_{T_0}(v)| = 1$ and the branch of T_0 touching v does not divide before touching another terminal. Denote by u the first terminal reachable from v in T_0. Again, let $T_1 = T_0 \setminus P_{vu} \cup \{u\}$. Then, by Lemma 1, T_1 is a minimum NWST connecting u and $D \setminus \{u, v\}$. By the induction assumption, $w(T_1) = q(u, D \setminus \{u, v\})$, so

$$\begin{aligned} w(T_0) &= d_{vu} + w(T_1) - w(u) \\ &= d_{vu} + q(u, D \setminus \{u, v\}) - w(u) \\ &= d_{vu} + (q(u, \{u\}) + q(u, D \setminus \{u, v\}) - w(u)) - w(u) \\ &\geq d_{vu} + p(u, D \setminus \{v\}) - w(u) \\ &\geq q(v, D \setminus \{v\}) \\ &= q(v, D). \end{aligned}$$

Then, $w(T_0) = q(v, D)$ holds by the same analysis as in Case 1.

So, $w(T_0) = q(v, D)$ holds in all cases. This shows $q(v, D)$ is the weight of a minimum NWST connecting v and D and the statement is true for $|D| = i$. So, the statement is true in general, and thus the returned value $q(s, C)$ is the weight of a minimum NWST connecting S.

The runtime of this algorithm is exactly the same as that of Dreyfus and Wagner, which is $O(3^k n + 2^k n^2 + n^3)$. □

2.2 Maximum-Weight Connected Subgraph

In this section, we describe algorithms for solving MWCS. First, we show that the problem is polytime solvable if the graph G has independence number $\alpha(G)$ at most two, i.e., there is no independent set of three vertices. This is shown via a polyhedral study of an appropriate linear programming (LP) relaxation and the machinery of the ellipsoid method. Then we describe two combinatorial algorithms for MWCS. The runtime of the first is parameterized by the number of negative-weight vertices, and the runtime of the second is parameterized by the number of positive-weight vertices. As a consequence of the second combinatorial algorithm and a preprocessing procedure, the MWCS problem is shown to be fixed-parameter tractable when parameterized by the graph's independence number. The two combinatorial algorithms imply a third algorithm that solves MWCS in time $O(4^{n/3}n^3) = O(1.5875^n)$.

2.2.1 Polyhedral description for case of no 3-vertex independent set

Here, we study the MWCS problem from a polyhedral perspective. The object of study is the *connected subgraph polytope* of a graph, which is the convex hull of subsets of vertices that induce a connected subgraph. This is essentially the feasible region for the MWCS problem. For convenience, consider zero-vertex and one-vertex graphs to be connected.

Definition 4. *The connected subgraph polytope of a graph $G = (V, E)$ is*

$$\mathscr{P}(G) := \text{conv.hull}\left\{x^S \in \{0,1\}^n \mid G[S] \text{ is connected}\right\},$$

where x^S denotes the characteristic vector of $S \subseteq V$.

A natural way to impose this type of induced connectivity constraint in an integer program is through the (exponentially-many) vertex separator inequalities, i.e., inequalities of the type

$$(a,b\text{-separator inequality}) \qquad x_a + x_b - \sum_{i \in C} x_i \leq 1,$$

where a and b are nonadjacent vertices and C is an a,b-separator. Recall that an a,b-separator C is a vertex subset (containing neither a nor b) such that nonadjacent vertices a and b are disconnected in $G[V \setminus C]$.

This leads to the following linear programming (LP) relaxation for $\mathscr{P}(G)$.

$$Q(G) := \{x \in [0,1]^n \mid x \text{ satisfies all separator inequalities}\}$$

Note that $Q(G)$ provides a *tractable* relaxation for $\mathscr{P}(G)$, as one can optimize a linear objective function over $Q(G)$ in polynomial time via the ellipsoid method [57]. This follows by the ability to separate over these inequalities in polytime. We will not discuss the details here.

A natural question to ask is—When is this LP relaxation tight? Theorem 2 below provides the answer, and the proof follows. Recall that the independence number $\alpha(G)$ of a graph G is the size of its largest independent set.

Theorem 2. *The equality $\mathscr{P}(G) = Q(G)$ holds if and only if $\alpha(G) \leq 2$.*

As a consequence of Theorem 2 and the ability to optimize over $Q(G)$ in polynomial time, we have the following corollary.

Corollary 1. *If $\alpha(G) \leq 2$, then MWCS is polynomial-time solvable.*

The result of Theorem 2 is interesting, in part, because there can be exponentially many inequalities defining $Q(G)$ even when $\alpha(G) = 2$. An example is shown in Figure 2.1, where the vertices within each rectangle form a clique. A minimal a, b-separator can be created by choosing, for each i, one vertex from $\{c_i, d_i\}$. The number of such separators is $2^{n/2-1}$.

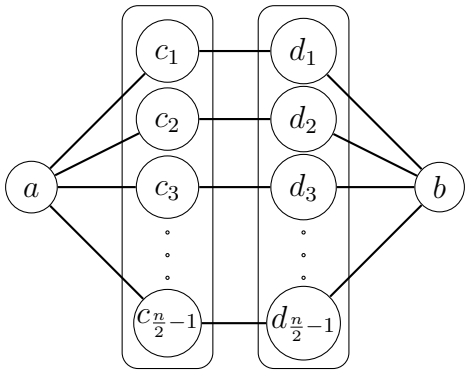

Figure 2.1: A graph G with $\alpha(G) = 2$, but many minimal a, b-separators. Vertices within a rectangle form a clique.

One direction of the proof of Theorem 2 is easier and is shown first.

Lemma 2. *If $\mathscr{P}(G) = Q(G)$, then $\alpha(G) \leq 2$.*

Proof. By the contrapositive. Suppose that G has an independent set S of three vertices. Then, it is easy to see that the inequality $\sum_{i \in S} x_i \leq 1$ induces a facet of $\mathscr{P}(G[S])$. Moreover, this inequality can be lifted to induce a facet of $\mathscr{P}(G)$. The resulting inequality has at least three positive coefficients, but the inequalities

defining $Q(G)$ have at most two. Since $Q(G)$ is full-dimensional, it has a unique half-space representation (up to scalar multiples). Then, since $\mathscr{P}(G)$ has a facet-defining inequality that is not facet-defining for $Q(G)$, they cannot be equal. □

The other direction of the proof is much more complicated and requires several lemmata.

Lemma 3. *Suppose that $\sum_{i\in V}\pi_i x_i \le \pi_0$ is valid for $\mathscr{P}(G)$. If vertices u and v are adjacent and $\pi_v \ge 0$, then the following inequality is also valid.*

$$(\pi_u+\pi_v)x_u + 0x_v + \sum_{i\in V\setminus\{u,v\}} \pi_i x_i \le \pi_0.$$

Proof. Suppose that $G[S]$ is connected, and consider the following two cases.

- If $u \in S$, then $S' = S \cup \{v\}$ is also connected, so

$$\begin{aligned}&(\pi_u+\pi_v)x_u^S + 0x_v^S + \sum_{i\in V\setminus\{u,v\}} \pi_i x_i^S \\ =&(\pi_u+\pi_v)x_u^{S'} + 0x_v^{S'} + \sum_{i\in V\setminus\{u,v\}} \pi_i x_i^{S'} = \sum_{i\in V} \pi_i x_i^{S'} \le \pi_0.\end{aligned}$$

- If $u \notin S$, then since $x_u^S = 0$ and $0x_v^S \le \pi_v x_v^S$, we have

$$(\pi_u+\pi_v)x_u^S + 0x_v^S + \sum_{i\in V\setminus\{u,v\}} \pi_i x_i^S \le \sum_{i\in V} \pi_i x_i^S \le \pi_0.$$

Thus, the inequality is valid in both cases, and is valid in general. □

Lemma 4 (folklore)**.** *Let $ax \le b$ and $cx \le d$ be valid inequalities for a* full-dimensional *polyhedron P such that (a,b) and (c,d) are not scalar multiples of each other. Then, the aggregated inequality $(a+c)x \le (b+d)$ cannot induce a facet of P.*

Lemma 5. *In a facet-defining inequality $\sum_{i\in V}\pi_i x_i \le \pi_0$ of $\mathscr{P}(G)$, no pair of adjacent vertices can have positive coefficients.*

Proof. Suppose that adjacent vertices u and v have positive coefficients. Then, by Lemma 3, the following inequalities are valid.

$$(\pi_u+\pi_v)x_u + 0x_v + \sum_{i\in V\setminus\{u,v\}} \pi_i x_i \le \pi_0;$$
$$0x_u + (\pi_u+\pi_v)x_v + \sum_{i\in V\setminus\{u,v\}} \pi_i x_i \le \pi_0.$$

These inequalities imply $\sum_{i\in V}\pi_i x_i \le \pi_0$. To wit, multiply the first inequality by $\beta := \pi_u/(\pi_u+\pi_v)$, multiply the second by $1-\beta$, and add these scaled inequalities together. Moreover, the three inequalities are distinct. So, by Lemma 4, $\sum_{i\in V}\pi_i x_i \le \pi_0$ cannot induce a facet. □

Lemma 6. *Suppose that $\sum_{i\in V}\pi_i x_i \le \pi_0$ induces a facet of $\mathscr{P}(G)$. If π_u and π_v are its only positive coefficients, then $\pi_u = \pi_v = \pi_0$.*

Proof. Since $G[\{u\}]$ and $G[\{v\}]$ are connected, this implies that $\pi_u \le \pi_0$ and $\pi_v \le \pi_0$. If $\pi_u+\pi_v \le \pi_0$, then any 0-1 solution $x^* \in \mathscr{P}(G)$ satisfying the inequality at equality must have $x^*_u = x^*_v = 1$, implying that the face of $\mathscr{P}(G)$ where $\sum_{i\in V}\pi_i x_i = \pi_0$ has dimension at most $n-2$, meaning that the inequality cannot induce a facet. Thus, we will assume that $\pi_u+\pi_v > \pi_0$.

We claim that $S := \{i \in V \mid \pi_i < 0\}$ is a u,v-separator. Suppose not, then there exists a path from u to v in $G[V\setminus S]$. Let P be the set vertices in the path. This implies that $\sum_{i\in V}\pi_i x^P_i = \pi_u+\pi_v > \pi_0$, which contradicts the validity of $\sum_{i\in V}\pi_i x_i \le \pi_0$.

For contradiction purposes, suppose that at least one of π_u and π_v is less than π_0. Without loss of generality, suppose that $\pi_u < \pi_0$. Now, let $S' \subseteq S$ be a minimal

u, v-separator, and define

$$\pi_{max} := \max\{\pi_i \mid i \in S'\}$$
$$\epsilon := \frac{1}{2}\min\{-\pi_{max}, \pi_0 - \pi_u\}.$$

Note that $\pi_{max} < 0$ and $\pi_0 - \pi_u > 0$, so $\epsilon > 0$. Also, $\pi_u + \epsilon < \pi_0$, and for every $i \in S'$, we have $\pi_i + \epsilon < 0$. Further, let

$$R = V \setminus (S' \cup \{u, v\}).$$

Then consider the following inequalities.

$$(\pi_u + \epsilon)x_u + \pi_v x_v + \sum_{i \in S'}(\pi_i - \epsilon)x_i + \sum_{i \in R}\pi_i x_i \leq \pi_0 \tag{2.3}$$
$$(\pi_u - \epsilon)x_u + \pi_v x_v + \sum_{i \in S'}(\pi_i + \epsilon)x_i + \sum_{i \in R}\pi_i x_i \leq \pi_0. \tag{2.4}$$

If these inequalities were valid, then they would imply $\sum_{i \in V}\pi_i x_i \leq \pi_0$, thus showing (by Lemma 4) that $\sum_{i \in V}\pi_i x_i \leq \pi_0$ cannot induce a facet, a contradiction. The rest of the proof is devoted to showing that inequalities (2.3) and (2.4) are indeed valid when $\pi_u < \pi_0$.

Consider $D \subseteq V$ such that $G[D]$ is connected. There are two cases. In the first case, $|D \cap \{u, v\}| \leq 1$. Then, since $\pi_i \leq 0$ for any $i \in R \subseteq V \setminus \{u, v\}$ and

$\pi_i - \epsilon < \pi_i + \epsilon < 0$ for any $i \in S'$,

$$\begin{aligned}
&(\pi_u + \epsilon)x_u^D + \pi_v x_v^D + \sum_{i \in S'} (\pi_i - \epsilon)x_i^D + \sum_{i \in R} \pi_i x_i^D \\
&\leq (\pi_u + \epsilon)x_u^D + \pi_v x_v^D \\
&\leq \max\{\pi_u + \epsilon, \pi_v\} \leq \pi_0.
\end{aligned}$$

The same logic shows that inequality (2.4) is valid when $|D \cap \{u, v\}| \leq 1$.

In the second case, $|D \cap \{u, v\}| = 2$. Since S' is a u, v-separator and both u and v belong to D, there exists $w \in D \cap S'$. Then, since $\pi_i \leq 0$ for any $i \in R \subseteq V \setminus \{u, v\}$, we have

$$\begin{aligned}
&(\pi_u + \epsilon)x_u^D + \pi_v x_v^D + \sum_{i \in S'} (\pi_i - \epsilon)x_i^D + \sum_{i \in R} \pi_i x_i^D \\
&\leq (\pi_u + \epsilon)x_u^D + \pi_v x_v^D + (\pi_w - \epsilon)x_w^D + \sum_{i \in S' \setminus \{w\}} \pi_i x_i^D + \sum_{i \in R} \pi_i x_i^D \\
&= \pi_u x_u^D + \pi_v x_v^D + \sum_{i \in V \setminus \{u,v\}} \pi_i x_i^D \\
&= \sum_{i \in V} \pi_i x_i^D \leq \pi_0.
\end{aligned}$$

Thus, inequality (2.3) is valid when $|D \cap \{u, v\}| = 2$.

Finally, we show that inequality (2.4) is valid when $|D \cap \{u, v\}| = 2$. Since u and v belong to D and $G[D]$ is connected, there is a path from u to v in $G[D]$. Moreover, at least one of these u-v paths crosses only one vertex, say w, from $S' \cap D$. This holds by minimality of S'. Let P be the set of vertices in this particular u-v path.

Then, since $\pi_i \leq 0$ for any $i \in R \subseteq V \setminus \{u, v\}$, and $\pi_i + \epsilon < 0$ for any $i \in S'$, we have

$$\begin{aligned}
&(\pi_u - \epsilon)x_u^D + \pi_v x_v^D + \sum_{i \in S'}(\pi_i + \epsilon)x_i^D + \sum_{i \in R} \pi_i x_i^D \\
&= (\pi_u - \epsilon)x_u^D + \pi_v x_v^D + (\pi_w + \epsilon)x_w^D + \sum_{i \in S' \setminus \{w\}} (\pi_i + \epsilon)x_i^D + \sum_{i \in R} \pi_i x_i^D \\
&\leq (\pi_u - \epsilon)x_u^D + \pi_v x_v^D + (\pi_w + \epsilon)x_w^D + \sum_{i \in R \cap P} \pi_i x_i^D \\
&= \pi_u x_u^P + \pi_v x_v^P + \pi_w x_w^P + \sum_{i \in R \cap P} \pi_i x_i^P \\
&= \sum_{i \in V} \pi_i x_i^P \leq \pi_0.
\end{aligned}$$

□

Lemma 7. *If facet-defining inequality $\sum_{i \in V} \pi_i x_i \leq \pi_0$ of $\mathscr{P}(G)$ has exactly two positive coefficients, then it is a separator inequality.*

Proof. Let the positive coefficients be π_a and π_b. By Lemma 6, $\pi_a = \pi_b = \pi_0$. Define

$$\begin{aligned}
C &= \{i \in V \mid \pi_i = -\pi_0\} \\
S &= \{i \in V \mid -\pi_0 < \pi_i < 0\} \\
R &= \{i \in V \mid \pi_i < -\pi_0\}.
\end{aligned}$$

We claim that $R = \emptyset$. If not, there is a vertex $v \in R$, and the following inequality is valid.

$$-\pi_0 x_v + \sum_{i \in V \setminus \{v\}} \pi_i x_i \leq \pi_0. \tag{2.5}$$

Indeed, suppose that $D \subseteq V$ induces a connected subgraph. If $v \in D$, then

$$-\pi_0 x_v^D + \sum_{i \in V \setminus \{v\}} \pi_i^D x_i \leq -\pi_0 + \pi_a + \pi_b = \pi_0;$$

and if $v \notin D$, then

$$-\pi_0 x_v^D + \sum_{i \in V \setminus \{v\}} \pi_i x_i^D = \sum_{i \in V} \pi_i x_i^D \leq \pi_0.$$

This shows that inequality (2.5) is valid. But, by Lemma 4, inequality (2.5) and the valid inequality $(\pi_v + \pi_0)x_v \leq 0$ show that $\sum_{i \in V} \pi_i x_i \leq \pi_0$ cannot induce a facet, a contradiction. Hence $R = \emptyset$.

Thus, we can write the facet-defining inequality as

$$\pi_0 x_a + \pi_0 x_b - \sum_{i \in C} \pi_0 x_i + \sum_{i \in S} \pi_i x_i \leq \pi_0. \tag{2.6}$$

Now see that $C \cup S$ must be an a, b-separator. If not, then there is a path P from a to b in $G[V \setminus (C \cup S)]$, yielding the contradiction that

$$2\pi_0 = \pi_a + \pi_b = \sum_{i \in V} \pi_i x_i^P \leq \pi_0.$$

If $S = \emptyset$, then inequality (2.6) is an a, b-separator inequality, as desired. So

suppose that $S \neq \emptyset$ and consider the following subsets of vertices.

$$A = \{v \in V \mid v \text{ and } a \text{ belong to the same component of } G[V \setminus (C \cup S)]\}$$
$$B = \{v \in V \mid v \text{ and } b \text{ belong to the same component of } G[V \setminus (C \cup S)]\}$$
$$S_A = \{s \in S \mid N_G(s) \cap A \neq \emptyset\}$$
$$S_B = \{s \in S \mid N_G(s) \cap B \neq \emptyset\}.$$

We argue that $S_A \cap S_B = \emptyset$. Otherwise, for any vertex $v \in S_A \cap S_B$, the set $D := A \cup B \cup \{v\}$ is connected, so

$$2\pi_0 + \pi_v = \sum_{i \in V} \pi_i x_i^D \leq \pi_0.$$

This implies that $\pi_v \leq -\pi_0$, which contradicts that $v \in S$. Thus, the three sets S_A, S_B, and $S \setminus (S_A \cup S_B)$ partition S.

We claim that $S_A \cup S_B \neq \emptyset$. For contradiction purposes, suppose that $S_A = S_B = \emptyset$. Then C is an a, b-separator, so $\pi_0 x_a + \pi_0 x_b - \sum_{i \in C} \pi_0 x_i \leq \pi_0$ is valid, and, for $i \in S$, the inequality $\pi_i x_i \leq 0$ is valid. Then, by Lemma 4, inequality (2.6) cannot induce a facet. Thus $S_A \cup S_B \neq \emptyset$.

Now, choose an $\epsilon > 0$ such that $\pi_i + \epsilon \leq 0$ for each $i \in S_A \cup S_B$. We will show that inequality (2.7) below is valid; the proof for inequality (2.8) is similar.

$$\sum_{i \in V \setminus (S_A \cup S_B)} \pi_i x_i + \sum_{i \in S_A} (\pi_i + \epsilon) x_i + \sum_{i \in S_B} (\pi_i - \epsilon) x_i \leq \pi_0 \tag{2.7}$$
$$\sum_{i \in V \setminus (S_A \cup S_B)} \pi_i x_i + \sum_{i \in S_A} (\pi_i - \epsilon) x_i + \sum_{i \in S_B} (\pi_i + \epsilon) x_i \leq \pi_0. \tag{2.8}$$

Suppose that $D \subseteq V$ induces a connected subgraph. If $a \notin D$ or $b \notin D$, then

inequality (2.7) obviously holds, so suppose $a, b \in D$. Now, if $D \cap C \neq \emptyset$, then

$$\sum_{i \in V \setminus (S_A \cup S_B)} \pi_i x_i^D + \sum_{i \in S_A} (\pi_i + \epsilon) x_i^D + \sum_{i \in S_B} (\pi_i - \epsilon) x_i^D$$
$$\leq \pi_0 x_a^D + \pi_0 x_b^D - \sum_{i \in C} \pi_0 x_i^D \leq \pi_0.$$

Now suppose $D \cap C = \emptyset$. Consider a shortest path from a to b in $G[D]$ measured in terms of the number of vertices used from $S \cap D$. Let P be the vertices along this path. Note that $|P \cap S_A| = |P \cap S_B| = 1$, so

$$\sum_{i \in V \setminus (S_A \cup S_B)} \pi_i x_i^D + \sum_{i \in S_A} (\pi_i + \epsilon) x_i^D + \sum_{i \in S_B} (\pi_i - \epsilon) x_i^D$$
$$\leq \sum_{i \in V \setminus (S_A \cup S_B)} \pi_i x_i^P + \sum_{i \in S_A} (\pi_i + \epsilon) x_i^P + \sum_{i \in S_B} (\pi_i - \epsilon) x_i^P$$
$$= \sum_{i \in V} \pi_i x_i^P \leq \pi_0.$$

So, in both cases, inequality (2.7) is valid.

Thus inequalities (2.7) and (2.8) are valid. But, by Lemma 4, this contradicts that inequality (2.6) induces a facet. So, $S = \emptyset$, and inequality (2.6) is (a scalar multiple of) an a, b-separator inequality. □

Lemma 8. *Consider a facet-defining inequality $\sum_{i \in V} \pi_i x_i \leq \pi_0$ of $\mathscr{P}(G)$. Then $\pi_0 \geq 0$. Further, the inequality is (a scalar multiple of) some nonnegativity bound $-x_j \leq 0$ if and only if $\pi_0 = 0$.*

Proof. As the empty set is assumed to induce a connected subgraph, $\pi_0 \geq 0$. The 'only if' direction is trivial.

Now, suppose that $\pi_0 = 0$. Then $\pi_i \leq 0$ for each vertex $i \in V$ (since the trivial graphs are connected). Further suppose that at least two coefficients are negative,

say π_u and π_v. Then $\sum_{i \in V} \pi_i x_i \leq 0$ is implied by the valid inequalities $\pi_u x_u \leq 0$ and $\sum_{i \in V \setminus \{u\}} \pi_i x_i \leq 0$. These two new inequalities are distinct, so Lemma 4 shows that $\sum_{i \in V} \pi_i x_i \leq 0$ cannot be facet-defining. □

Lemma 9. *If $\alpha(G) \leq 2$, then $\mathscr{P}(G) = Q(G)$.*

Proof. Consider an arbitrary facet-defining inequality $\sum_{i \in V} \pi_i x_i \leq \pi_0$ of $\mathscr{P}(G)$. Let $S = \{i \in V \mid \pi_i > 0\}$. Suppose that S contains at least three vertices, say $u, v, w \in S$. Then, by Lemma 5, $\{u, v, w\}$ is independent, contradicting $\alpha(G) \leq 2$. Thus $|S| \leq 2$. Consider the following three cases. In each case, we show that the inequality (or a scalar multiple thereof) is already in the description of $Q(G)$.

In the first case, suppose $|S| = 0$. Recall that $\pi_0 \geq 0$ by Lemma 8. Then, since no variable has a positive coefficient, π_0 cannot be positive, since otherwise no point in $\mathscr{P}(G)$ could satisfy the inequality at equality. Thus $\pi_0 = 0$. Then, by Lemma 8, the inequality is a (scalar multiple of a) nonnegativity bound.

In the second case, $|S| = 1$, and suppose $S = \{j\}$. Then, $\pi_0 \geq \pi_j > 0$, since $G[\{j\}]$ is connected. Further, $\pi_0 = \pi_j$, since otherwise no point in $\mathscr{P}(G)$ satisfies the inequality at equality. Now, the inequality $\pi_j x_j \leq \pi_0$ is valid, and $0x_j + \sum_{i \in V \setminus \{j\}} \pi_i x_i \leq 0$ is valid since $\pi_i \leq 0$ for every $i \in V \setminus \{j\}$. If $\pi_i = 0$ for every $i \in V \setminus \{j\}$, then $\sum_{i \in V} \pi_i x_i \leq \pi_0$ is a scalar multiple of $x_j \leq 1$, as desired. Otherwise, there is vertex $k \in V \setminus \{j\}$ with $\pi_k < 0$. Then the inequality $0x_j + \sum_{i \in V \setminus \{j\}} \pi_i x_i \leq 0$ discussed previously is not the $0x \leq 0$ inequality, and it, along with $\pi_j x_j \leq \pi_j$ imply $\sum_{i \in V} \pi_i x_i \leq \pi_0$, so by Lemma 4, the inequality $\sum_{i \in V} \pi_i x_i \leq \pi_0$ cannot induce a facet, a contradiction.

In the third and final case, $|S| = 2$. Then, by Lemma 7, the facet-defining inequality is a separator inequality.

Thus, in every case, the facet-defining inequality $\sum_{i \in V} \pi_i x_i \leq \pi_0$ of $\mathscr{P}(G)$ is already part of the description of $Q(G)$. Thus $Q(G) \subseteq \mathscr{P}(G)$. The reverse inclusion

is easy to see, since $Q(G)$ is a relaxation for $\mathscr{P}(G)$. □

2.2.2 Algorithm for case of few negative-weight vertices

We describe a simple algorithm for the maximum-weight connected subgraph problem. It is based on the insight that once one has determined which negative-weight vertices belong to the solution, it is easy to optimally extend the solution. The algorithm simply tests all possible subsets of the negative-weight vertices and then extends these partial solutions to full solutions. Accordingly, its runtime is exponential in the number of negative-weight vertices.

Interestingly, under the *strong exponential time hypothesis* (SETH) of [63], this algorithm cannot be improved. SETH is an unproven complexity assumption that is stronger than P≠NP. It asserts that for every $\epsilon > 0$, there is a k such that k-CNF-SAT cannot be solved in time $O((2-\epsilon)^n)$, where n denotes the number of variables. If true, this would imply that CNF-SAT cannot be solved in time $(2-\epsilon)^n(n+m)^{O(1)}$, where m denotes the number of clauses. While many doubt the verity of SETH, it is consistent with the fastest known algorithms for SAT, and we argue that designing a faster algorithm for the maximum-weight connected subgraph problem is as hard as finding a faster algorithm for SAT. The reader is referred to the survey of [76] for more information on lower bounds conditional on SETH and its weaker variant, the *exponential time hypothesis* (ETH).

The lower bound (conditional on SETH) for MWCS is based on the hardness of Hitting Set. Recall that Hitting Set is defined as follows.

Problem: Hitting Set.

Input: a family $\mathcal{F} \subseteq 2^{\mathcal{U}}$ of subsets of $\mathcal{U}$ and an integer t.

Question: does there exist $X \subseteq U$ such that $|X| \leq t$ and X has nonempty intersection with each $F \in \mathcal{F}$?

Algorithm 2 An algorithm for MWCS with few negative-weight vertices

```
Let S be the set of negative-weight vertices;
Find the connected components of G[V \ S];
Let z equal the weight of a maximum-weight component of G[V \ S];
for all D ⊆ S with D ≠ ∅ do
    Find the connected components of H := G[(V \ S) ∪ D];
    if all vertices from D belong to one component H' of H then
        z ← max{z, w(H')};
    end if
end for
return z;
```

Lemma 10 (Cygan et al. [38])**.** *For every $\epsilon > 0$, Hitting Set cannot be solved in time $(2-\epsilon)^{|\mathcal{U}|}(|\mathcal{U}| + |\mathcal{F}|)^{O(1)}$, unless SETH fails.*

For the following theorem, denote by q the number of negative-weight vertices, n the total number of vertices, and m the number of edges.

Theorem 3. *MWCS can be solved in time $O(2^q(m+n))$. For every $\epsilon > 0$, MWCS cannot be solved in time $(2-\epsilon)^q n^{O(1)}$, unless SETH fails.*

Proof. That the MWCS problem can be solved in such a time bound follows by Algorithm 2. The proof of the algorithm's runtime and correctness are straightforward.

For the lower bound, we construct a reduction from Hitting Set and rely on Lemma 10. Let the instance of Hitting Set be defined by a family $\mathcal{F} \subseteq 2^{\mathcal{U}}$ of subsets of $\mathcal{U}$ with target-size t. We construct an instance of MWCS on a graph $G = (V, E)$, where $V = \mathcal{U} \cup \mathcal{F}$. Construct the edge set E so that $G[\mathcal{U}]$ is complete, and for every pair of vertices $u \in \mathcal{U}$ and $F \in \mathcal{F}$ such that $u \in F$, add the edge $\{u, F\}$. Give each vertex from $\mathcal{U}$ weight -1 and each vertex from $\mathcal{F}$ weight $t+1$.

First we claim that G has a connected subgraph of weight $W := |\mathcal{F}|(t+1) - t$ if and only if the instance of Hitting Set is a yes-instance. If the instance of Hitting

Set has a solution $X \subseteq U$ with $|X| \leq t$, then $\mathcal{F} \cup X$ induces a connected subgraph of weight at least $|\mathcal{F}|(t+1) - t = W$. For the other direction, suppose that G has a connected subgraph $G[S']$ of weight at least W. Recognize that all vertices from $\mathcal{F}$ must belong to S' since otherwise the weight of S' is at most

$$(|\mathcal{F}| - 1)(t+1) = |\mathcal{F}|t + \mathcal{F} - t - 1 < |\mathcal{F}|t + \mathcal{F} - t = W. \tag{2.9}$$

Then, by construction of the edge set of G, the set $X' = S' \setminus \mathcal{F}$ provides a solution to the instance of Hitting Set. The inequality $|X'| \leq t$ holds by a weight argument.

Now we consider the runtime. Suppose there is an $\epsilon > 0$ such that MWCS can be solved in time $(2-\epsilon)^q n^{O(1)}$. Then, by the reduction described herein, Hitting Set can be solved in time $(2-\epsilon)^{|\mathcal{U}|}(|\mathcal{U}| + |\mathcal{F}|)^{O(1)}$, but Lemma 10 shows that this would disprove SETH. □

2.2.3 Algorithm for case of few positive-weight vertices

We now describe an algorithm for instances of MWCS that have few positive-weight vertices. This leads to an efficient algorithm for solving instances with small independence number. The most important insights in this section are described below.

1. **Preprocessing**. For a pair of adjacent, nonnegative-weight vertices, the edge between them can be contracted without altering the MWCS optimal objective.

2. **Use of NWST subroutine**. If one first decides which positive-weight vertices to select, then the problem of optimally connecting them with nonpositive-weight vertices is an instance of NWST.

For the following theorem, let p denote the number of positive-weight vertices, and let $T_{n,k}$ denote the time to solve an n-vertex instance of NWST with k terminals.

Algorithm 3 An algorithm for MWCS with few positive-weight vertices

Let U be the set of positive-weight vertices in G;
$z \leftarrow \max\{w(u) \mid u \in U\}$ or $z \leftarrow 0$ if $|U| = 0$;
for $i = 2, \ldots, |U|$ **do**
  for all $D \subseteq U$ with $|D| = i$ **do**
    Construct instance of NWST in $G[(V \setminus U) \cup D]$ with terminal set D and weight function w'', where $w''(u) = 0$ for vertices $u \in D$, and $w''(v) = -w(v)$ for vertices $v \in V \setminus U$.;
    Solve NWST instance yielding optimal NWST cost z_D;
    $z \leftarrow \max\{z, w(D) - z_D\}$;
  end for
end for
return z;

Theorem 4. *Algorithm 3 correctly solves MWCS in time* $O(n + \sum_{k=2}^{p} \binom{p}{k} T_{n,k})$.

Proof. The proofs of correctness and runtime are straightforward. □

Corollary 2. *MWCS can be solved in time* $O(4^p n^3)$*, where* p *denotes the number of positive-weight vertices.*

Proof. This follows directly from Theorem 4 using Algorithm 1 as the NWST subroutine, since $\sum_{k=2}^{p} \binom{p}{k} O(3^k n^3) = O(4^p n^3)$. □

Corollary 3. *Instances of MWCS with independence number* $\alpha(G)$ *can be solved in time* $O(4^{\alpha(G)} n^3)$.

Proof. Given an instance of MWCS, perform a preprocessing procedure that iteratively contracts edges between positive-weight vertices. We show that the number of remaining positive-weight vertices is at most $\alpha(G)$. Actually, this entire procedure can be done in time $O(n^2)$ as follows.

Formally, denote by S the set of positive-weight vertices, and let the components of $G[S]$ be $G[S_i],\ i = 1, \ldots, p$. Construct the smaller graph $G' = (V', E')$ as below,

where vertex $u_i \in U = \{u_1, \ldots, u_p\}$ represents component $G[S_i]$ and

$$V' = (V \setminus S) \cup U$$
$$E' = \{\{i,j\} \mid \{i,j\} \in E \text{ and } i,j \in V \setminus S\}$$
$$\cup \{\{v,u_i\} \mid \exists s \in S_i \text{ and } v \in V \setminus S \text{ such that } \{v,s\} \in E\}.$$

Then, create a new weight function $w' : V' \to \mathbb{R}$, where:

- for each $v \in V \setminus S$, set $w'(v) := w(v)$, and
- for each $j = 1, \ldots, p$, set $w'(u_j) := \sum_{s \in S_j} w(s)$.

Now we show that the number p of positive-weight vertices in the instance defined by graph G' and weight function w' has at most $\alpha(G)$ positive-weight vertices. Indeed, for each component $G[S_i]$ of $G[S]$, choose a vertex $s_i \in S_i$. Then, the set $\{s_1, \ldots, s_p\}$ is independent, so $p \leq \alpha(G)$.

The new instance of MWCS on graph G' with weight function w' can be solved in time $O(4^p|V'|^3)$ by Corollary 2. Then, since $p \leq \alpha(G)$, the original instance of MWCS can be solved in time $O(|V|^2 + 4^p|V'|^3) = O(4^{\alpha(G)}|V|^3)$. □

2.2.4 Combining the approaches

Now we can combine Algorithms 2 and 3 to beat the time $O(2^n(m+n))$ exhaustive-search algorithm. Letting p denote the number of positive-weight vertices, the algorithm solves MWCS as follows.

1. If $p \leq n/3$, run Algorithm 3;
2. Otherwise, run Algorithm 2.

Corollary 4. *MWCS can be solved in time* $O(1.5875^n)$.

3. ALGORITHMS FOR CLIQUE AND EXTENSIONS TO 0-1 PROGRAMS

This chapter is based on work with Jose Walteros, Sergiy Butenko, and Panos Pardalos [26, 25][1]. Here, we provide parameterized algorithms for the maximum clique problem and show how they can be generalized to solve arbitrary 0-1 programs. One challenge is that the natural parameter—the size of the clique—seemingly leads to a dead end. This motivates the search for other parameters that are small on real-life instances. One such parameter is the *degeneracy* of the graph, which is a measure of the graph's sparsity. It turns out that the maximum clique problem is fixed-parameter tractable when parameterized by degeneracy.

Then, we extend the ideas to solve arbitrary 0-1 programs. The parameter in this case depends on properties of an associated *conflict graph*. Conflict graphs are used to model pairwise-dependencies between the 0-1 variables. Roughly speaking, the algorithms that we develop are quick when the conflict graph is dense.

3.1 Algorithms for Maximum Clique

Recall that a clique $C \subseteq V$ in a graph $G = (V, E)$ is a subset of pairwise adjacent vertices. The problem of finding a largest clique in a given graph is one of the most well-known and well-studied NP-hard problems [68], and it has numerous applications in bioinformatics, computer vision, and coding theory [19].

Unfortunately, the maximum clique problem can be very challenging to solve exactly or even approximately, and most theoretical results are rather discouraging. For example, for any $\epsilon > 0$, the problem admits no polynomial-time $O(n^{1-\epsilon})$ approximation algorithm, unless P=NP [59, 113]. However, picking a single vertex provides

[1]Reprinted with permission from "Solving maximum clique in sparse graphs: an $O(nm + n2^{d/4})$ algorithm for d-degenerate graphs" by A. Buchanan, J.L. Walteros, S. Butenko, and P.M. Pardalos, 2014. *Optimization Letters*, 8(5):1611-1617, Copyright 2014 by Springer.

an n-approximation.

One approach is to *parameterize* the maximum clique problem by the number k of vertices in the clique. This k-Clique problem admits a trivial exhaustive-search algorithm that checks every subset of k vertices. Since there are $\binom{n}{k}$ such subsets, the runtime is $\Omega(n^k)$. More complicated algorithms based on matrix multiplication solve $3k$-clique in time $O(n^{\omega k})$, where $\omega < 2.373$ is the exponent of matrix multiplication [87]. So, it is possible to solve in time $o(n^k)$. However, no one has found an algorithm running in time $O(n^{o(k)})$, and there is reason to believe that no such algorithm exists [30].

Despite these depressing worst-case results, many real-life instances of the maximum clique problem are surprisingly easy to solve. For example, we show that many million-node instances can be solved in a few seconds, and a 17-million-node instance can be solved in 20 seconds [107]. A natural question to ask is—Why do these instances appear to be easier? Is there a rigorous explanation? This has been answered in the affirmative by Eppstein et al. [47], who showed how to list all maximal cliques in d-degenerate graphs in time $O(dn3^{d/3})$. This shows that the maximum clique problem is also fpt when parameterized by the graph's degeneracy.

Definition 5 (degeneracy [74])**.** *A graph is said to be d-degenerate if every (non-empty) subgraph has a vertex of degree at most d. The degeneracy of a graph is the smallest value of d such that it is d-degenerate.*

The degeneracy d of a graph is a measure of its sparsity, and it has been acknowledged that real-life graphs are sparse and have low degeneracy [33]. In fact, the 17-million-node instance that we were able to solve so quickly has degeneracy $d = 20$, which is many orders of magnitude smaller than n. Still, the implementation of Eppstein and Löffler [48] had trouble with this instance; it did not finish within

10 hours. Since the maximum clique problem is easier than the problem of listing all maximal cliques, it is an interesting question as to whether the dependence on d in the Eppstein et al. algorithm can be improved, so that larger values of d can be considered tractable. It may also be helpful to find algorithms that are based on a different parameter that is smaller than degeneracy. In the following sections, we proceed on both fronts, improving the dependence on d to $O^*(2^{d/4})$ and then introduce a different parameter called *community degeneracy* c (which is smaller than d) that leads to an algorithm running in time $O^*(2^{c/4})$.

Before proceeding with the algorithms, we remark that the parameters d and c are indeed much smaller than n for many real-life graphs. This is shown to be the case for several instances in Table 3.1.

Table 3.1: Parameters d and c on some real-life graphs from [5, 107].

Graph	n	m	ω	c	d
as-22july06	22,963	48,436	17	15	25
kron_g500-simple-logn16	65,536	2,456,071	136	283	432
citationCiteseer	268,495	1,156,647	13	11	15
ldoor	952,203	22,785,136	21	19	34
in-2004	1,382,908	13,591,473	489	487	488
cage15	5,154,859	47,022,346	6	4	25
rgg_n_2_24_s0	16,777,216	132,557,200	21	19	20
uk-2002	18,520,486	261,787,258	944	942	943

3.1.1 Algorithm based on degeneracy

In this section, we describe an algorithm for the maximum clique problem whose runtime is parameterized by the degeneracy d of the graph. Table 3.2 illustrates how our contribution fits within the literature.

The runtime of our algorithm relies upon the following lemma.

Table 3.2: Comparison of fastest known clique algorithms.

	arbitrary graphs	d-degenerate graphs
list all maximal cliques	$O(3^{n/3})$ by [23]	$O^*(3^{d/3})$ by [47]
find a maximum clique	$O(2^{n/4})$ by [95]	$O^*(2^{d/4})$ this section [26]

Lemma 11 (Lick and White [74])**.** *A graph is d-degenerate if and only if it admits an ordering of its vertices $(v_1, \ldots, v_n)$ such that each vertex v_i has at most d neighbors after it in the ordering, i.e., $|N(v_i) \cap \{v_i, \ldots, v_n\}| \leq d$.*

Such an ordering of the graph's vertices is called a degeneracy ordering. We will use the degeneracy ordering algorithm of Matula and Beck [80] as a subroutine in our algorithm.

Lemma 12 (Matula and Beck [80])**.** *A degeneracy ordering of a graph with n vertices and m edges can be found in time $O(n + m)$.*

The following lemma shows how we can decompose the maximum clique problem into smaller subproblems.

Lemma 13. *Let $(v_1, \ldots, v_n)$ be a vertex ordering of an n-vertex graph $G = (V, E)$. Denote by $\omega(G)$ the clique number of G, and let $S_i = N(v_i) \cap \{v_i, \ldots, v_n\}$. Then,*

$$\omega(G) = 1 + \max_{1 \leq i \leq n} \omega(G[S_i]). \tag{3.1}$$

Proof. First see that $\omega(G) \geq 1 + \omega(G[S_i])$ for any vertex $v_i \in V$; take a maximum clique in $G[S_i]$ and add v_i. Now we show the reverse inequality. Let S be a maximum clique in G and let $v_{i^*} \in S$ be its earliest vertex in the vertex-ordering. Then $S \subseteq S_{i^*} \cup \{v_{i^*}\}$ and $\omega(G[S_{i^*}]) \geq \omega(G) - 1$. □

If we use a degeneracy ordering, each of the $G[S_i]$ subproblems in Lemma 13 has at most d vertices. This is the main insight to our approach.

Let T_d denote the time to solve the maximum clique problem in an arbitrary d-vertex graph. Note that $T_d = O(2^{d/4})$ by the well-cited but unpublished paper [95] or $T_d = O(1.2114^d)$ by the peer-reviewed [20]. Either of these algorithms, or any other clique algorithm, can be used as MaxCliqueSubroutine$(\cdot)$.

Algorithm 4 A maximum clique algorithm parameterized by degeneracy.

Data: A graph $G = (V, E)$

Result: The clique number $\omega(G)$

compute a degeneracy ordering $(v_1, \ldots, v_n)$ of G

for $i = 1, \ldots, n$ **do**

  $S_i \leftarrow N(v_i) \cap \{v_i, \ldots, v_n\}$

  $\omega(G[S_i]) \leftarrow$ MaxCliqueSubroutine$(G[S_i])$

end

return $\omega(G) = 1 + \max_{1 \leq i \leq n} \omega(G[S_i])$

Theorem 5. *Algorithm 4 solves the maximum clique problem in d-degenerate graphs in time $O^*(T_d) = O^*(2^{d/4})$.*

Proof. The algorithm's correctness follows by Lemma 13, so we only need to consider the runtime. The degeneracy ordering $(v_1, \ldots, v_n)$ can be found in time $O(m + n)$ time by Lemma 12. By the degeneracy ordering, each of the $G[S_i]$ subgraphs has at most d vertices, hence the maximum clique subroutine takes time $O(T_d) = O(2^{d/4})$ by Robson [95]. The number of such subproblems and the time needed to set them up is polynomial, and hence is ignored in the $O^*(\cdot)$ notation. □

The most time-consuming step of Algorithm 4, namely, solving the n subproblems of the for-loop, can be done in parallel. This may be helpful for very large graphs.

3.1.2 Algorithm based on community degeneracy

We now move on to the second maximum clique algorithm, which is parameterized by the community degeneracy c of the graph. In one sense, this improves upon the algorithm from the previous section, since its runtime is exponential in c instead of d and since the inequality $c < d$ holds for any graph. However, this comes at the cost of a larger polynomial factor, since we solve m subproblems instead of n.

Definition 6 (community degeneracy)**.** *A graph is said to be c-community-degenerate if every (non-edgeless) subgraph G' has an edge $\{u, v\}$ with $|N_{G'}(u) \cap N_{G'}(v)| \leq c$. The community degeneracy of a graph is the smallest value of c such that it is c-community-degenerate.*

Just like degeneracy, the community degeneracy of a graph can equivalently be defined based on an ordering.

Lemma 14. *A graph $G = (V, E)$ is c-community degenerate if and only if it admits an ordering of its edges $(e_1, \ldots, e_m)$ such that each edge $e_i = \{u_i, v_i\}$ has $|N_{G[E_i]}(u_i) \cap N_{G[E_i]}(v_i)| \leq c$, where $G[E_i]$ is the* edge-induced *subgraph of $E_i = \{e_i, \ldots, e_m\}$.*

Proof. If G is c-community degenerate, then it contains an edge $e_1 = \{u_1, v_1\}$ satisfying $|N_G(u_1) \cap N_G(v_1)| \leq c$. Place edge e_1 first in the ordering. Then, in the graph $G_2 := G - e_1$, there exists an edge $e_2 = \{u_2, v_2\}$ satisfying $|N_{G_2}(u_2) \cap N_{G_2}(v_2)| \leq c$. Place edge e_2 second in the ordering. Clearly, this process can be repeated until all edges of G have been placed into the ordering.

Now suppose G admits such an ordering $(e_1, \ldots, e_m)$ of its edges. Consider an arbitrary subgraph $G' = (V', E')$ of G. Consider the earliest edge $e_j = \{u_j, v_j\}$ in the ordering that belongs to E' and let G_j denote the edge-induced subgraph of E_j.

Then,

$$|N_{G_j}(u_j) \cap N_{G_j}(v_j)| \leq |N_{G'}(u_j) \cap N_{G'}(v_j)| \leq c,$$

where the first inequality holds since G_j is a subgraph of G' and the second inequality holds by the edge ordering. Thus G is c-community degenerate. □

Such an ordering of the graph's edges is called a community degeneracy ordering. We now give an efficient algorithm for computing community degeneracy, which follows by the proof of Lemma 14.

Algorithm 5 An algorithm for finding a community degeneracy ordering.

Data: A graph $G = (V, E)$

Result: A community degeneracy ordering $(e_1, \ldots, e_m)$ of $E(G)$

$H \leftarrow G$

for $i = 1, \ldots, m$ **do**

  find an edge $e = \{u, v\}$ in H that minimizes $|N_H(u) \cap N_H(v)|$

  $e_i \leftarrow e$

  $H \leftarrow H - e$

end

return $(e_1, \ldots, e_m)$

Lemma 15. *Algorithm 5 finds a community degeneracy ordering and can be implemented to run in time $O(nm)$.*

Proof. The correctness is clear, so we are left with devising data structures that achieve $O(nm)$ time. The idea is to mimic the data structures used in the degeneracy

algorithm [80], but instead of using an adjacency list of the neighbors for each vertex, we have an intersection list for each edge $e = \{u, v\}$ that lists the vertices from $N(u) \cap N(v)$. Clearly, the intersection list of an edge contains at most $n-2$ vertices, and since $N(u) \cap N(v)$ can be obtained in $O(n)$ time, generating all such lists takes $O(nm)$ time.

Let $D_H(e) = |N_H(u) \cap N_H(v)|$. To execute the steps in the for-loop, instead of updating the intersection lists at each iteration, it is possible to keep track of the edges that have not been removed using a bucket structure. This structure is comprised of $n-1$ buckets (namely, $\{0, 1, \ldots, n-2\}$) and an array of headers pointing to one of the elements in each bucket. The buckets are stored as linked lists and are initialized while creating the intersection lists. At each iteration of the for-loop, if edge e is still in H, it is stored in bucket $D_H(e)$. Identifying the edge to be removed can be done in $O(n)$ time by searching for the first nonempty bucket. Furthermore, whenever an edge $e = \{u, v\}$ is removed from H, the algorithm scans the intersection list of e checking if edges $e' = \{u, w\}$ and $e'' = \{v, w\}$ remain in H, for all w in the list. If e' or e'' were not removed before, they are relocated to buckets $D_H(e') - 1$ and $D_H(e'') - 1$, respectively. Scanning the intersection list takes $O(n)$ time, and any edge relocation in a linked list takes constant time. Thus, since the algorithm removes all m edges and every iteration takes $O(n)$ time, the algorithm runs in $O(nm)$ time. □

Lemma 16. *Let $(e_1, \ldots, e_m)$ be any edge ordering of an m-edge graph $G = (V, E)$ with $m \geq 1$. Denote by $\omega(G)$ the size of a maximum clique in a graph G. Then,*

$$\omega(G) = 2 + \max_{1 \leq i \leq m} \omega(G_i), \tag{3.2}$$

where $e_i = \{u_i, v_i\}$, $E_i = \{e_i, \ldots, e_m\}$, $S_i = N_{G[E_i]}(u_i) \cap N_{G[E_i]}(v_i)$, and $G_i = \left(S_i, E_i \cap \binom{S_i}{2}\right)$.

If we use a community degeneracy ordering, each of the G_i subproblems in Lemma 16 has at most c vertices. This is the main insight to our approach.

Proof. First see that $\omega(G) \geq 2 + \omega(G_i)$ for any edge $e_i = \{u_i, v_i\} \in E$; take a maximum clique in G_i and add vertices u_i and v_i. Now we show the reverse inequality. Let S be a maximum clique in G and let $e_{i^*} = \{u_{i^*}, v_{i^*}\}$ be the earliest edge of $G[S]$ in the edge ordering. Then every vertex in $S \setminus \{u_{i^*}, v_{i^*}\}$ belongs to G_{i^*} and every pair of vertices in $S \setminus \{u_{i^*}, v_{i^*}\}$ is adjacent in G_{i^*}, so $\omega(G_{i^*}) \geq \omega(G) - 2$. □

Algorithm 6 A maximum clique algorithm parameterized by community degeneracy.

Data: A graph $G = (V, E)$

Result: The clique number $\omega(G)$

compute a community degeneracy ordering $(e_1, \ldots, e_m)$ of $E(G)$

for $i = 1, \ldots, m$ **do**

$\{u_i, v_i\} \leftarrow e_i \;\; E_i \leftarrow \{e_i, \ldots, e_m\}$

$S_i \leftarrow N_{G[E_i]}(u_i) \cap N_{G[E_i]}(v_i)$

$G_i \leftarrow \left(S_i, E_i \cap \binom{S_i}{2}\right)$

$\omega(G_i) \leftarrow$ MaxCliqueSubroutine(G_i)

end

return $\omega(G) = 2 + \max_{1 \leq i \leq m} \omega(G_i)$

Theorem 6. *Algorithm 6 solves the maximum clique problem in c-community degenerate graphs in time $O^*(T_c) = O^*(2^{c/4})$.*

Proof. The community degeneracy ordering $(e_1, \ldots, e_m)$ can be found in polynomial time by Lemma 15. There are polynomially many iterations of the for-loop, and in

each we compute the maximum clique of a graph on at most c vertices. This proves the runtime. Correctness of the algorithm follows by Lemma 16. □

Again, the iterations of the for-loop can be run in parallel.

3.1.3 Discussion

In this section, we discuss the relationship between the parameters c, d, and n–and hence the runtime of our algorithms.

We have mentioned that $c < d$, but have not proven it yet. However, this is not hard to see, as any d-degenerate-ordering immediately gives a $(d-1)$-community-degenerate ordering: replace the vertex v_1 in the vertex-ordering by the edges incident to v_1, replace the vertex v_2 by the edges incident to v_2 but not to v_1, etc. In general, the community degeneracy c is sandwiched between $\min_{\{u,v\}\in E} |N(u)\cap N(v)|$ and $\max_{\{u,v\}\in E} |N(u) \cap N(v)|$, whereas the degeneracy is sandwiched between the minimum and maximum degrees of the graph.

While $c < d$, it is possible for them to be very close to each other. Indeed, the n-vertex complete graph K_n has $d = n-1$ and $c = n-2$. However, there are classes of graphs for which $c \ll d \ll n$. For example, the p-dimensional hypercube graph Q_p is triangle-free and hence has $c = 0 \ll d = p \ll n = 2^p$.

However, we are not so interested in these contrived classes of graphs. What we really want to know is whether the parameters c and d are small compared to n on real-life instances. This was demonstrated empirically in Table 3.1, but we would also like some analytical evidence. It has been noted that the degrees of real-life graphs often follow a power-law distribution [33], which leads to the notion of power-law graphs. A graph is said to be power-law if the number of vertices with degree q is proportional to $q^{-\alpha}$, where $\alpha \in (1,3)$ is a constant. There are numerous theoretical models of power-law graphs, including the Bianconi-Marsili and Barabási-

Albert models. It has been shown [10] that, in Bianconi-Marsili power-law random graph model, $d = O(n^{1/(2\alpha)})$ whenever $1 < \alpha \leq 2$ and $d = O(n^{(3-\alpha)/4})$ whenever $2 < \alpha < 3$. In this case, our algorithm that is parameterized by degeneracy runs in $2^{O(\sqrt{n})}$ time with high probability ($\alpha > 1$). The time bound improves as α increases, running in time $2^{O(n^{1/4})}$ for $\alpha > 2$. We also note that the Barabási-Albert model creates graphs with bounded degeneracy [47], in which case our algorithms run in polynomial time.

In both of these power-law models, our algorithms run in time $2^{o(n)}$, whereas it is believed that no such algorithm exists for arbitrary graphs [30]. This provides theoretical support for the claim that the maximum clique problem is easier in real-life graphs than it is in arbitrary graphs, just as empirical observations suggest [107].

3.2 Algorithms for 0-1 Programs

In this section, we consider the canonical 0-1 programming problem:

$$\begin{aligned} &\text{maximize } c^T x \\ &\text{subject to } Ax \leq b \\ &x \in \{0,1\}^n. \end{aligned}$$

Here, n denotes the number of 0-1 variables, and we will let m denote the number of rows of A, i.e., the number of constraints.

Many combinatorial optimization problems can be formulated as a 0-1 programs. This includes the maximum clique problem, the minimum dominating set problem, and the maximum satisfiability problem. Consequently, the worst-case complexity of optimizing or approximating 0-1 programs is just as dismal. Indeed, the problem of determining whether a 0-1 program is even *feasible* cannot be solved in time

$O((2-\epsilon)^n \text{poly}(m))$ for any constant $\epsilon > 0$, unless the Strong Exponential Time Hypothesis (SETH) is false [61, 29]. However, a 0-1 program can trivially be solved in time $O^*(2^n)$ by exhaustive search, but it is doubtful that anyone would want to actually do this in practice.

This suggests that we should look for parameters besides n. A natural parameter is the number k of nonzeros in the solution vecor x. This leads to the problem

$$\max_{x \in \{0,1\}^n} \left\{ c^T x \;\middle|\; Ax \leq b \text{ and } \sum_{i=1}^{n} x_i \leq k \right\}.$$

However, even this problem cannot be solved in time $O(n^{o(k)})$ under reasonable complexity assumptions since we can formulate the k-clique problem in this way. Even worse, the k-dominating set problem cannot be solved in time $O(n^{k-\epsilon})$ for any constant $\epsilon > 0$, unless SETH fails [93]. This suggests that these types of 0-1 program cannot be solved in time $O(n^{k-1-\epsilon}m)$.

The approach that we take here uses ideas from our clique algorithms. In the degeneracy-based clique algorithm, we branched on the decision to include a particular vertex. In the branch where we choose to include the vertex, the resulting subproblem has at most d vertices by the degeneracy ordering. To solve a 0-1 program in a similar fashion, there should always be an unfixed variable such that when we fix it, one of its subproblems has at most d unfixed variables. In the clique algorithms, the graph's topology determined how the vertices should be fixed; if a vertex is included in the solution, then its nonneighbors are fixed out of the solution. To solve a 0-1 program, we seek a similar way to represent these logical implications, e.g., the implication $x_i = 1 \implies x_j = 0$.

A common approach in computer science is to use an *implication graph*. This is a directed graph in which there are two vertices x_i and $\overline{x}_i$ for each variable x_i,

and the directed edges denote logical implications between pairs of variables. For example, there is a directed edge $(x_i, \overline{x}_j)$ to represent $x_i = 1 \implies x_j = 0$. By the contrapositive, there will also be a directed edge $(x_j, \overline{x}_i)$. Using this representation, one can solve 2SAT in linear time [3]. In this case, the 2SAT instance is infeasible if and only if there is a variable x_i and its negation $\overline{x}_i$ that belong to the same strongly connected component of the implication graph [3].

While implication graphs (and extensions thereof) are sometimes used in the operations research literature [1], it is perhaps more common to see *conflict graphs* [4]. The vertex set in this case is the same as in the implication graph; however, the edges are undirected and represent conflicts instead of implications. For example, an edge $\{x_i, \overline{x}_j\}$ means that we cannot simultaneously set $x_i = 1$ and $x_j = 0$. One important observation is that any feasible solution to our 0-1 program (and hence to the conflict graph) corresponds to a subset of n vertices that is independent in the conflict graph. This allows us to generate valid inequalities for the 0-1 program based on known polyhedral results for the independent set polytope. For example, the so-called *clique inequalities* [89] can be used to tighten the linear programming relaxation of the 0-1 program [4]. In this case, for a clique $C \subseteq V(G)$ of the conflict graph G, the corresponding clique inequality is

$$\sum_{i:x_i \in C} x_i + \sum_{i:\overline{x}_i \in C} (1 - x_i) \leq 1.$$

So, the use of conflict graphs is not new in solving 0-1 programs. However, we know of no previous research in which a conflict graph is used to develop an algorithm or branching strategy with provable worst-case runtimes, which is our task.

It is often easier to work with the complement of the conflict graph. This graph, which we will call a *compatibility* graph, will be sparse whenever the conflict graph

is dense. An optimal solution to the 0-1 program corresponds to a clique in the compatibility graph. The converse is obviously not true.

In the following, we will consider the complexity of safely generating conflict edges. Then, we extend the concepts of degeneracy and community degeneracy to compatibility graphs. This results in compatibility degeneracy and bicompatibility degeneracy. Informally, compatibility degeneracy d means that every subproblem of the 0-1 program is clearly infeasible (via a 2SAT algorithm) or has an unfixed variable x_i and $t \in \{0, 1\}$ such that when it is fixed to $x_i = t$, at most d other variables in the $x_i = t$ branch remain undetermined. Bicompatibility degeneracy d' is defined analogously for pairs of unfixed variables. We show that these parameters d and d' can be efficiently computed. They lead to algorithms that list all feasible solutions to 0-1 programs in time $O^*(2^d)$ or $O^*(2^{d'})$. This shows that the problem of solving 0-1 programs is fixed-parameter tractable when parameterized by d or d'. Moreover, these algorithm are optimal in the sense that there are cases where the number of solutions is $\Omega(2^d)$ or $\Omega(2^{d'})$. We also include calculations of the parameter d for instances from literature.

3.2.1 The complexity of generating conflict edges

We will assume that we are given the compatibility graph as part of the problem input; we do not construct it in the course of our algorithms. In practice, conflict graphs are created using simple preprocessing rules and probing [99, 4, 1]. While these procedures can quickly find many conflict edges for real-life instances (typically for problems with packing constraints $\sum_{i \in S} x_i \leq 1$), we show that this appears to be extremely difficult in the worst-case.

Problem: Conflict Edge Creation.

Input: matrix A; vector b; variables x_i and x_j; values $s, t \in \{0, 1\}$.

Question: Is it true that no $x \in \{0,1\}^n$ simultaneously satisfies $x_i = s,\ x_j = t$, and $Ax \leq b$?

Notice that the answer to the question above is 'yes' if and only if we can safely generate the corresponding conflict edge.

Proposition 1. *For any $\epsilon > 0$, Conflict Edge Creation cannot be solved in time $O\left((2-\epsilon)^n m^{O(1)}\right)$, unless SETH fails.*

Proof. Suppose there is an $\epsilon > 0$ such that we can solve Conflict Edge Creation in time $O\left((2-\epsilon)^n m^{O(1)}\right)$. Consider the following instance of Conflict Edge Creation. Formulate CNF-SAT as a 0-1 program in the usual way and add two additional variables x_{n+1} and x_{n+2} that appear in no constraints. Let $i = n+1,\ j = n+2,\ s = t = 0$. Clearly, the CNF formula is satisfiable if and only if this Conflict Edge Creation instance is a 'no' instance. But this means we can solve CNF-SAT in time $O\left((2-\epsilon)^n m^{O(1)}\right)$, which contradicts SETH. □

It may be the case that we are willing to accept *any* conflict edge. This leads to the following problem, where the variables and values are not specified in the input as they were for Conflict Edge Creation.

Problem: Unspecified Conflict Edge Creation.

Input: matrix A; vector b.

Output: (If any exist) variables x_i and x_j, $i \neq j$, and values $s, t \in \{0,1\}$ such that no solution $x \in \{0,1\}^n$ to $Ax \leq b$ has $(x_i, x_j) = (s,t)$.

Consider the following algorithm to determine feasibility of 0-1 programs. First attempt to find a conflict edge, in time $f(n,m)$. If none exist, then the 0-1 program is feasible. Otherwise, there is a conflict edge that shows that no solution satisfies $x_i = s$ and $x_j = t$ for some i, j, s, t with $i \neq j$. In this case, create two subproblems:

in the first fix $x_i = s$ and $x_j = 1 - t$; in the second fix $x_i = 1 - s$. This is a valid disjunction by the conflict edge. Solve the two subproblems recursively using the same procedure.

Lemma 17. *The number of subproblems in the algorithm above is $O(\phi^n)$, where $\phi = \frac{1+\sqrt{5}}{2} = 1.6180\ldots$ is the golden ratio.*

Proof. The number $g(n)$ of subproblems satisfies the recurrence $g(n) \leq g(n-1) + g(n-2)$. By standard techniques (cf. Theorem 2.1 of [52]), the solution to this recurrence is $g(n) = O(\phi^n)$. □

Proposition 2. *For any $\epsilon > 0$, Unspecified Conflict Edge Creation cannot be solved in time $O^*\left(\left(\frac{2}{\phi} - \epsilon\right)^n\right)$, unless SETH fails.*

Proof. Suppose Unspecified Conflict Edge Creation can be solved in time $O^*\left(\left(\frac{2}{\phi} - \epsilon\right)^n\right)$ for some constant $\epsilon > 0$. Then, consider the algorithm proposed for determining feasibility of 0-1 programs that is described above. There are $O(\phi^n)$ subproblems by Lemma 17. And, at each node in the computation tree, we solve an instance Unspecified Conflict Edge Creation in time $O^*\left(\left(\frac{2}{\phi} - \epsilon\right)^n\right)$. Thus, the total time to determine feasibility a 0-1 program is

$$O^*\left(\left(\frac{2}{\phi} - \epsilon\right)^n \phi^n\right) = O^*\left((2 - \epsilon\phi)^n\right) = O^*\left((2 - \epsilon')^n\right),$$

where $\epsilon' = \epsilon\phi > 0$. But, this would show that SETH is false. □

The exhaustive search algorithm solves Unspecified Conflict Edge Creation in time $O^*(2^n)$, but we do not know how to do better. How can we close the gap between the (conditional) lower bound of $\left(\frac{2}{\phi}\right)^n$ and the upper bound of $O^*(2^n)$?

3.2.2 Extending degeneracy for compatibility graphs

In this section, we extend the notion of degeneracy for conflict graphs. To make the definitions similar to those before, we will work with the complement of the conflict graph, which we will call a compatibility graph. It will also be convenient to write the vertices x_i and $\overline{x}_i$ of the conflict graph as x_{i1} and x_{i0}, respectively. Thus the edge $\{x_i, \overline{x}_j\}$ can alternatively be written as $\{x_{i1}, x_{j0}\}$.

Given a compatibility graph one may ask if there is a binary assignment to the variables that avoids all conflicts. This is an instance of 2SAT, which can be solved in linear time [3] (possibly quadratic with respect to the number n of variables). We will say that a compatibility graph is feasible if such an assignment exists. This can be generalized for subgraphs of compatibility graphs, and feasibility can still be determined in linear time.

Definition 7 (feasible subgraph)**.** *A subgraph $G' = (V', E')$ of a compatibility graph $G = (V, E)$ is said to be feasible if and only if there exists a mutually compatible subset $S \subseteq V'$ of n vertices, i.e., for every pair of distinct vertices $u, v \in S$, we have $\{u, v\} \in E'$. Otherwise, G' is said to be infeasible.*

We remark that the class of infeasible subgraphs of a compatibility graph is closed under taking induced subgraphs. This is easy to see because the n "mutually compatible" vertices are a clique. Note that a feasible 0-1 program implies a feasible subgraph, but the converse may not be true.

For simple undirected graphs the open (closed) neighborhood of a vertex is denoted by $N(\cdot)$ ($N[\cdot]$). A similar notion, specifically for compatibility graphs, is denoted by the calligraphic $\mathcal{N}(\cdot)$.

Definition 8 (compatibility-neighborhood)**.** *Given a feasible subgraph $G' = (V', E')$ of a compatibility graph G, the compatibility-neighborhood of a vertex $x_{it} \in V'$ in G'*

is denoted by $\mathcal{N}_{G'}(x_{it}) = \{j \mid \{x_{it}, x_{j0}\} \in E' \text{ and } \{x_{it}, x_{j1}\} \in E'\}$*, which is the set of indices of variables that remain free when fixing* $x_i = t$*.*

Definition 9 (compatibility degeneracy)**.** *A compatibility graph is said to be* d*-compatibility-degenerate if for every feasible subgraph* $G' = (V', E')$ *there exists a vertex* $x_{it} \in V'$ *with compatibility degree* $|\mathcal{N}_{G'}(x_{it})| \leq d$*. The compatibility degeneracy is the smallest value of* d *such that the graph is* d*-compatibility-degenerate.*

Just as a d-degenerate graph admits a degeneracy-ordering, a d-compatibility-degenerate graph admits a compatibility-degeneracy ordering. We will see that a compatibility-degeneracy ordering can also be computed in polynomial time.

Proposition 3. *A compatibility graph* G *is* d*-compatibility-degenerate if and only if it admits an ordering* $(x_{i_1t_1}, \ldots, x_{i_{2n}t_{2n}})$ *of its vertices such that for every vertex* $x_{i_jt_j}$ *either* $G[S_j]$ *is infeasible or* $|\mathcal{N}_{G[S_j]}(x_{i_jt_j})| \leq d$*, where* $S_j = \{x_{i_jt_j}, \ldots, x_{i_{2n}t_{2n}}\}$*.*

Proof. It is clear that a d-compatibility-degenerate graph admits such an ordering: iteratively remove a vertex of minimum compatibility degree and append it to the ordering (until the subgraph is infeasible). Once the subgraph is infeasible, append the remaining vertices to the ordering arbitrarily. By definition of d-compatibility-degeneracy, the subgraph obtained after removing each vertex in this way will have minimum compatibility degree at most d, or the subgraph will be infeasible.

For the other direction, consider a satisfactory vertex ordering $(x_{i_1t_1}, \ldots, x_{i_{2n}t_{2n}})$. We want to show that every feasible subgraph of G has a vertex of compatibility degree at most d. We can assume, without loss of generality, that G' is induced by a vertex subset S and that $G' = G[S]$ is feasible. Let $v \in S$ be the earliest vertex (among vertices from S) in the ordering. By the assumption about the ordering, $|\mathcal{N}_{G'}(v)| \leq d$, and since G' was chosen arbitrarily, G is d-compatibility-degenerate.

□

Thus, if we have fixed $x_{i_j} = t_j$, there are at most d variables whose vertices occur later in the ordering, i.e., at most d variables remain undetermined. This is the main idea that will be exploited in the algorithm that is based on compatibility degeneracy.

Now we describe a different parameter related to the compatibility graph that is based on pairs of adjacent vertices. This results in the notion of bicompatibility degeneracy d', which is a generalization of community degeneracy introduced in our previous paper [26]. We will show that the bicompatibility degeneracy (and such an ordering) can be found in polynomial time.

Definition 10 (bicompatibility degeneracy)**.** *A compatibility graph is said to be d-bicompatibility-degenerate if for every feasible subgraph $G' = (V', E')$ there exists an edge $\{x_{is}, x_{jt}\} \in E'$ with $|\mathcal{N}_{G'}(x_{is}) \cap \mathcal{N}_{G'}(x_{jt})| \leq d$. The bicompatibility degeneracy d' is the smallest value of d such that the graph is d-bicompatibility-degenerate.*

Given a graph $G = (V, E)$, the *edge-induced* subgraph of $E' \subseteq E$, denoted $G[E'] = (V', E')$, includes those vertices V' that are an endpoint of an edge in E'.

Proposition 4. *A compatibility graph G is d-bicompatibility-degenerate if and only if it admits an ordering $(e_1, \ldots, e_m)$ of its edges such that for every edge $e_k = \{u, v\}$ either $G[E_k]$ is infeasible or $|\mathcal{N}_{G[E_k]}(u) \cap \mathcal{N}_{G[E_k]}(v)| \leq d$, where $E_k = \{e_k, \ldots, e_m\}$.*

Proof. The proof is very similar to that of Proposition 3 and is omitted. □

3.2.3 Algorithms based on compatibility degeneracy

Before providing the main algorithm, we describe an efficient algorithm for finding a compatibility-degeneracy ordering.

Algorithm 7 An algorithm for finding a compatibility-degeneracy ordering.

Data: a $(2n)$-vertex compatibility graph $G = (V, E)$.

Result: a compatibility-degeneracy ordering $(x_{i_1t_1}, \ldots, x_{i_{2n}t_{2n}})$.

Initialize $V' \leftarrow V$

while $V' \neq \emptyset$ **do**

let $v \in V'$ be a vertex with minimum compatibility degree $|\mathcal{N}_{G[V']}(v)|$

append v to compatibility-degeneracy ordering

$V' \leftarrow V' \setminus \{v\}$

end

return *compatibility-degeneracy ordering* $(x_{i_1t_1}, \ldots, x_{i_{2n}t_{2n}})$

Lemma 18. *Algorithm 7 finds a compatibility-degeneracy ordering of a* $(2n)$*-vertex graph and can be implemented to run in time and space* $O(n^2)$*.*

Proof. The algorithm mimics the degeneracy algorithm [80], as described in the proof of Proposition 3, and is clearly correct. In each iteration of the while-loop, the algorithm finds a vertex of minimum compatibility degree and updates the compatibility degrees in $O(n)$ time. There are $O(n)$ iterations for a total runtime of $O(n^2)$. □

We note that after Algorithm 7 has been executed, the actual value of the compatibility degeneracy d can be calculated in time $O(n^2 \log n)$. Namely, perform binary search across the subgraphs $G[S_j]$ for $j = 1, \ldots, 2n$, where $S_j = \{x_{i_jt_j}, \ldots, x_{i_{2n}t_{2n}}\}$ to find the last subgraph $G[S_q]$ that is feasible. Then, let $d = \max_{j \in [q]}\{|\mathcal{N}_{G[S_j]}(x_{i_jt_j})|\}$.

Algorithm 8 An algorithm for solving 0-1 programs, parameterized by compatibility degeneracy.

Data: m-vector b; n-vector c; $m \times n$ matrix A;

an associated $(2n)$-vertex compatibility graph $G = (V, E)$,

where for every variable x_i we have

vertices x_{i0} (x_{i1}) representing $x_i = 0$ ($x_i = 1$).

Result: $z^* = \max\{c^T x : Ax \leq b, x \in \{0,1\}^n\}$. ($z^* = -\infty$ when infeasible).

compute a compatibility-degeneracy ordering $(x_{i_1 t_1}, \ldots, x_{i_{2n} t_{2n}})$ of G

let $S_j = \{x_{i_j t_j}, \ldots, x_{i_{2n} t_{2n}}\},\ j = 1, \ldots, 2n$

find last feasible subgraph, i.e., $q = \max_{j \in [2n]}\{j \mid G[S_j] \text{ is feasible}\}$, via binary search

for $j = 1, \ldots, q$ **do**

$F_j = \{x \in \{0,1\}^n \mid x_{i_k} = 1 - t_k\ \forall x_{i_k t_k} \notin N_{G[S_j]}[x_{i_j t_j}]\}$; `// fix vars`

$z_j \leftarrow \max_x\{c^T x \mid Ax \leq b,\ x \in F_j\}$; `// −∞ if infeasible`

end

return $z^* = \max\{z_j : 1 \leq j \leq q\}$

For the following theorem, let $T_{d,m}$ denote the time to solve a 0-1 program with d variables and m constraints. In general, the best known bound for $T_{d,m}$ is achieved by a simple recursive algorithm that runs in time $O(m2^d)$. Unfortunately, this is essentially best-possible if the Strong Exponential Time Hypothesis holds [61, 29], but it can be improved when restricted to 0-1 programs for which $m = O(d)$ [62].

Theorem 7. *Algorithm 8 solves 0-1 programs with d-compatibility-degenerate graphs in time* $O(n^2 \log n + n^2 m + nT_{d,m}) = O^*(2^d)$.

Proof. First we examine the runtime. The degeneracy ordering can be found in time $O(n^2)$ by Lemma 18. The value of q can be found in time $O(n^2 \log n)$: check feasibil-

ity of $G[S_j]$ in time $O(n^2)$ using a 2SAT algorithm [3] and perform $O(\log n)$ iterations using binary search. Before solving the subproblem for z_j, fix the appropriate variables and update the right-hand-sides. The process of updating the right-hand-sides takes $O(nm)$ time. Solving for z_i takes time $T_{d,m} = O(m2^d)$ since the subproblem has at most d non-fixed variables by the compatibility-degeneracy ordering. This proves the runtime.

Now we show correctness, claiming that $z^* = \max_{j\in[q]}\{z_j\}$. Clearly $z^* \geq \max_{j\in[q]}\{z_j\}$, since z_j is the optimal objective over a smaller feasible region. To establish the reverse inequality, consider an optimal solution $x^* \in \{0,1\}^n$ to the original problem (assuming it is feasible). Consider the n corresponding vertices $V^* = \{x_{i,t_i} \mid x_i^* = t_i,\ i = 1,\ldots,n\}$ and the earliest vertex $v \in V^*$ in the ordering among vertices from V^*. Let its position in the ordering be $k \leq q$. Then the point x^* is feasible to subproblem k, so $\max_{j\in[q]}\{z_j\} \geq z_k \geq z^*$. □

We note that Algorithm 8 can be modified to slightly improve the runtime to $O\left(n^2 \log n + (n-d)(nm + T_{d,m})\right)$. When $q \leq n-d$, the algorithm is the same. However, when $q > n-d$, solve the first $n-d$ subproblems as usual, and solve one last subproblem where we fix $x_{i_k} = 1 - t_k\ \forall k \leq n-d$ and all other variables are free. The same idea slightly improves the runtime in Theorem 8 below as well.

Theorem 8. *All feasible solutions of d-compatibility-degenerate 0-1 programs can be listed in time $O(n^2(\log n + m2^d)) = O^*(2^d)$. Any such algorithm requires $\Omega(n(n-d)2^d)$ time, so the approach is optimal to within a polynomial factor.*

Proof. The approach is similar to Algorithm 8. Instead of solving for z_j, check all possible solutions (by exhaustive search) of the at most d unfixed variables, and list the feasible points. For the second claim, it is sufficient to find an infinite class of

d-degenerate graphs with $(n-d+1)2^d$ cliques. By [111], for all $n \geq d$ there exists a d-degenerate n-vertex graph with $(n-d+1)2^d$ cliques. □

This is not the only algorithm that can be shown to list all solutions in time $O^*(2^d)$. In another approach, one could rely on a 2SAT enumeration algorithm that runs with polynomial delay, e.g., [37]. In this case, list all of the $O(n2^d)$ possible solutions and check each to see if it is feasible for the 0-1 program. Such an algorithm would run in time $O((D+nm)n2^d)$, where D denotes the delay of the 2SAT enumeration algorithm. Note that there exist 2SAT enumeration algorithms for which D is linear in the size of the 2SAT instance [96]. Thus, $D = \Omega(n^2)$ when the formula has many clauses, but it could be that $D = O(n)$. Either approach might be preferred depending on the values of d, D, and n.

3.2.4 Algorithms based on bicompatibility degeneracy

We first provide an efficient algorithm for computing a bicompatibility-degeneracy edge ordering. The algorithm mimics the community degeneracy algorithm [26], and its runtime is derived with similar arguments.

Algorithm 9 An algorithm for finding a bicompatibility-degeneracy edge ordering.

Data: a $(2n)$-vertex compatibility graph $G = (V, E)$.
Result: a bicompatibility-degeneracy edge ordering $(e_1, \ldots, e_{|E|})$.
Initialize $E' \leftarrow E$
while $E' \neq \emptyset$ **do**
  let $e = \{u, v\} \in E'$ be an edge with minimum $|\mathcal{N}_{G[E']}(u) \cap \mathcal{N}_{G[E']}(v)|$
  append e to bicompatibility-degeneracy ordering
  $E' \leftarrow E' \setminus \{e\}$
end
return *bicompatibility-degeneracy edge ordering* $(e_1, \ldots, e_{|E|})$

Lemma 19. *Algorithm 9 finds a bicompatibility-degeneracy edge ordering of a* $(2n)$*-vertex* $|E|$*-edge compatibility graph and can be implemented to run in time* $O(n|E|^2)$*.*

We note that after Algorithm 9 has been executed, the actual value of the bicompatibility degeneracy d' can be calculated in time $O(n^2 \log n)$. Namely, perform binary search across the subgraphs $G[E_j]$ for $j = 1, \ldots, |E|$, where $E_j = \{e_j, \ldots, e_{|E|}\}$ to find the last subgraph $G[E_q]$ that is feasible. Then let $d' = \max_{j \in [q]}\{|\mathcal{N}_{G[E_j]}(u_j) \cap \mathcal{N}_{G[E_j]}(v_j)|\}$.

Algorithm 10 An algorithm for solving 0-1 programs, parameterized by bicompatibility degeneracy.

Data: m-vector b; n-vector c; $m \times n$ matrix A;

an associated $(2n)$-vertex compatibility graph $G = (V, E)$,

where for every variable x_i we have

vertices x_{i0} (x_{i1}) representing $x_i = 0$ ($x_i = 1$).

Result: $z^* = \max\{c^T x : Ax \leq b, x \in \{0,1\}^n\}$. ($z^* = -\infty$ when infeasible).

compute a bicompatibility-degeneracy edge ordering $(e_1, \ldots, e_{|E|})$ of G

let $E_k = \{e_k, \ldots, e_{|E|}\},\ k = 1, \ldots, |E|$

find last feasible subgraph, i.e., $q = \max_{k \in [|E|]}\{k \mid G[E_k] \text{ is feasible}\}$ via binary search

for $k = 1, \ldots, q$ **do**

$\{u, v\} \leftarrow e_k$

$F_k = \{x \in \{0,1\}^n : x_{i_k} = 1 - t_k\ \forall x_{i_k t_k} \notin N_{G[E_k]}[u] \cap N_{G[E_k]}[v]\}$

$z_k \leftarrow \max_x\{c^T x : Ax \leq b,\ x \in F_k\}$; `// -∞ if infeasible`

end

return $z^* = \max\{z_k : 1 \leq k \leq q\}$

Theorem 9. *Algorithm 10 solves 0-1 programs with d'-bicompatibility-degenerate graphs $G = (V, E)$ in time $O\left(n^2 \log n + |E|(n|E| + nm + T_{d',m})\right) = O^*(2^{d'})$. Using q processors (where $q = O(|E|)$, as defined in Algorithm 10), this reduces to time $O\left(n|E|^2 + n^2 \log n + nm + T_{d',m})\right)$.*

Proof. The proof is similar to that of Theorem 5 and is omitted. The key point is that the subproblems have at most d' unfixed variables by the bicompatibility-degeneracy ordering. □

Lemma 20. *For any $d' \leq n$, with $n - d'$ even, there exists a d'-community-degenerate graph with $[\frac{1}{4}(n - d')^2 + (n - d') + 1]2^{d'}$ cliques.*

Proof. Recall that, by definition of [26], every non-edgeless subgraph G' of a d'-community-degenerate graph has an edge $\{u, v\}$ with $|N_{G'}(u) \cap N_{G'}(v)| \leq d'$. For all d' and $n \geq d'$ such that $n - d'$ is even, we construct such a graph $G = (U \cup V_1 \cup V_2, E)$. Let $|U| = d'$ and $|V_1| = |V_2| = (n - d')/2$. Let G have all edges, except that V_1 and V_2 should each be independent sets in G. The graph is d'-community-degenerate: find an appropriate ordering by first removing all edges with both endpoints in $V_1 \cup V_2$, then all edges with one endpoint in $V_1 \cup V_2$, then all edges with no endpoints in $V_1 \cup V_2$. There are $2^{d'}$ cliques in $G[U]$. Any such clique can be enlarged by adding any one of the $\frac{1}{4}(n - d')^2$ pairs of adjacent vertices from $V_1 \cup V_2$ or any of the $(n - d')$ vertices from $V_1 \cup V_2$. □

Theorem 10. *All feasible solutions of d-compatibility-degenerate 0-1 programs can be listed in time $O^*(2^{d'})$. Any such algorithm requires $\Omega(n(n - d')^2 2^{d'})$ time, so the approach is optimal to within a polynomial factor.*

Proof. The proof of the first claim is similar to that of Theorem 8. The second claim follows by Lemma 20. □

3.2.5 Preliminary computations

For the proposed algorithms to be practically useful, there should exist real-life 0-1 programs for which compatibility degeneracy d or bicompatibility degeneracy d' are small compared to the number n of 0-1 variables. Here, we calculate d for some problems that were considered by [4] and [26].

To formulate the maximum clique problem, one typically uses a binary variable x_i for each vertex i, and for each pair of nonadjacent vertices i, j the constraint $x_i + x_j \leq 1$ is added. So, the conflict graph (based solely on feasibility conflicts) closely resembles the complement of the input graph, and the compatibility graph resembles the input graph. In fact, for clique, the values of compatibility degeneracy (in the compatibility graph) and degeneracy (in the input graph) coincide.

Table 3.3: Degeneracy d of graphs (i.e., compatibility degeneracy for max-clique). The left (right) table includes those graphs from the 2nd (10th) DIMACS Challenge [65] ([5]) that were considered by [4] ([26]).

Graph	n	d
brock200_2	200	84
c-fat200-1	200	14
c-fat200-2	200	32
p_hat300-1	300	49
san200_0.7_2	200	125

Graph	n	d
as-22july06	22,963	25
kron_g500-simple-logn16	65,536	432
citationCiteseer	268,495	15
ldoor	952,203	34
in-2004	1,382,908	488
cage15	5,154,859	25
uk-2002	18,520,486	943

We now turn to MIPLIB 3.0 [13] instances that were considered by [4]. The instance air03 is also included as it had the largest difference between d and n. Note that the value of d depends largely on the process used to generate the conflict graph. Here, the conflict graph is generated using built-in functions of GLPK [77]. Denser

conflict graphs could be generated with more computational effort. However, since the MIPLIB instances are known to be challenging, one should expect that the values of d and n will be close to each other. The instance vpm2 illustrates the worst-case example. Its conflict graph has only the trivial edges; for each variable x_i, the edge $\{x_{i0}, x_{i1}\}$ appears.

Table 3.4: Compatibility-degeneracy d for some instances from MIPLIB 3.0 [13]. The parameter n refers to the number of 0-1 variables, $|\overline{E}|$ is the number of conflict edges, and ρ is the density of the conflict graph as a percentage (rounded to three significant digits).

Instance	n	d	$\lvert\overline{E}\rvert$	ρ
air03	10,757	3,301	32,095,418	13.9%
air04	8,904	7,899	2,121,648	1.34%
air05	7,195	5,857	2,527,253	2.44%
dcmulti	75	72	107	0.957%
mitre	10,724	10,696	155,498	0.0676%
vpm2	168	167	168	0.299%

4. FIXED-PARAMETER TRACTABLE EXTENDED FORMULATIONS

This chapter is based on work with Sergiy Butenko [24]. Here, we apply ideas from parameterized complexity to the study of extended formulations. When constructing an extended formulation for an optimization problem, an important thing to keep in mind is its size. However, due to the computational intractability of many problems, one cannot always expect to find polynomial-size extended formulations. To deal with this, one approach is to introduce a parameter, and look for extended formulations whose size grows polynomially in the problem size, but exponentially (or worse) in the parameter. We illustrate this approach on the independent set and vertex cover problems and consider parameters such as the number μ of maximal independent sets (or minimal vertex covers), the treewidth $\mathbf{tw}$ of the graph, and the size k of the vertex cover.

The first formulation is for arbitrary independence systems and has size $O(n+\mu)$, which implies size $O(1.4423^n)$ for the independent set polytope of graphs.

The second formulation, of size $O(2^{\mathbf{tw}}n)$, applies to both independent set and vertex cover and relies on a framework for generating extended formulations from dynamic programs due to Martin et al. [79]. This improves upon the size $O(n^{\mathbf{tw}+1})$ extended formulations implied by the Sherali-Adams reformulation procedure [103] (as shown by Bienstock and Ozbay [12]). This leads to small formulations for particular classes of graphs: size $O(n)$ extended formulations for outerplanar, series-parallel, and Halin graphs; size $2^{O(\sqrt{n})}$ extended formulations for planar graphs; and size $O(1.2247^n)$ extended formulations for graphs of maximum degree three.

The third and fourth extended formulations are for the cardinality-constrained variants. The third has size $O(n\mu_k)$ where μ_k denotes the number of maximal in-

dependent sets that have size at least k (or minimal vertex covers that have size at most k). This implies size $O(2^k n)$ extended formulations for the k-vertex cover polytope, which significantly improves upon the naive $O(n^k)$ extended formulation. A more complicated approach yields an extended formulation for k-vertex covers of size $O(1.466^k n^2)$ using insights of Chen et al. [31].

4.1 Background on Extended Formulations and Independent Set Polytope

Frequently, when one wants to solve a discrete optimization problem, an integer programming (IP) formulation is created. However, it is generally difficult to solve, partially because the linear programming relaxation does not do a good job of approximating the integer hull. In some cases, the formulation can be modified (by adding variables and constraints), so that the resulting linear programming relaxation is tight. In this case, one can drop the integrality constraints and solve a linear program. This modified formulation is called an extended formulation. In this chapter, the polyhedron in the following definition is typically the integer hull for a particular combinatorial optimization problem (and not its linear programming relaxation).

Definition 11. *Let $P = \{x \mid Ax \leq b\} \subseteq \mathbb{R}^n$ be a polyhedron. A polyhedron $Q \subseteq \mathbb{R}^d$ is an* extension *for P if $\text{proj}_x(Q) = P$, where $\text{proj}_x(Q) := \{x \mid \exists y : (x, y) \in Q(G)\}$. The* size *of an extension is the number of its facets.*

Extended formulations for numerous combinatorial optimization problems can be found in literature; consult the surveys of Conforti et al. [34] and Kaibel [66] for some notable cases. We mention two important "meta" extended formulations that will be useful later. Balas [6, 7] creates polysize extended formulations for the union of polyhedra. Martin et al. [79] craft extended formulations for a broad class of dynamic programs.

Only recently have researchers shown that many important combinatorial optimization problems have high *extension complexity*, that is, if one wants to drop the integrality constraints, a very large number of constraints must be added in the worst case.

Definition 12. *The* extension complexity *of a polyhedron P is*

$$\mathrm{xc}(P) := \min\{\mathrm{size}(Q) \mid Q \textit{ is an extension for } P\}.$$

Since the pioneering work of Yannakakis [112], there have been numerous advances showing that certain polytopes have high extension complexity. For example, it has been shown that polytopes associated with NP-hard problems such as the traveling salesman problem and the 0-1 knapsack problem admit no polysize extended formulation [49, 94]—irrespective of whether P=NP. There are even polytime solvable problems, such as matching, that do not admit polysize extended formulations [97].

Research into extended formulations often mimics the trajectory of algorithm design after the theory of NP-completeness was introduced. Under widely-held beliefs in complexity theory, "no algorithm *exactly* solves *all* instances in *polynomial time*." Relaxing the words in this statement leads, roughly, to the fields of approximation algorithms, parameterized algorithms, and exponential algorithms. Each has a polyhedral counterpart.

The independent set and vertex cover problems have both been studied from the perspective of approximate extended formulations. It has been shown that independent set admits no polynomial-size *uniform* extended formulation[1] that achieves an $O(n^{1-\epsilon})$ approximation for any constant $\epsilon > 0$ [22], which matches the inapproxima-

[1]This notion of extended formulation is different than that of Definition 11 as it refers to the optimal objective value of the LP relaxation, not necessarily a polyhedral approximation.

bility of the maximum independent set problem [59, 113]. If we allow for *non-uniform* extended formulations, that is, the inequalities defining the feasible region need not be the same for every n-vertex graph, $O(1)$-approximate formulations still require superpolynomial size [9]. Somewhat surprisingly, however, a $O(\sqrt{n})$-approximation can be achieved with size $O(n)$ extended formulations [9]. For the vertex cover problem, no polysize extended formulation achieves a $(2-\epsilon)$-approximation [9], and the naive linear programming relaxation provides a matching upper bound of 2.

Prior to our work, little was known about parameterized and exponential-size extended formulations for independent set and vertex cover. It was even mentioned [21] as an open question whether it was possible to beat 2^n. We show a size $O(1.4423^n)$ bound. Perhaps the most interesting previous work was that of Bienstock and Ozbay [12], who showed that **tw** levels of Sherali-Adams reformulation procedure [103], when applied to the traditional edge formulation, are enough to recover the independent set polytope of the graph. This shows that there are size $O(n^{\mathbf{tw}+1})$ extended formulations. Our formulation significantly improves this bound to $O(2^{\mathbf{tw}}n)$. After we posted this work online [24], the results were generalized to 0-1 programs by Bienstock and Munoz [11] and to Constraint Satisfaction Problems by Kolman and Koutecký [70].

There are also some small extended formulations for independent set and vertex cover for particular classes of graphs. Barahona and Mahjoub [8] showed that, in the case of series-parallel graphs, there are linear-size extended formulations for independent set. (Series-parallel graphs have treewidth at most 2, so our formulation is also of linear-size.) In the case of bipartite graphs, there are linear-size formulations (with no need for extra variables) using the 0-1 bounds and edge inequalities [57]. Perfect graphs admit size $n^{O(\log n)}$ extended formulations [112]. There are also polynomial-size formulations for comparability graphs and chordal graphs, which are subclasses

of perfect graphs [112].

In the following definitions, the characteristic vector x^S of $S \subseteq V$ is an n-dimensional 0-1 vector that has $x_i^S = 1$ if and only if $i \in S$.

Definition 13. *The independent set polytope of a graph $G = (V, E)$ is*

$$\begin{aligned} P(G) &= \text{conv.hull}\{x^S \in \{0,1\}^n \mid S \textit{ is an independent set of } G\} \\ &= \text{conv.hull}\{x \in \{0,1\}^n \mid x_i + x_j \leq 1 \textit{ for every } \{i,j\} \in E\}. \end{aligned}$$

Definition 14. *The vertex cover polytope of a graph $G = (V, E)$ is*

$$\begin{aligned} Q(G) &= \text{conv.hull}\{x^S \in \{0,1\}^n \mid S \textit{ is a vertex cover for } G\} \\ &= \text{conv.hull}\{x \in \{0,1\}^n \mid x_i + x_j \geq 1 \textit{ for every } \{i,j\} \in E\}. \end{aligned}$$

Before proceeding, we make the following simple, but important, connection between the independent set polytope $P(G)$ of a graph G and its vertex cover polytope $Q(G)$.

Lemma 21 (folklore)**.** *The equality $P(G) = \mathbf{1} - Q(G)$ holds, that is, $x \in P(G)$ if and only if $\mathbf{1} - x \in Q(G)$.*

Proof. ($\Longrightarrow$) Let $x^* \in P(G)$. Without loss of generality, suppose that x^* is an extreme point and is hence the characteristic vector of an independent set S of G. It is easy to see that $V \backslash S$ is a vertex cover for G. Hence, $\mathbf{1} - x^* = \mathbf{1} - x^S = x^{V \backslash S} \in Q(G)$.

($\Longleftarrow$) Similarly. □

Proposition 5. *For any graph G, $\text{xc}(P(G)) = \text{xc}(Q(G))$.*

Proof. This follows from Lemma 21 by a change of variables. Namely, given an extended formulation for $P(G)$, we can construct an extended formulation for $Q(G)$

of the same size by replacing every instance of the variable x_i by $1 - x_i$. Hence $\text{xc}(Q(G)) \leq \text{xc}(P(G))$. The reverse inequality holds by the same argument. □

4.2 Formulation Based on Maximal Independent Sets

Our first extended formulation is fairly simple and is based on introducing a variable for each maximal independent set of the graph. The number μ of maximal independent sets of a graph is at most $3^{n/3}$, and this bound is tight on what we will call the MM graphs, which are the disjoint union of $n/3$ triangles. Historically, MM referred to Moon and Moser [84], but the same results were given several years earlier by Miller and Muller [82]. Further, all maximal independent sets of an arbitrary graph can be listed in time $O(3^{n/3})$ [23, 104]. In fact, there are output-sensitive algorithms that, for example, list all maximal independent sets in time $O(nm\mu)$, where m denotes the number of edges and μ denotes the number of maximal independent sets [105]. As a consequence, if a graph has polynomially many maximal independent sets, not only does its independent set polytope admit a compact extended formulation, but it can be constructed in polynomial time.

While the focus of this chapter is on the independent set polytope of graphs, we will state the first extended formulation for the more general case of an arbitrary independence system. An independence system is a pair $(I, \mathcal{I})$, where I is a finite ground set and $\mathcal{I}$ is a collection of subsets of I satisfying:

1. (non-emptiness) $\emptyset \in \mathcal{I}$, and

2. (down-monotonicity) $S \subseteq S' \in \mathcal{I}$ implies $S \in \mathcal{I}$.

In the extended formulation below, x is the decision vector representing the chosen independent set, and for every maximal independent set S, there is a variable y_S. Denote by $\mathcal{I}_M$ the set of all inclusion-wise maximal independent sets.

Extended Formulation 1:

$$\sum_{S \in \mathcal{I}_M} y_S = 1 \tag{4.1}$$

$$x_i \leq \sum_{S \in \mathcal{I}_M : i \in S} y_S, \text{ for every } i \in I \tag{4.2}$$

$$y_S \geq 0, \text{ for every maximal independent set } S \in \mathcal{I}_M \tag{4.3}$$

$$x_i \geq 0, \text{ for every } i \in I. \tag{4.4}$$

Note that the *maximal* independent set polytope admits an extended formulation of the same size and can be obtained by changing every inequality $x_i \leq \sum_{S \in \mathcal{I}_M : i \in S} y_S$ to an equality. Such an extended formulation is clearly correct as it simply writes x as a convex combination of maximal independent sets.

Lemma 22. *For an independence system $(I, \mathcal{I})$, let $F_1(I, \mathcal{I})$ be the set of all (x, y) satisfying constraints* (4.1)–(4.4)*. Then the projection of $F_1(I, \mathcal{I})$ onto the x variables is precisely $(I, \mathcal{I})$'s independence system polytope $P(I, \mathcal{I})$.*

Proof. First see that $P(I, \mathcal{I}) \subseteq \text{proj}_x F_1(I, \mathcal{I})$. Consider $x' \in P(I, \mathcal{I})$, which we can assume, without loss of generality, is integer. Then x' is the characteristic vector of some independent set I which is a subset of a maximal independent set I'. Then the binary vector (x', y') belongs to $F_1(I, \mathcal{I})$, where $y'_S = 1$ iff $I' = S$.

To show $P(I, \mathcal{I}) \supseteq \text{proj}_x F_1(I, \mathcal{I})$, let $(u, v) \in F_1(I, \mathcal{I})$. Then also $(x, v) \in F_1(I, \mathcal{I})$, where, for each $i \in I$, $x_i := \sum_{S \in \mathcal{I}_M : i \in S} v_S$. Note that $x \in P(I, \mathcal{I})$, since it belongs to the maximal independent set face of $P(I, \mathcal{I})$. Then, since $0 \leq u \leq x$, and by down-monotonicity of $P(I, \mathcal{I})$ (see, e.g., [58]), we have $u \in P(I, \mathcal{I})$. □

Theorem 11. *An independence system with n ground elements and μ maximal independent sets admits an extended formulation of size $2n + \mu$.*

Corollary 5. *The extension complexity of a graph's independent set polytope is at most* $2n + 3^{n/3} = O(1.4423^n)$.

Proof. This follows directly from Theorem 11 and the fact that a graph has at most $3^{n/3}$ maximal independent sets [82, 84]. □

It is not too hard to see that similar results hold if, instead of down-monotonicity, we enforce up-monotonicity, i.e., that $S \supseteq S' \in \mathcal{I}$ implies $S \in \mathcal{I}$. This allows us to write extended formulations for, say, the dominating set polytope of a graph with $O(1.7159^n)$ variables and constraints [51]. The extended formulation also implies that the dominating set polytope admits a compact extended formulation whenever the graph has polynomially many minimal dominating sets. However, it is not yet clear if such an extended formulation could be constructed in polynomial time, as, to date, there is no known *output-polynomial time* algorithm for enumerating minimal dominating sets [67]. This is to be expected for some independence systems, as it has been shown that no algorithm lists all maximal independent sets of an independence system in output-polynomial time, unless P=NP [72].

4.3 Formulation Based on Treewidth

The second extended formulation that we describe borrows ideas from a treewidth-based dynamic programming algorithm for independent set. We will first represent the problem as a network flow problem of sorts. The directed network that we construct has hyperarcs, complicating the proof of the linear programming formulation's integrality. For clarity, we will refer to the input graph of the independent set problem as a graph with vertices and edges; the directed graph that represents the network flow problem will be called a network with nodes and (hyper)arcs.

If we based the extended formulation on a pathwidth-based dynamic programming algorithm, then there would be no hyperarcs. In this case, it is pretty straight-

forward to achieve an extended formulation with $O(2^{\mathbf{pw}}n)$ entities, where **pw** denotes pathwidth. It turns out that $\mathbf{pw}(G) = O(\mathbf{tw}(G)\log n)$ so this would yield polynomial-size extended formulations for graphs of bounded treewidth. However, if we construct the formulation from a treewidth-based dynamic programming algorithm, then we can make a stronger claim—that graphs of bounded treewidth admit *linear-size* extended formulations for their independent set polytopes.

Since there is the possibility for hyperarcs, the usual total unimodularity argument is not enough to show that the proposed formulation is integral. Fortunately for us, Martin et al. [79] have shown how to craft extended formulations for these types of dynamic programs. We will only need to construct the necessary directed acyclic hypergraph and show that it fits into their paradigm. First, however, we will need some background information about treewidth and the treewidth-based dynamic programming algorithm.

Definition 15. *A* tree decomposition *of a graph $G = (V, E)$ is a pair $(\mathcal{B}, T)$, where $T = (J, F)$ is a tree and $\mathcal{B} = \{B_j \mid j \in J\}$ is a collection of subsets of V (each B_j is called a bag) such that*

- $\bigcup_{j \in J} B_j = V$;
- *for every edge $\{u, v\} \in E$ there is a bag that contains u and v; and*
- *for all $i, j, k \in J$: if j is on the path from i to k in T then $B_i \cap B_k \subseteq B_j$.*

The width of the decomposition is $\max_i\{|B_i|\}-1$. *The treewidth of G, denoted* $\boldsymbol{tw}(G)$, *is the minimum width among the tree decompositions of G.*

The "-1" in the definition of width is merely a cosmetic detail done so that the treewidth of a tree is one. A path decomposition is a tree decomposition, where T is further required to be a path graph. Pathwidth is defined similarly.

While many problems are quickly solvable on graphs of small treewidth, actually determining a graph's treewidth is NP-hard. However, Bodlaender's theorem states that, for any fixed w, there is a linear-time algorithm that finds a tree decomposition of width w (if one exists). Even though Bodlaender's algorithm runs in linear time for fixed w, its dependence on w is very large and the algorithm is notoriously impractical. Still, there are practical, linear-time algorithms for small values of treewidth, e.g., for $\mathbf{tw} = 1, 2, 3, 4$. Consult the surveys of Bodlaender for these and other facts about treewidth [15, 16].

It will be convenient to work with a *nice* tree decomposition, and from now on we will assume, without loss of generality, that our tree decompositions will be nice and will have $O(n)$ bags. This follows by a standard linear-time algorithm that, when given a tree decomposition, outputs a nice tree decomposition of the same width and with at most $4n$ bags (see Lemma 13.1.2 of [69]).

Definition 16. *A tree decomposition is* nice *if it is a rooted binary tree such that each node $j \in J$ is one of the following four types:*

- ***Leaf nodes*** *j are leaves of T and have $|B_j| = 1$.*
- ***Introduce nodes*** *j have one child c with $B_j = B_c + v$ for some vertex $v \in V$.*
- ***Forget nodes*** *j have one child c with $B_j = B_c - v$ for some vertex $v \in V$.*
- ***Join nodes*** *j have two children c_1 and c_2 with $B_j = B_{c_1} = B_{c_2}$.*

We will now describe the treewidth-based dynamic programming algorithm for weighted independent set [17]. For each bag $B_j \subseteq V$ and for every subset $S \subseteq B_j$ of the bag, let $f(j, S)$ be the weight of a maximum weight independent set I of the subgraph induced by V_j such that $S = I \cap B_j$. Here, V_j is the union of B_j along with all of its descendant bags (not necessarily direct descendants). Whenever S

is itself not independent, the subproblem is infeasible with the convention that its objective is $-\infty$. The formula for computing $f(j, S)$ depends on the type of bag B_j. The weight of a vertex v is denoted w_v, and the weight of $S \subseteq V$ is denoted by $w(S) := \sum_{v \in S} w_v$.

- **Leaf node**, where $B_j = \{v\}$. Set $f(j, \emptyset) = 0$ and $f(j, \{v\}) = w_v$.
- **Introduce node**, where $B_j = B_c + v$. For every $S \subseteq B_c$, set

$$f(j, S) = f(c, S), \text{ and}$$

$$f(j, S + v) = \begin{cases} w_v + f(c, S) & \text{if } S + v \text{ is independent} \\ -\infty & \textit{otherwise.} \end{cases}$$

- **Forget node**, where $B_j = B_c - v$. For every $S \subseteq B_j$, set

$$f(j, S) = \max\{f(c, S), f(c, S + v)\}.$$

- **Join node**, where $B_j = B_{c_1} = B_{c_2}$. For every $S \subseteq B_j$, set

$$f(j, S) = f(c_1, S) + f(c_2, S) - w(S).$$

The objective of the maximum independent set problem for the original graph can be found by looking at the root bag B_r and computing the maximum of $f(r, S)$ such that $S \subseteq B_r$.

Notice that the algorithm does not depend on the graph's structure, in the sense that dependent subsets are penalized in the objective with a weight of $-\infty$, instead of being explicitly excluded during algorithm's execution. For example, the complete

graph on n nodes and the empty graph on n nodes both admit the trivial tree decomposition where a single bag contains all vertices. The algorithm's execution on these two graphs with the trivial decomposition is essentially the same, and hence, a polyhedral representation of this dynamic programming algorithm will not describe the graph's independent set polytope. Hard constraints are necessary.

We are now ready to construct our directed acyclic hypergraph $D = (N, A)$ that will model the treewidth-based dynamic programming algorithm for the independent set problem for a graph $G = (V, E)$. The main idea is to disallow nodes that represent infeasible solutions, i.e., dependent subsets of vertices. We can assume, without loss of generality, that the given tree decomposition is *nicer* and has width w.

Definition 17. *A* **nicer** *tree decomposition is nice tree decomposition with $O(n)$ bags that is rooted at an empty bag.*

The node set N is created as follows. For every bag B_j in the tree decomposition, and for every subset $S \subseteq B_j$ *that is independent in* G (including the empty set), create a node S^j. This implies, by the nicer tree decomposition, a single node $t = \emptyset^r \in N$ from the empty root bag B_r that we will call the sink node. Finally, for every leaf bag B_j, create a source node s_j. The number of nodes is $|N| = O(2^w n)$, since there are $O(n)$ bags, and for each bag B_j there are at most $2^{|B_j|} \leq 2^{w+1}$ independent sets.

The arc set A will allow a partial solution to "grow" at introduce bags and "shrink" at forget bags. Create A as follows depending on the type of bag B_j.

- **Leaf node**, where $B_j = \{v\}$. Add the arcs $(\emptyset, s_j), (s_j, \emptyset^j)$, and $(s_j, \{v\}^j)$. Note that $(\emptyset, s_j)$ is strange in that it has no tail and is called a boundary arc in Theorem 12.

- **Introduce node**, where $B_j = B_c + v$. For every independent $S \subseteq B_c$, add the arc (S^c, S^j) and if $S + v$ is also independent, then add the arc $(S^c, (S + v)^j)$.

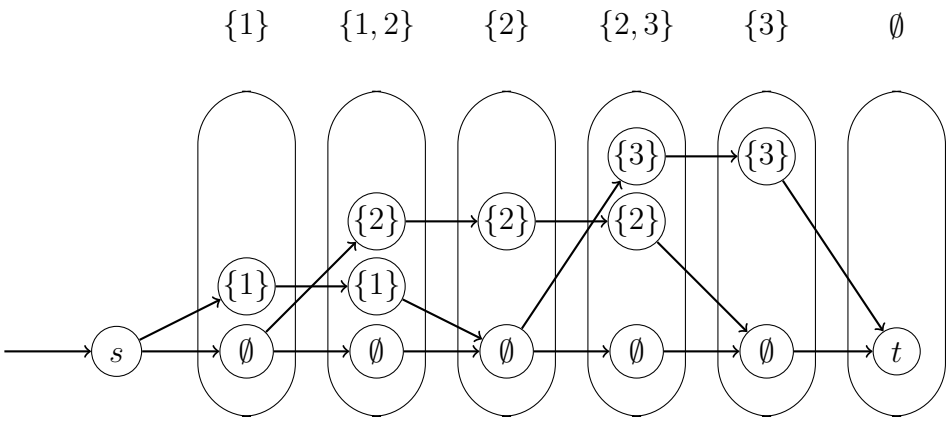

Figure 4.1: A *nicer* tree decomposition of P_3 (the path on 3 vertices) and the proposed construction D. (This is also a nice path decomposition.) There are no "join" nodes in the tree decomposition, so there is no need for hyperarcs.

- **Forget node**, where $B_j = B_c - v$. For every independent $S \subseteq B_j$, add the arc (S^c, S^j), and if $S + v$ is also independent, then add the arc $((S + v)^c, S^j)$.
- **Join node**, where $B_j = B_{c_1} = B_{c_2}$. For every independent subset $S \subseteq B_j$, add the *hyperarc* $(\{S^{c_1}, S^{c_2}\}, S^j)$.

The c in S^c and $(S + v)^c$ refers to bag B_c and not to the set's complement.

Examples of the constructed hypergraphs can be found in Figures 4.1, 4.2, and 4.5. Figure 4.1 illustrates the most basic case, where each node in the directed network represents at most one vertex in the input graph and there are no 'join' bags in the tree decomposition. Figure 4.2 shows an example where some bags contain independent sets of size two. Figure 4.3 shows the smallest graph with $\mathbf{tw} = 1$ and $\mathbf{pw} = 2$. A nicer tree decomposition and constructed hypergraph follow in Figures 4.4 and 4.5. Since the given tree decomposition has 'join' bags, there are hyperarcs in the directed network.

We are now ready to provide the extended formulation. For each (hyper)arc $a \in A$ of D, there is a variable y_a representing the amount of flow across it. As

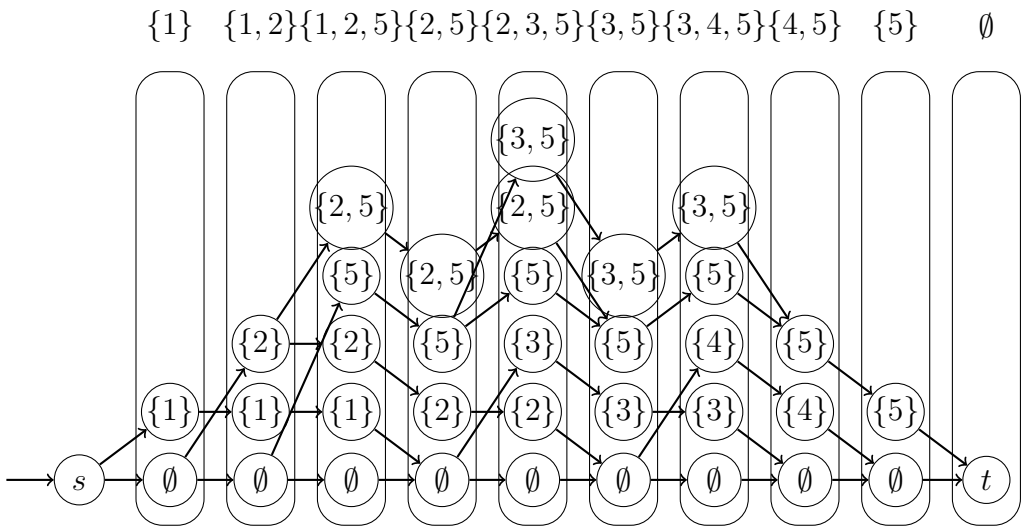

Figure 4.2: A width-2 *nicer* tree decomposition of the cycle graph on five vertices and the proposed construction D. (This is also a path decomposition.)

usual, x is the decision vector representing the chosen independent set of G. For a node $v \in N$, $\delta^{out}(v)$ is the set of (hyper)arcs that have v as (one of) its tail(s). The set $\delta^{in}(v)$ is defined similarly. The set FORGET(v) is the set of all arcs that "forget" $v \in N$, i.e., arcs of the form $((S+v)^c, S^j)$. The polytope $F_2(G)$ is the set of all (x, y) satisfying the following constraints.

Extended Formulation 2:

$$\sum_{a \in \delta^{in}(t)} y_a = 1, \text{ for sink node } t \tag{4.5}$$

$$\sum_{a \in \delta^{out}(v)} y_a - \sum_{a \in \delta^{in}(v)} y_a = 0, \text{ for every node } v \in N \setminus \{t\} \tag{4.6}$$

$$x_i - \sum_{a \in \text{FORGET}(i)} y_a = 0, \text{ for every vertex } i \in V \tag{4.7}$$

$$y_a \geq 0, \text{ for every (hyper)arc } a. \tag{4.8}$$

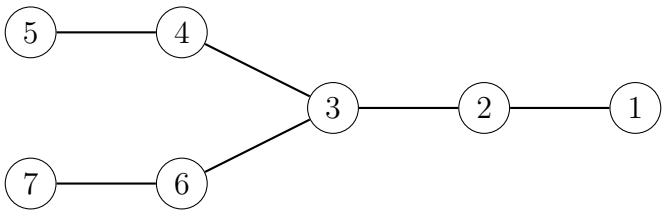

Figure 4.3: A tree (of pathwidth 2).

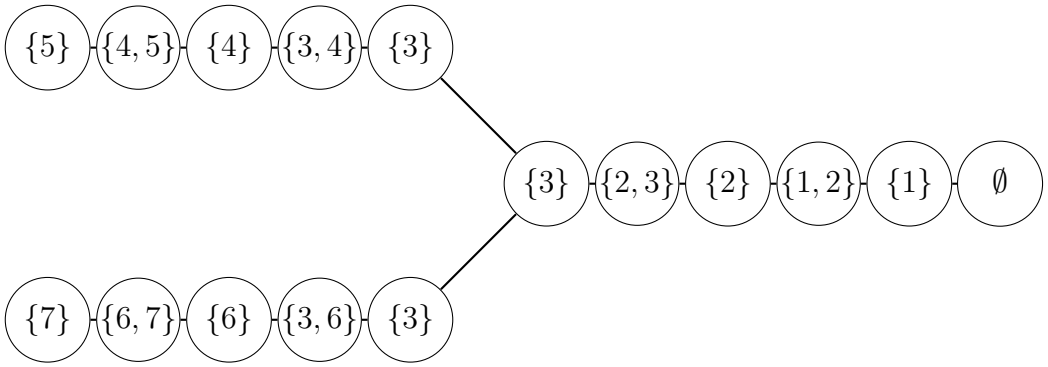

Figure 4.4: A *nicer* tree decomposition of width 1 that is rooted at the right.

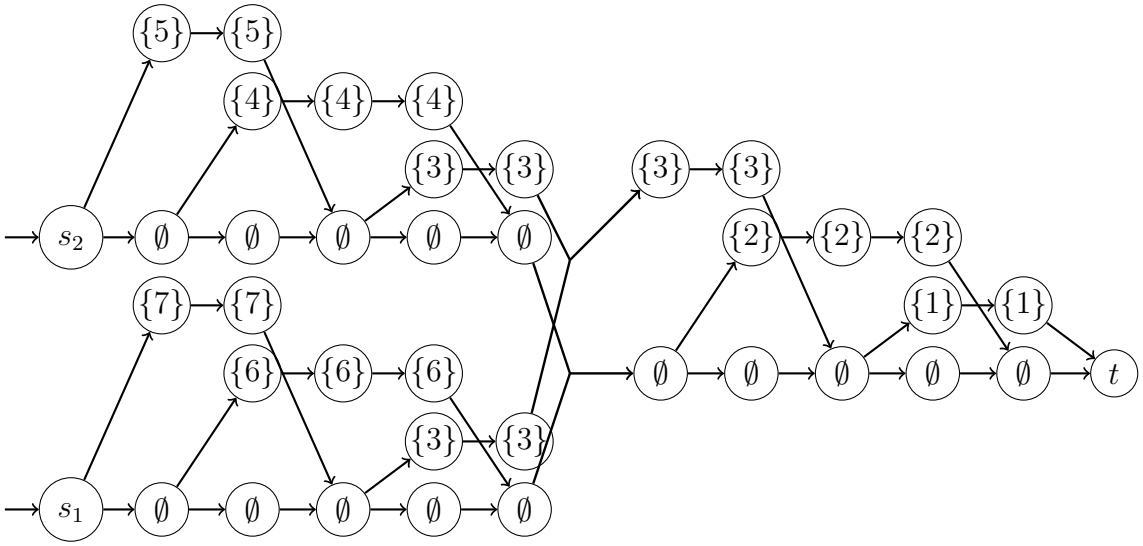

Figure 4.5: The proposed directed acyclic hypergraph D. Since there is a "join" node in the tree decomposition, D has hyperarcs.

Theorem 12 (Martin et al. [79]). *Let $\mathcal{H} = (\mathcal{V}, \mathcal{A})$ be a directed hypergraph such that*

1. *each hyperarc has a single head, i.e., hyperarcs are of the form (J, i) where $J \subseteq \mathcal{V}$ and $i \in \mathcal{V}$;*

2. *$\mathcal{H}$ is acyclic; more specifically, there is a mapping $\sigma : \mathcal{V} \to \mathbb{R}$ such that for every hyperarc $(J, i) \in \mathcal{A}$ and every $j \in J$, we have $\sigma(j) < \sigma(i)$;*

3. *there is finite set Q and a mapping $f : \mathcal{V} \to 2^Q$ such that*

   (a) *f is "consistent" with the acyclicity, namely, for every hyperarc $(J, i) \in \mathcal{A}$ and for every $j \in J$, we have $f(j) \subseteq f(i)$;*

   (b) *for every hyperarc $(J, i) \in \mathcal{A}$ and for distinct "tails" $j, j' \in J$ of the hyperarc, we have $f(j) \cap f(j') = \emptyset$;*

   (c) *there is a single "sink" node t with $f(t) = Q$.*

4. *every $i \in \mathcal{V}$ has at least one incoming arc. Since the graph is acyclic this implies that some arcs (called boundary arcs) will have no tail nodes, i.e., arcs of the type (J, i) with $J = \emptyset$.*

Then, the set of all z satisfying the following constraints is a 0-1 polytope.

$$\sum_{a=(J,t)\in\mathcal{A}} z_a = 1 \tag{4.9}$$

$$\sum_{a=(J,i)\in\mathcal{A}} z_a - \sum_{a=(J,j)\in\mathcal{A}: i\in J} z_a = 0, \text{ for every node } i \in V \setminus \{t\} \tag{4.10}$$

$$z_a \geq 0, \ \forall a \in \mathcal{A}. \tag{4.11}$$

Lemma 23. $\mathrm{proj}_y(F_2(G))$ *is a 0-1 polytope.*

Proof. Apply Theorem 12. The directed hypergraph that we constructed clearly satisfies points 1, 2, and 4. For point 3, let Q be the set of source nodes, and for $v \in N$, let $f(v)$ be the set of source nodes from which there is a directed path to v in D. □

Lemma 24. *In a nicer tree decomposition, each vertex is 'forgotten' once, i.e., for each $v \in V$, there is one pair (B_j, B_c) of bags, where B_j is the parent of B_c, such that $B_j = B_c - v$.*

Proof. Each vertex is forgotten at least once, since each vertex belongs to at least one bag and all vertices have been forgotten by the empty root bag. Now suppose that a vertex is forgotten at least twice, so that there are distinct bags $B_{j_1} = B_{c_1} - v$ and $B_{j_2} = B_{c_2} - v$ that forget v. We consider two cases. In the first case, assume that one of the bags that forgets v is a descendant of the other bag that forgets v. Without loss of generality suppose that B_{j_2} is a descendant of B_{j_1}. Then, bags B_{c_2} and B_{c_1} both contain v, but bag B_{j_2} does not, yet it lies between B_{c_1} and B_{c_2}, contradicting the tree decomposition. In the second case, B_{j_1} is neither a descendant nor an ancestor of B_{j_2}. In this case, they lie in different branches of the tree and both of B_{j_1} and B_{j_2} lie on the unique path between B_{c_1} and B_{c_2}, and the same contradiction occurs. □

Note that for a feasible solution (x, y) to $F_2(G)$ there will be one unit of flow 'from' bag B_c 'to' its parent B_j. For example, when $B_j = B_c - v$, we have

$$\sum_{\substack{a=(S^c,S^j)\in A \\ \text{s.t. } S\subseteq B_c \text{ is independent}}} y_a + \sum_{\substack{a=((S+v)^c,S^j)\in A \\ \text{s.t. } S+v\subseteq B_j \text{ is independent}}} y_a = 1. \tag{4.12}$$

If this flow were greater (less) than one, then the flow into the sink node t would be greater (less) than one, violating constraint (4.5).

Lemma 25. *$F_2(G)$ is a 0-1 polytope.*

Proof. First see that $F_2(G)$ is an integral polytope, since $\text{proj}_y(F_2(G))$ is a 0-1 polytope (by Lemma 23), and since there is a nonnegative integer matrix M such that $x = My$. Now we must show that the x variables are bounded by zero and one. By Lemma 24, for any vertex $v \in V$, there will be one bag $B_j = B_c - v$ that forgets v. Then, for any $(x, y) \in F_2(G)$, we have that

$$\begin{aligned} 0 \leq x_v &= \sum_{a \in \text{FORGET}(v)} y_a \\ &\leq \sum_{\substack{a=(S^c,S^j)\in A \\ \text{s.t. } S \subseteq B_c \text{ is independent}}} y_a + \sum_{a \in \text{FORGET}(v)} y_a \\ &= \sum_{\substack{a=(S^c,S^j)\in A \\ \text{s.t. } S \subseteq B_c \text{ is independent}}} y_a + \sum_{\substack{a=((S+v)^c,S^j)\in A \\ \text{s.t. } S+v \subseteq B_j \text{ is independent}}} y_a = 1. \end{aligned}$$

□

Lemma 26. *$P(G) \subseteq \text{proj}_x(F_2(G))$.*

Proof. Consider $x \in P(G)$. Without loss of generality, suppose that x is an extreme point of $P(G)$, and is thus the characteristic vector of an independent set I. We construct an integral feasible point of $F_2(G)$ as follows. For every non-boundary arc $a = (S_1^c, S_2^j) \in A$ that is not a hyperarc, set

$$y_a = \begin{cases} 1, & \text{if } S_1 = B_c \cap I \text{ and } S_2 = B_j \cap I \\ 0, & \text{otherwise.} \end{cases}$$

For each boundary arc, set the corresponding variable to one. Similarly, for every hyperarc, say $a = (\{S^{c_1}, S^{c_2}\}, S^j) \in A$, set $y_a = 1$ iff $S = B_j \cap I$. Then, for every

arc, say $a = (s_j, S^i)$, emanating from a source node s_j, set $y_a = 1$ iff $S = B_i \cap I$. It can be verified that $(x, y) \in F_2(G)$. □

Lemma 27. $\text{proj}_x(F_2(G)) \subseteq P(G)$.

Proof. Consider $(x', y') \in F_2(G)$. Without loss of generality, suppose that (x', y') is an extreme point of $F_2(G)$. By Lemma 25, this means that (x', y') is 0-1. We are to show that $x' \in P(G)$. By the flow constraints of $F_2(G)$, the integrality of (x', y'), and equality (4.12), the set of all arcs with positive flow induce a directed tree of D—a sort of reverse arborescence rooted at the sink $\emptyset^t$ with the boundary arcs at the leaves. We claim that $S' := \{i \in V \mid x'_i > 0\}$ is an independent set in G. Suppose not, then there exist adjacent $u, v \in S'$. By the tree decomposition, there is a bag B_{j_1} that contains u and v. Further, there is a unique path $(S_1^{j_1}, S_2^{j_2}, S_3^{j_3}, \ldots, \emptyset^t)$ leading to the sink node $\emptyset^t$ crossing only arcs of nonzero flow. Notice that, by Lemma 24, there is a single opportunity to "forget" u and a single opportunity to "forget" v along this path, and both arcs must be taken to have $x_u > 0$ and $x_v > 0$. Moreover, u and v cannot be re-introduced along this path, since this would contradict the tree decomposition. This implies that $S_1^{j_1}$ must contain both u and v, but this contradicts the construction of N, since for every node $S^{j_1} \in N$, S is independent in G. Thus, S' is independent, so $x' = x^{S'} \in P(G)$. □

Theorem 13. *The extension complexity of a graph's independent set polytope is* $O(2^{\boldsymbol{tw}}n)$*, where* $\boldsymbol{tw}$ *denotes its treewidth.*

Proof. Lemmata 26 and 27 show that $\text{proj}_x(F_2(G)) = P(G)$. Since $F_2(G)$ has size $O(2^{\mathbf{tw}}n)$, the result follows. □

4.4 Formulation for Cardinality-Constrained Independence Systems

In this section, we study extended formulations for cardinality-constrained independence systems. Given an independence system $(I, \mathcal{I})$, define the following cardinality-constrained polytope.

$$P_k(I, \mathcal{I}) = \text{conv.hull}\left\{x^S \in \{0,1\}^{|I|} \;\middle|\; S \in \mathcal{I};\ |S| = k\right\}. \tag{4.13}$$

The extended formulation for $P_k(I, \mathcal{I})$ that we propose is based on Balas's extended formulation for the disjunction of polyhedra.

Theorem 14 (Balas [6, 7])**.** *Consider q nonempty polytopes $P^i \subseteq \mathbb{R}^n,\ i = 1, \ldots, q$ and let $P = \text{conv.hull}\left(\bigcup_{i=1}^q P^i\right)$. Then, $\text{xc}(P) = O(\sum_{i=1}^q \text{xc}(P^i))$.*

We will refer to the set $\mathcal{I}_{max}$ of (inclusion-wise) maximal independent sets, and a cardinality-constrained counterpart:

$$\mathcal{I}_{max,k} = \{S \mid S \in \mathcal{I}_{max};\ |S| \geq k\}.$$

Now, for each $S \in \mathcal{I}_{max,k}$ we will define a polytope:

$$P_k(I, \mathcal{I}, S) = \left\{ x \in [0,1]^{|I|} \;\middle|\; \sum_{i \in I} x_i = k;\ x_j = 0\ \forall j \in I \setminus S \right\}. \tag{4.14}$$

Lemma 28. $P_k(I, \mathcal{I}) = \text{conv.hull}\left(\bigcup_{S \in \mathcal{I}_{max,k}} P_k(I, \mathcal{I}, S)\right)$.

Proof. First we show the inclusion $P_k(I, \mathcal{I}) \subseteq \text{conv.hull}\left(\bigcup_{S \in \mathcal{I}_{max,k}} P_k(I, \mathcal{I}, S)\right)$. Consider a point x^S of $P_k(I, \mathcal{I})$, which we can assume, without loss of generality, is an extreme point and is thus the characteristic vector of an independent set S. Then S is a k-vertex subset of a maximal independent set S'. Hence $S' \in \mathcal{I}_{max,k}$ and

$x^S \in P_k(I, \mathcal{I}, S')$, as desired.

Now we show the inclusion $P_k(I, \mathcal{I}) \supseteq \text{conv.hull}\left(\bigcup_{S \in \mathcal{I}_{max,k}} P_k(I, \mathcal{I}, S)\right)$. Consider a point x' of conv.hull $\left(\bigcup_{S \in \mathcal{I}_{max,k}} P_k(I, \mathcal{I}, S)\right)$, which we can assume, without loss of generality, is an extreme point of polytope $P_k(I, \mathcal{I}, S)$ for some $S \in \mathcal{I}_{max,k}$. If x' is integral, then x' is the characteristic vector of a k-vertex subset of S, and any subset of S is independent, so $x' \in P_k(I, \mathcal{I})$, as desired. Now, if x' is not integral, then at least one component x'_j is fractional. Then there must be another component x'_k that is also fractional, since otherwise the sum of the x variables could not equal k. This implies an $\epsilon > 0$ such that

$$0 \leq x'_j - \epsilon < x'_j + \epsilon \leq 1$$
$$0 \leq x'_k - \epsilon < x'_k + \epsilon \leq 1.$$

Then x' can be written as a convex combination of points from $P_k(I, \mathcal{I}, S)$, namely $x' + \epsilon(e_j - e_k)$ and $x' + \epsilon(e_k - e_j)$. (The vector e_i has zeros in all entries except for a one in position i.) This contradicts that x' is an extreme point, hence x' must be integral. This concludes the proof. □

Theorem 15. $\text{xc}(P_k(I, \mathcal{I})) = O(|I||\mathcal{I}_{max,k}|)$.

Proof. Directly from Theorem 14 and Lemma 28, since $\text{xc}(P_k(I, \mathcal{I}, S)) = O(|I|)$. □

4.5 Formulation for Cardinality-Constrained Vertex Covers

In this section we provide extended formulations for the k-vertex cover polytope, based on Theorem 15 and from ideas of Chen et al. [31]. The former easily leads to a size $O(2^k n)$ bound, while the latter improves the dependence on k to $O(1.466^k n^2)$.

Definition 18. *The k-vertex cover polytope of a graph G is*

$$\begin{aligned} Q_k(G) &= \text{conv.hull}\{x^S \in \{0,1\}^n \mid |S| = k;\ S \text{ is a vertex cover for } G\} \\ &= \text{conv.hull}\left\{ x \in \{0,1\}^n \;\middle|\; \sum_{i \in V} x_i = k;\ x_i + x_j \geq 1\ \forall \{i,j\} \in E \right\}. \end{aligned}$$

The following lemma is implied by the bounded-search tree algorithm for vertex cover, c.f. [42], and has been explicitly noted by Damaschke [39]. The bound is sharp on the graph comprised of k disjoint edges.

Lemma 29. *The number of (inclusion-wise) minimal vertex covers that have cardinality $\leq k$ is at most 2^k.*

An immediate consequence is as follows.

Corollary 6. *The number of maximal independent sets of an n-vertex graph that have cardinality $\geq n - k$ is at most 2^k.*

Corollary 7. $\text{xc}(Q_k(G)) = O(2^k n)$.

Proof. Directly from Corollary 6, Theorem 15, and a change of variables. □

Now we will improve the dependence on k. In the following theorem, we say that a vertex cover C of G is *consistent* with a partition (F, D, R) of $V(G)$ if $F \subseteq C$ and $D \cap C = \emptyset$. The idea is that vertices from F are fixed in the cover, vertices from D are fixed out of the cover, and the remaining vertices from R are undetermined.

Theorem 16 (Chen et al. [31])**.** *There is an algorithm, running in time $O(1.47^k n)$, that returns a collection $\mathcal{L}(G,k)$ of triples that satisfies:*

1. $|\mathcal{L}(G,k)| \leq 1.466^k$;
2. *each $(F,D,R) \in \mathcal{L}(G,k)$ is a partition of $V(G)$;*

3. each k-vertex cover of G is consistent with exactly one triple in $\mathcal{L}(G,k)$;

4. for each $(F,D,R) \in \mathcal{L}(G,k)$, the degree of each vertex in $G[R]$ is ≤ 2.

Note that a graph with maximum degree at most 2 is the disjoint union of path and cycle graphs. Hence, $\mathbf{tw}(G[R]) \leq 2$.

Theorem 17. *For any k, $\mathrm{xc}(Q_k(G)) = O(2^{\boldsymbol{tw}(G)}n^2)$.*

Proof. The proof is long, so we provide a brief and informal sketch. The ideas are similar to those used in Section 4.3 to show that the independent set polytope of a graph has extension complexity $O(2^{\mathbf{tw}}n)$. The main change is to make $n-k+1$ 'layers' of the hypergraph's vertex set, and anytime a vertex v is forgotten in the tree decomposition, the corresponding edge that forgets v in the hypergraph should be routed to the next layer. In the last layer, the forget edges should be removed. This will ensure that the independent set has cardinality $n-k$, so the resulting vertex cover has cardinality k. The number of vertices and edges in this directed acyclic hypergraph is $O(2^{\mathbf{tw}}n(n-k)) = O(2^{\mathbf{tw}}n^2)$. Then, the machinery of [79] is used to show that a flow-based extended formulation over this hypergraph is integral. □

Lemma 30. *For any $(F,D,R) \in \mathcal{L}(G,k)$, the convex hull of k-vertex covers that are consistent with (F,D,R) has extension complexity $O(n^2)$.*

Proof. We can also assume, without loss of generality, that $S \subseteq V(G)$ is a vertex cover for G if and only if $S \cap R$ is a vertex cover for $G[R]$. Thus, we only need a polyhedral representation of the $(k-|F|)$-vertex cover polytope of $G[R]$. By Theorem 17 and the observation that $\mathbf{tw}(G[R]) \leq 2$, we have $\mathrm{xc}(Q_{k-|F|}(G[R])) = O(|R|^2) = O(n^2)$. □

Theorem 18. $\mathrm{xc}(Q_k(G)) = O(1.466^k n^2)$.

Proof. For each $(F, D, R) \in \mathcal{L}(G, k)$ create an extended formulation for k-vertex covers that are consistent with (F, D, R). Each of these polytopes has extension complexity $O(n^2)$ by Lemma 30. By Theorem 16, there are at most 1.466^k of these triples. It can then be shown that the convex hull of the union of these polytopes is precisely $Q_k(G)$, so $\text{xc}(Q_k(G)) = O(1.466^k n^2)$ by Theorem 14. □

4.6 Discussion

It should be noted that the size bounds of $O(2^{\mathbf{tw}} n)$ and $O(n+\mu)$ from Sections 4.2 and 4.3 are incomparable. For example, the number $\mu(P_n)$ of maximal independent sets of the n-vertex path graph P_n satisfies the recurrence $\mu(P_n) = \mu(P_{n-2}) + \mu(P_{n-3})$ with initial values $\mu(P_{-1}) = \mu(P_0) = \mu(P_1) = 1$, and this sequence, the Padovan sequence, grows as ρ^n, where $\rho = 1.3247\ldots$ is the plastic number [53]. This implies that the first extended formulation would use exponentially many variables, but the treewidth-based formulation would have size $O(n)$. In the other extreme, the complete graph K_n on n vertices has $\mathbf{tw}(K_n) = n - 1$, but K_n has n maximal independent sets.

5. CONCLUSION AND FUTURE WORK

In this dissertation we study challenging combinatorial optimization problems from the perspective of parameterized complexity. Below, we revisit our contributions and offer ideas for future work.

In Chapter 2, we provide fixed-parameter tractable (fpt) algorithms for the Node-Weighted Steiner Tree (NWST) problem and the Maximum-Weight Connected Subgraph (MWCS) problem. The NWST algorithm is parameterized by the number of terminals. The MWCS algorithms are parameterized by the number of positive- and negative-weight vertices. Interestingly, a complexity assumption called the Strong Exponential Time Hypothesis suggests that the MWCS algorithm parameterized by the number of negative-weight vertices is essentially best-possible.

On the other hand, we strongly suspect that the other algorithms can be improved. As noted previously, the Steiner tree algorithm due to Dreyfus and Wagner [44] runs in time $O^*(3^k)$ for instances with k terminals. A more recent paper [83] solves the Steiner tree problem in time $O^*((2+\delta)^k)$ where δ is any positive constant[1]. We suspect that their algorithm can be generalized to solve the NWST problem in roughly the same time bound. However, the analysis is more involved, and the authors state that the constants in their algorithm's runtime "become very large even for moderate δ." For these reasons, we do not attempt to improve the dependence on k for NWST here. This is a subject for future work which would improve upon our runtimes for solving both NWST and MWCS.

It may also be interesting to study the connected subgraph polytope $\mathscr{P}(G)$ for other small values of $\alpha(G)$. Our work implies that any facet-defining inequality of

[1] We can achieve a runtime of $O^*(2^k)$ using polynomial space if the edge weights are bounded [14, 85].

$\mathscr{P}(G)$ has at most $\alpha(G)$ positive coefficients. So, $\mathscr{P}(G)$ is, in some sense, relatively tame when $\alpha(G)$ is small. Another research direction is to look for extended formulations for $\mathscr{P}(G)$ whose size grows polynomially in n but exponentially in $\alpha(G)$.

In Chapter 3, we provide fpt algorithms for the maximum clique problem (parameterized by the graph's degeneracy and community degeneracy). Then we extend the approach to solve arbitrary 0-1 programs based on properties of an associated conflict graph. Roughly speaking, our algorithms are quick when the conflict graph is dense. We also examine the complexity of generating conflict edges, and show that this is difficult (conditioned on the hardness of SAT).

Our algorithms for solving 0-1 programs have runtimes that are $O^*(T_{d,m})$, where $T_{d,m}$ denotes the time to solve a subproblem of an m-constraint 0-1 program in which all but at most d variables are unfixed. By the exhaustive search algorithm, $T_{d,m} = O^*(2^d)$, and SETH would imply that this cannot really be improved. However, we note that $T_{d,m}$ can be improved for special classes of 0-1 programs where the subproblems admit nontrivial algorithms. For example, d-compatibility-degenerate set packing problems can be solved in time $O^*(2^{d/4})$ by our analysis and the independent set algorithm of [95]. (Alternatively, this is implied by our maximum clique algorithm.) The analysis provides additional theoretical evidence for the usefulness of conflict graphs in solving integer programs; the running time is bounded by a function that is exponential in the compatibility-degeneracy or the bicompatibility-degeneracy instead of in the number of 0-1 variables. These results can provide one explanation for the ability to solve problems in practice that are intractable in general—provided the associated conflict graphs are suitably dense.

There is nothing particular about the approach that limits it to linear 0-1 programs. It is just as easily applied to other mathematical optimization problems that have 0-1 variables. Similar worst-case runtimes can be achieved for mixed 0-1 linear

programs, for quadratic 0-1 programs, and even for tractable problems whose best-known algorithms have high-polynomial runtimes. The only requirement is that the subproblems have the same structure as the original problem, where subproblems are defined as having some binary variables fixed to zero or one. It is an interesting empirical question to see if our algorithms work well for these problems.

In Chapter 4, we study fpt extended formulations for various combinatorial optimization problems. Below, we summarize our contributions.

Theorem 19. *The extension complexities of the independent set polytope $P(G)$ and of the vertex cover polytope $Q(G)$ of a graph G satisfy:*

1. $\mathrm{xc}(P(G)) = \mathrm{xc}(Q(G))$;
2. $\mathrm{xc}(P(G)) = O(1.4423^n)$;
3. $\mathrm{xc}(P(G)) = O(2^{\boldsymbol{tw}}n)$.

Further, the k-vertex cover polytope $Q_k(G)$ satisfies

1. $\mathrm{xc}(Q_k(G)) = O(2^k n)$;
2. $\mathrm{xc}(Q_k(G)) = O(1.466^k n^2)$.

This improves upon the previously best bounds of $\mathrm{xc}(P(G)) = O(n^{\mathbf{tw}+1})$; $\mathrm{xc}(P(G)) = O(2^n)$; and $\mathrm{xc}(Q_k(G)) = O(n^k)$.

We suspect that several of our size bounds can be improved. The k-vertex cover problem can be solved in time $O^*(1.2738^k)$ [32], which leads one to believe that a similar size bound can be achieved. However, some of the techniques that are used to achieve the $O^*(1.2738^k)$ runtime remove feasible solutions, and may not be applicable when developing extended formulations. On the other hand, we think that it is possible that our independent set formulations based on maximal independent

sets and treewidth are optimal. To disprove this, one should find a constant $\epsilon > 0$ such that $\text{xc}(P(G))$ can be improved to size $O^*((2-\epsilon)^{\mathbf{tw}})$ or to size $O((\sqrt[3]{3}-\epsilon)^n)$.

We suspect that the following is true. Here, $P_k(I, \mathcal{I})$ is the polytope corresponding to the (cardinality constrained) independence system $(I, \mathcal{I})$.

Conjecture 1. $\text{xc}(P_k(I, \mathcal{I})) = O(|I| + |\mathcal{I}_{max,k}|)$.

The formulation below is our prime candidate to prove it. Introduce a variable y_S for each $S \in \mathcal{I}_{max,k}$, where $\mathcal{I}_{max,k}$ is the set of inclusion-wise maximal independent sets that have cardinality at least k. The formulation is the set of $(x, y) \geq 0$ satisfying:

$$\sum_{i \in I} x_i = k \tag{5.1}$$

$$\sum_{S \in \mathcal{I}_{max,k}} y_S = 1 \tag{5.2}$$

$$x_i - \sum_{S \in \mathcal{I}_{max,k} : i \in S} y_S \leq 0, \ \forall i \in I. \tag{5.3}$$

It would be interesting to find nontrivial lower bounds for $\text{xc}(Q_k(G))$, specifically to study its extension complexity as a function of k. It would also be interesting to investigate lower and upper bounds for the cardinality-constrained independent set polytope. Due to the widely held belief that the independent set problem is not fpt with respect to solution size, this makes us think that fpt extended formulations are unachievable. However, we know of no complexity-theoretic justification for the belief that they do not exist (only that they cannot be constructed efficiently).

REFERENCES

[1] T. Achterberg. Conflict analysis in mixed integer programming. *Discrete Optimization*, 4(1):4–20, 2007.

[2] E. Álvarez-Miranda, I. Ljubić, and P. Mutzel. The maximum weight connected subgraph problem. In M. Jünger and G. Reinelt, editors, *Facets of Combinatorial Optimization*, pages 245–270. Springer, Berlin, 2013.

[3] B. Aspvall, M.F. Plass, and R.E. Tarjan. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Information Processing Letters*, 8(3):121–123, 1979.

[4] A. Atamtürk, G.L. Nemhauser, and M.W.P. Savelsbergh. Conflict graphs in solving integer programming problems. *European Journal of Operational Research*, 121(1):40–55, 2000.

[5] D. A. Bader, H. Meyerhenke, P. Sanders, and D. Wagner, editors. *Graph partitioning and graph clustering*, volume 588 of *Contemporary Mathematics*. American Mathematical Society, 2013.

[6] E. Balas. Disjunctive programming and a hierarchy of relaxations for discrete optimization problems. *SIAM Journal on Algebraic Discrete Methods*, 6(3):466–486, 1985.

[7] E. Balas. Disjunctive programming: Properties of the convex hull of feasible points. *Discrete Applied Mathematics*, 89(1):3–44, 1998.

[8] F. Barahona and A.R. Mahjoub. Compositions of graphs and polyhedra II: stable sets. *SIAM Journal on Discrete Mathematics*, 7(3):359–371, 1994.

[9] A. Bazzi, S. Fiorini, S. Pokutta, and O. Svensson. No small linear program approximates vertex cover with $2 - \epsilon$. *arXiv preprint arXiv:1503.00753*, 2015.

[10] G. Bianconi and M. Marsili. Emergence of large cliques in random scale-free networks. *Europhysics Letters*, 74(4):740, 2006.

[11] D. Bienstock and G. Munoz. On optimization problems with bounded tree-width. *arXiv preprint arXiv:1501.00288*, 2015.

[12] D. Bienstock and N. Ozbay. Tree-width and the Sherali–Adams operator. *Discrete Optimization*, 1(1):13–21, 2004.

[13] R. Bixby, S. Ceria, C. McZeal, and M. Savelsbergh. An updated mixed integer programming library: MIPLIB 3.0, 1996.

[14] A. Björklund, T. Husfeldt, P. Kaski, and M. Koivisto. Fourier meets möbius: fast subset convolution. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 67–74. ACM, 2007.

[15] H.L. Bodlaender. Treewidth: Algorithmic techniques and results. In *Mathematical Foundations of Computer Science 1997: 22nd International Symposium, MFCS'97, Bratislava, Slovakia, August 25-29, 1997, Proceedings*, volume 22, page 19. Springer, 1997.

[16] H.L. Bodlaender. A partial-k-arboretum of graphs with bounded treewidth. *Theoretical Computer Science*, 209(1):1–45, 1998.

[17] H.L. Bodlaender and A.M.C.A. Koster. Combinatorial optimization on graphs of bounded treewidth. *The Computer Journal*, 51(3):255–269, 2008.

[18] V. Boginski, S. Butenko, and P.M. Pardalos. On structural properties of the market graph. In A. Nagurney, editor, *Innovations in financial and economic networks*, pages 29–45. Edward Elgar Publishing, 2003.

[19] I.M. Bomze, M. Budinich, P.M. Pardalos, and M. Pelillo. The maximum clique problem. In D.-Z. Du and P.M. Pardalos, editors, *Handbook of combinatorial optimization*, pages 1–74. Springer, 1999.

[20] N. Bourgeois, B. Escoffier, V.T. Paschos, and J.M.M. van Rooij. Fast algorithms for max independent set. *Algorithmica*, 62(1-2):382–415, 2012.

[21] G. Braun, S. Fiorini, and S. Pokutta. Average case polyhedral complexity of the maximum stable set problem. In *RANDOM 2014*, volume 28 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 515–530, 2014.

[22] M. Braverman and A. Moitra. An information complexity approach to extended formulations. In *Proceedings of the 45th Annual ACM Symposium on Theory of Computing*, pages 161–170. ACM, 2013.

[23] C. Bron and J. Kerbosch. Algorithm 457: finding all cliques of an undirected graph. *Communications of the ACM*, 16(9):575–577, 1973.

[24] A. Buchanan and S. Butenko. Tight extended formulations for independent set. 2014. Working paper available at `http://www.optimization-online.org/DB_FILE/2014/09/4540.pdf`.

[25] A. Buchanan, J.L. Walteros, S. Butenko, and P.M. Pardalos. Solving integer programs with dense conflict graphs. In L.G. Casado, I. García, and E.M.T. Hendrix, editors, *Proceedings of the XII Global Optimization Workshop*, pages 125–128, 2014.

[26] A. Buchanan, J.L. Walteros, S. Butenko, and P.M. Pardalos. Solving maximum clique in sparse graphs: an $O(nm + n2^{d/4})$ algorithm for d-degenerate graphs. *Optimization Letters*, 8(5):1611–1617, 2014.

[27] A. Buchanan, Y. Wang, and S. Butenko. Exact algorithms for node-weighted Steiner tree and maximum-weight connected subgraph. 2015. Manuscript.

[28] S. Butenko and W.E. Wilhelm. Clique-detection models in computational biochemistry and genomics. *European Journal of Operational Research*, 173(1):1–17, 2006.

[29] C. Calabro, R. Impagliazzo, and R. Paturi. The complexity of satisfiability of small depth circuits. In J. Chen and F.V. Fomin, editors, *Parameterized and Exact Computation*, pages 75–85. Springer, 2009.

[30] J. Chen, X. Huang, I.A. Kanj, and G. Xia. Strong computational lower bounds via parameterized complexity. *Journal of Computer and System Sciences*, 72(8):1346–1367, 2006.

[31] J. Chen, I.A. Kanj, J. Meng, G. Xia, and F. Zhang. Parameterized top-K algorithms. *Theoretical Computer Science*, 470:105–119, 2013.

[32] J. Chen, I.A. Kanj, and G. Xia. Improved upper bounds for vertex cover. *Theoretical Computer Science*, 411(40):3736–3756, 2010.

[33] F. Chung. Graph theory in the information age. *Notices of the AMS*, 57(6):726–732, 2010.

[34] M. Conforti, G. Cornuéjols, and G. Zambelli. Extended formulations in combinatorial optimization. *Annals of Operations Research*, 204(1):97–143, 2013.

[35] M. Conforti, G. Cornuéjols, and G. Zambelli. *Integer Programming*, volume 271 of *Graduate Texts in Mathematics*. Springer International Publishing, 2014.

[36] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to algorithms*. The MIT Press, 3rd edition, 2009.

[37] N. Creignou and J.-J. Hébrard. On generating all solutions of generalized satisfiability problems. *Informatique théorique et applications*, 31(6):499–511, 1997.

[38] M. Cygan, H. Dell, D. Lokshtanov, D. Marx, J. Nederlof, Y. Okamoto, R. Paturi, S. Saurabh, and M. Wahlstrom. On problems as hard as CNF-SAT. In *Computational Complexity (CCC), 2012 IEEE 27th Annual Conference on*, pages 74–84. IEEE, 2012.

[39] P. Damaschke. Parameterized enumeration, transversals, and imperfect phylogeny reconstruction. *Theoretical Computer Science*, 351(3):337–350, 2006.

[40] R. Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer, Heidelberg, fourth edition, 2010.

[41] M.T. Dittrich, G.W. Klau, A. Rosenwald, T. Dandekar, and T. Müller. Identifying functional modules in protein–protein interaction networks: an integrated exact approach. *Bioinformatics*, 24(13):i223–i231, 2008.

[42] R.G. Downey and M.R. Fellows. *Parameterized Complexity*. Springer, 1999.

[43] R.G. Downey and M.R. Fellows. *Fundamentals of Parameterized complexity*. Springer, 2013.

[44] S.E. Dreyfus and R.A. Wagner. The Steiner problem in graphs. *Networks*, 1(3):195–207, 1971.

[45] J. Edmonds. Matroids and the greedy algorithm. *Mathematical Programming*, 1(1):127–136, 1971.

[46] J. Edmonds. An interview with Jack Edmonds. *Optima*, 97:9–13, 2015. Interview conducted by V. Kaibel, J. Lee, and J. Linderoth.

[47] D. Eppstein, M. Löffler, and D. Strash. Listing all maximal cliques in sparse graphs in near-optimal time. In O. Cheong, K.-Y. Chwa, and K. Park, editors, *Algorithms and Computation*, volume 6506 of *Lecture Notes in Computer Science*, pages 403–414. Springer, 2010.

[48] D. Eppstein and D. Strash. Listing all maximal cliques in large sparse real-world graphs. In P.M. Pardalos and S. Rebennack, editors, *Experimental Algorithms*, volume 6630 of *Lecture Notes in Computer Science*, pages 364–375. Springer, 2011.

[49] S. Fiorini, S. Massar, S. Pokutta, H.R. Tiwary, and R. de Wolf. Linear vs. semidefinite extended formulations: exponential separation and strong lower bounds. In *Proceedings of the 44th Annual ACM Symposium on Theory of Computing*, pages 95–106. ACM, 2012.

[50] J. Flum and M. Grohe. *Parameterized complexity theory*. Texts in Theoretical Computer Science. Springer-Verlag Berlin Heidelberg, 2006.

[51] F.V. Fomin, F. Grandoni, A.V. Pyatkin, and A.A. Stepanov. Combinatorial bounds via measure and conquer: Bounding minimal dominating sets and applications. *ACM Transactions on Algorithms*, 5(1):9, 2008.

[52] F.V. Fomin and D. Kratsch. *Exact exponential algorithms*. Springer, 2010.

[53] Z. Füredi. The number of maximal independent sets in connected graphs. *Journal of Graph Theory*, 11(4):463–470, 1987.

[54] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, 1979.

[55] O. Goldreich. *Computational Complexity: A Conceptual Perspective*. Cambridge University Press, 2008.

[56] M. Grötschel, L. Lovász, and A. Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1(2):169–197, 1981.

[57] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*, volume 2 of *Algorithms and Combinatorics*. Springer-Verlag, Berlin, 2nd edition, 1993.

[58] P.L. Hammer, E.L. Johnson, and U.N. Peled. Facet of regular 0–1 polytopes. *Mathematical Programming*, 8(1):179–206, 1975.

[59] J. Håstad. Clique is hard to approximate within $n^{1-\varepsilon}$. *Acta Mathematica*, 182(1):105–142, 1999.

[60] F.K. Hwang, D.S. Richards, and P. Winter. *The Steiner tree problem*, volume 53 of *Annals of Discrete Mathematics*. Elsevier, 1992.

[61] R. Impagliazzo and R. Paturi. Complexity of k-SAT. In *Proceedings of the Fourteenth Annual IEEE Conference on Computational Complexity*, pages 237–240. IEEE, 1999.

[62] R. Impagliazzo, R. Paturi, and S. Schneider. A satisfiability algorithm for sparse depth two threshold circuits. In *Foundations of Computer Science (FOCS), 2013 IEEE 54th Annual Symposium on*, pages 479–488. IEEE, 2013.

[63] R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001.

[64] D.S. Johnson. The NP-completeness column: an ongoing guide. *Journal of Algorithms*, 6(1):145–159, 1985.

[65] D.S. Johnson and M.A. Trick, editors. *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, Workshop, October 11-13, 1993*,

volume 26 of *Discrete Mathematics and Theoretical Computer Science*. AMS, 1996.

[66] V. Kaibel. Extended formulations in combinatorial optimization. *Optima*, 85:2–7, 2011.

[67] M.M. Kanté, V. Limouzy, A. Mary, and L. Nourine. Enumeration of minimal dominating sets and variants. In *Fundamentals of Computation Theory*, pages 298–309. Springer, 2011.

[68] R.M. Karp. Reducibility among combinatorial problems. In R.E. Miller and J.W. Thatcher, editors, *Complexity of Computer Computations*. Plenum Press, 1972.

[69] T. Kloks. Treewidth: Computations and approximations. *Lecture Notes in Computer Science*, 842, 1994.

[70] P. Kolman and M. Koutecký. Extended formulation for CSP that is compact for instances of bounded treewidth. *arXiv preprint arXiv:1502.05361*, 2015.

[71] E.L. Lawler. *Combinatorial optimization: networks and matroids*. Courier Corporation, 1976.

[72] E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan. Generating all maximal independent sets: NP-hardness and polynomial-time algorithms. *SIAM Journal on Computing*, 9(3):558–565, 1980.

[73] J.K. Lenstra. The mystical power of twoness: in memoriam Eugene L. Lawler. *Journal of Scheduling*, 1(1):3–14, 1998.

[74] D.R. Lick and A.T. White. k-degenerate graphs. *Canad. J. Math*, 22:1082–1096, 1970.

[75] D. Lokshtanov, D. Marx, and S. Saurabh. Known algorithms on graphs of bounded treewidth are probably optimal. In *Proceedings of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 777–789. SIAM, 2011.

[76] D. Lokshtanov, D. Marx, S. Saurabh, et al. Lower bounds based on the exponential time hypothesis. *Bulletin of the EATCS*, (105):41–72, 2011.

[77] A.O. Makhorin. GLPK (GNU linear programming kit) version 4.52, 2013. `http://www.gnu.org/software/glpk/`.

[78] R.K. Martin. Using separation algorithms to generate mixed integer model reformulations. *Operations Research Letters*, 10(3):119–128, 1991.

[79] R.K. Martin, R.L. Rardin, and B.A. Campbell. Polyhedral characterization of discrete dynamic programming. *Operations Research*, 38(1):127–138, 1990.

[80] D. W. Matula and L. L. Beck. Smallest-last ordering and clustering and graph coloring algorithms. *Journal of the ACM*, 30(3):417–427, 1983.

[81] R.R. Meyer. On the existence of optimal solutions to integer and mixed-integer programming problems. *Mathematical Programming*, 7(1):223–235, 1974.

[82] R.E. Miller and D.E. Muller. A problem of maximum consistent subsets. *IBM Research Rep. RC-240, IBM Research Center, Yorktown Heights, New York, USA*, 1960.

[83] D. Mölle, S. Richter, and P. Rossmanith. A faster algorithm for the Steiner tree problem. In B. Durand and W. Thomas, editors, *STACS 2006*, volume 3884 of *Lecture Notes in Computer Science*, pages 561–570. Springer, 2006.

[84] J.W. Moon and L. Moser. On cliques in graphs. *Israel Journal of Mathematics*, 3(1):23–28, 1965.

[85] J. Nederlof. Fast polynomial-space algorithms using M obius inversion: Improving on Steiner tree and related problems. In S. Albers, A. Marchetti-Spaccamela, Y. Matias, S. Nikoletseas, and W. Thomas, editors, *Automata, Languages and Programming*, volume 5555 of *Lecture Notes in Computer Science*, pages 713–725. Springer Berlin Heidelberg, 2009.

[86] G.L. Nemhauser and L.A. Wolsey. *Integer and Combinatorial Optimization*, volume 18 of *Wiley-Interscience Series in Discrete Mathematics and Optimization*. Wiley, New York, 1988.

[87] J. Nešetřil and S. Poljak. On the complexity of the subgraph problem. *Commentationes Mathematicae Universitatis Carolinae*, 26(2):415–419, 1985.

[88] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*, volume 31 of *Oxford Lecture Series in Mathematics and Its Applications*. Oxford University Press, 2006.

[89] M.W. Padberg. On the facial structure of set packing polyhedra. *Mathematical Programming*, 5(1):199–215, 1973.

[90] C. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 229–234. ACM, 1988.

[91] C.H. Papadimitriou. *Computational complexity*. Addison-Wesley Publishing Company, 1995.

[92] C.H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Dover Publications, Mineola, 1998.

[93] M. Pătraşcu and R. Williams. On the possibility of faster SAT algorithms. In *SODA*, volume 10, pages 1065–1075. SIAM, 2010.

[94] S. Pokutta and M. Van Vyve. A note on the extension complexity of the knapsack polytope. *Operations Research Letters*, 41(4):347–350, 2013.

[95] J. M. Robson. Finding a maximum independent set in time $O(2^{n/4})$. 2001. LaBRI, Université de Bordeaux I. Available online at `https://www.labri.fr/perso/robson/mis/techrep.html`.

[96] M. Rospocher. *On the computational complexity of enumerating certificates of NP problems*. PhD thesis, University of Trento, 2006.

[97] T. Rothvoß. The matching polytope has exponential extension complexity. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*, pages 263–272. ACM, 2014.

[98] D.S. Rubin. On the unlimited number of faces in integer hulls of linear programs with a single constraint. *Operations Research*, 18(5):940–946, 1970.

[99] M.W.P. Savelsbergh. Preprocessing and probing techniques for mixed integer programming problems. *ORSA Journal on Computing*, 6(4):445–454, 1994.

[100] A. Schrijver. *Theory of linear and integer programming*. Wiley-Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons, 1998.

[101] A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*, volume 24 of *Algorithms and Combinatorics*. Springer, 2003.

[102] A. Segev. The node-weighted Steiner tree problem. *Networks*, 17(1):1–17, 1987.

[103] H.D. Sherali and W.P. Adams. A hierarchy of relaxations between the continuous and convex hull representations for zero-one programming problems. *SIAM Journal on Discrete Mathematics*, 3(3):411–430, 1990.

[104] E. Tomita, A. Tanaka, and H. Takahashi. The worst-case time complexity for generating all maximal cliques and computational experiments. *Theoretical Computer Science*, 363(1):28–42, 2006.

[105] S. Tsukiyama, M. Ide, H. Ariyoshi, and I. Shirakawa. A new algorithm for generating all the maximal independent sets. *SIAM Journal on Computing*, 6(3):505–517, 1977.

[106] A. Vergis. Manuscript, 1983.

[107] A. Verma, A. Buchanan, and S. Butenko. Solving the maximum clique and vertex coloring problems on very large sparse networks. *INFORMS Journal on Computing*, 27(1):164–177, 2015.

[108] Y. Wang, A. Buchanan, and S. Butenko. On imposing connectivity constraints in integer programs. Working paper available at `http://www.optimization-online.org/DB_FILE/2015/02/4768.pdf`, 2015.

[109] R. Williams. Algorithms for circuits and circuits for algorithms. In *28th IEEE Conference on Computational Complexity (CCC 2014)*, 2014.

[110] L.A. Wolsey. *Integer programming*, volume 42 of *Wiley-Interscience Series in Discrete Mathematics and Optimization*. John Wiley & Sons, 1998.

[111] D.R. Wood. On the maximum number of cliques in a graph. *Graphs and Combinatorics*, 23(3):337–352, 2007.

[112] M. Yannakakis. Expressing combinatorial optimization problems by linear programs. *Journal of Computer and System Sciences*, 43:441–466, 1991.

[113] D. Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing*, pages 681–690. ACM, 2006.