FACULTY OF INFORMATION TECHNOLOGY CTU IN PRAGUE

ASSIGNMENT OF BACHELOR'S THESIS

Title:	Kernels for the Max Cut problem
Student:	František Koutenský
Supervisor:	RNDr. Dušan Knop, Ph.D.
Study Programme:	Informatics
Study Branch:	Computer Science
Department:	Department of Theoretical Computer Science
Validity:	Until the end of winter semester 2021/22

Instructions

MaxCut, a well-known NP-hard problem, is the following. We are given a graph G and a positive integer k. The task is to determine if there is a cut of size at least k in G or not. Here, a cut is defined by a two-partition (U,W) of the vertex set of G, denoted V(G), i.e., the union of U and W is V(G) and they do not intersect. The size of a cut is the number of edges in G with one endpoint in U and the other endpoint in W.

It is known that MaxCut is fixed-parameter tractable when parameterized (FPT) by the treewidth (TW) of the input graph G. Furthermore, if a problem is FPT, then it admits the so-called kernel---roughly speaking, a polynomial-time algorithm (in the size of G) that returns an equivalent instance of MaxCut (i.e., a graph H and an integer z) such that size of H can be bounded in terms of the parameter (in our case treewidth). Is it possible to prove that the size of H is in fact polynomial in the vertex cover number (note that this is very unlikely for TW)?

References

[1] Hans L. Bodlaender, Klaus Jansen: On the Complexity of the Maximum Cut Problem. Nord. J. Comput. 7(1): 14-31 (2000)

[2] Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, Saket Saurabh: Parameterized Algorithms. Springer 2015, ISBN 978-3-319-21274-6, pp. 3-555

[3] Garey, M. R.; Johnson, D. S. (1979). Victor Klee (ed.). Computers and Intractability: A Guide to the Theory of NP-Completeness. A Series of Books in the Mathematical Sciences. San Francisco, Calif.: W. H. Freeman and Co. pp. x+338. ISBN 0-7167-1045-5. MR 0519066.

[4] Fomin, F., Lokshtanov, D., Saurabh, S., & Zehavi, M. (2019). Kernelization: Theory of Parameterized Preprocessing. Cambridge: Cambridge University Press. doi:10.1017/9781107415157

doc. Ing. Jan Janoušek, Ph.D.
Head of Department

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
Dean

Prague April 19, 2020

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Bachelor's thesis

Kernels for the Max Cut problem

František Koutenský

Department of Theoretical Computer Science
Supervisor: RNDr. Dušan Knop, Ph.D.

June 4, 2020

Acknowledgements

I would like to express my deepest appreciation to Dr. Dušan Knop for his persistent support in writing this thesis, for providing invaluable insight into this topic and for his helpful advice that helped me a lot. Without his guidance this thesis would not have been completed.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60 (1) of the Act.

In Prague on June 4, 2020 . . . . . .

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Koutenský, František. *Kernels for the Max Cut problem.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2020.

Abstrakt

Tato práce shrnuje dosavadní výzkum problému Max Cut a představuje nový kernelizační algoritmus pro Simple Max Cut problém parametrizovaný velikostí minimálního vrcholového pokrytí vstupního grafu G omezující velikost jádra funkcí $O(\mathrm{vc}(G)^4)$, kde $\mathrm{vc}(G)$ označuje velikost minimálního vrcholového pokrytí grafu G.

Klíčová slova maximální řez, kernelizace, parametrizovaná složitost, FPT algoritmy

Abstract

This thesis summarizes known results of the Max Cut problem and introduces a new kernelization algorithm for the Simple Max Cut problem parameterized by vertex cover number of the input graph G bounding the size of the kernel by $O(\mathrm{vc}(G)^4)$, where $\mathrm{vc}(G)$ denotes vertex cover number of G.

Keywords maximum cut, kernelization, parameterized complexity, FPT algorithms

Contents

1 Introduction **1**
1.1 Parameterized complexity 3
1.2 Kernelization 4
1.3 Motivation 5
1.4 Contribution 6

2 Review **7**
2.1 Applications 7
2.2 Tractability 7
2.3 Special classes of graphs 8
2.4 Approximation algorithms 11
2.5 FPT algorithms 12

3 Current work **13**
3.1 Preliminaries 13
3.2 Polynomial kernel for Max Cut parameterized by vertex cover number 15

Conclusion **21**

Bibliography **23**

Chapter 1

Introduction

Partitioning graph vertices into j-disjoint subsets is one of the most fundamental concepts in graph theory. It can be represented as a problem of finding a j-cut in an undirected (edge) weighted graph G. Roughly speaking, a *j-cut* is a set of edges whose removal from G leaves G with j disjoint subset of vertices, such that there is no edge having endvertices in different subsets. There are two natural problems associated with graph partitioning. The Min j-Cut problem asks for the total weight of a j-cut to be as small as possible, while the Max j-Cut seeks the total weight of a j-cut to be as large as possible. Although the Min j-Cut problem is NP-complete when j is part of the input, it becomes polynomial solvable when j is fixed [1]. Furthermore, the Min 2-Cut problem (known as Min Cut) can be solved in a very elegant way using the *max flow min cut* theorem that was presented in the famous work of Ford and Fulkerson [2]. On the other hand, even the Max 2-Cut problem (known as Max Cut) was proven to be NP-complete by Karp [3] and was included in Karp's 21 NP-complete problems. Moreover, Garey, Johnson, and Stockmeyer [4] proved that the Max Cut problem is NP-complete even for unweighted graphs, which corresponds to the Max Cut problem where weights of all edges are one. The Max Cut problem for unweighted graphs is known as the Simple Max Cut problem.

Unlike the Min Cut problem, the Max Cut problem is surprisingly hard to solve. However, due to its importance in many industries, mainly in very-large-scale integration (VLSI) circuit design [5, 6], statistical physics [5, 7] and data clustering [8], the algorithmic perspective of Max Cut is still actively investigated.

Let us define the problem in a more formal way. In this chapter, a graph is always an undirected and (edge) weighted. A *partition* of a graph G is a pair (U, W), where $U \subset V(G)$ and $W \subset V(G)$ are nonempty subsets of vertices, such that $U \cup W = V(G)$ and $U \cap W = \emptyset$. A *cut* in a graph G with a partition (U, W) is a subset of edges $E(U, W)$.

Max Cut	
Input:	An undirected (edge) weighted graph G and a positive integer ℓ.
Question:	Is there a cut C in G, such that the total weight of C is at least ℓ?

Since it is highly unlikely that $\mathsf{P} = \mathsf{NP}$, there is little hope for polynomial algorithm for any NP-complete problem. But these "hard" problems often have many applications in practise and there is an effort to solve them as effectively as possible.

One way of dealing with "hard" problems represent algorithms that do not guarantee a good performance for all possible instances of a problem (worst-case), but only for many of them. These algorithms are usable wherever they are efficient for instances that are most common in practise. Such an algorithm for Max Cut was provided by Rendl, Rinaldi, and Wiegele [9]. Their algorithm uses the *branch and bound* method and is based on the basic semidefinite relaxation of the Max-Cut problem. They claim that their algorithm works in a reasonable time for any instance of up to 100 vertices.

When the optimal solution is not chased, the powerful tool for tackling NP-complete problems represents so-called approximation algorithms. Formally, for some constant c, a c-approximation algorithm for a maximization problem is an algorithm that returns at least c times the optimal value for a given instance. The running time of an approximation algorithm is required to be polynomial, in the context of NP-complete problems. For the Max Cut problem, Goemans and Williamson [10] came with a famous randomized approximation algorithm based on semidefinite programming guaranteeing the expected value of a solution to be at least 0.878 times the optimal value. Moreover, Khot et al. [11] showed that if the *unique games conjecture* holds, then the algorithm of Goemans and Williamson is optimal. The concept of approximation algorithms was introduced by Garey, Graham, and Ullman [12] and Johnson [13].

Another method is to restrict the input of a problem and find a polynomial time algorithm for that restriction. The Max Cut problem, like other classical NP-complete problems, is actively studied when restricted to various graph classes. Perhaps one of the most significant results is the polynomial time algorithm for planar graphs invented independently by Hadlock [14] and Dorfman and Orlova [15]. Another important outcome is the polynomial time algorithm for Max Cut restricted on graphs with bounded treewidth designed by Wimer [16]. Furthermore, Bodlaender showed polynomial algorithm for Simple Max Cut for cographs [17]. On the other hand, Max Cut remains NP-complete on graphs with maximum degree 3 [18], chordal graphs [17], 2-split graphs [17], and tripartite graphs [17]. We provide a comprehensive overview of studies of the Max Cut problem with restricted input in chapter 2.

1.1 Parameterized complexity

Unless P = NP, it is obvious that polynomial time algorithm for any NP-complete problem does not exist. However, parameterized complexity describes the running time of algorithms by a function that does not depend only on the input size n, but also on some parameter k. The goal is to bound the computational time of the "hard" part of the input instance by the parameter k, while the other part that depends on n can be solved efficiently. Parameterized complexity, in the context of NP-complete problems, looks for fixed-parameter tractable (FPT) algorithms, which are algorithms running in time $f(k) \cdot n^c$, where $f\colon \mathbb{N} \to \mathbb{N}$ is an arbitrary computable function, k is a parameter, and c is a constant independent of both n and k. Straightforwardly, if k is bounded by a constant, then the algorithm is in fact polynomial and the problem becomes tractable.

Let us formalize basic concepts of parameterized complexity. Note that our notation mostly coincides with that of Cygan et al. [19]. In what follows, Σ is always a finite alphabet and the set of all strings over Σ is denoted Σ^*.

Definition 1 (Parameterized problem)**.** *A* parameterized problem *is a language $L \subseteq \Sigma^* \times \mathbb{N}$. For an instance $(I, k) \in \Sigma^* \times \mathbb{N}$, k is called the* parameter*.*

Definition 2 (Fixed-parameter tractability)**.** *A parameterized problem $L \subseteq \Sigma^* \times \mathbb{N}$ is* fixed-parameter tractable *(FPT), if there is an algorithm that correctly decides whether an instance $(I, k) \in \Sigma^* \times \mathbb{N}$ belongs to L in time bounded by $f(k) \cdot n^c$, where $f\colon \mathbb{N} \to \mathbb{N}$ is a computable function, n is the size of I, and c is a constant independent from both k and n.*

The key part of FPT algorithm design is the choice of the parameter. The parameter can be either explicitly given, or a measure of some property of the input instance (called a *structural* parameter). For example, an explicitly given parameter p for Max Cut might be the minimum size of cut partitions, that is, for a graph G, a cut C in G with a partition $(A, V(G) \setminus A)$ is not valid unless the size of both A and $V(G) \setminus A$ is at least p. A structural parameter for a graph G can be, for example, the maximum degree of G. Squarely, if a problem is FPT, then it is with respect to the parameter by which the problem is parameterized. In other words, for some parameters k and p, a problem can be in FPT when parameterized by k, while not in FPT when parameterized by p. Moreover, if the parameter k is the input size n, the complexity class FPT is degraded to the complexity class NP. On the other hand, if $k = 1$, the complexity class FPT is equal to the complexity class P.

A powerful structural parameter is treewidth of a graph, discovered by Robertson and Seymour [20]. Roughly speaking, treewidth measures "tree-likeness" of a graph (the formal definition will be given in Chapter 2). Many famous problems were shown to be FPT when parameterized by treewidth; these problems include, for example, Hamiltonian Circuit, Stable Set,

and Vertex Cover [21]. Moreover, Courcelle [22] achieved a very strong result by showing that every problem expressible in *monadic second order logic* is in FPT when parameterized by treewidth. Treewidth parameterization has applications wherever input graphs have relatively small treewidth; some of these applications are shown in surveys of Bodlaender [23, 24, 21], Reed [25] and Kloks [26].

The foundation of parameterized comlexity was built by Downey and Fellows in several papers [27, 28, 29]. The classical reference to parameterized complexity are textbooks [30] and [31] of Downey and Fellows. Another pioneer works in parameterized comlexity was provided by Niedermeier [32], Flum and Grohe [33], and Cygan et al. [19].

1.2 Kernelization

An important method of designing FPT algorithms is the *data preprocessing* or the so-called *kernelization*. The aim of kernelization is, roughly speaking, to "reduce" data of the input instance so that it contains only the "hardest" part or the "kernel" of the input instance. Kernelization of a problem is realized by a *kernelization algorithm*, which has the following defition. Here, the notation of most concepts matches with that of Fomin et al. [34].

Definition 3 (Kernelization algorithm)**.** *A* kernelization algorithm *for a parameterized problem $L \subseteq \Sigma^* \times \mathbb{N}$ is an algorithm that, for an instance $(I, k) \in \Sigma^* \times \mathbb{N}$, runs in time $O(|I|^c)$, where c is a constant, returns an instance $(I', k') \in \Sigma^* \times \mathbb{N}$, and the following properties hold:*

- *$(I, k) \in L$ if and only if $(I', k') \in L$,*
- *$k' \leq k$,*
- *$|I'| \leq g(k)$, for some function $g \colon \mathbb{N} \to \mathbb{N}$.*

A kernel *(of a kernelization algorithm) refers to the output instance. The function g of a kernelization algorithm refers to the size of the* kernel*. The* kernel *is polynomial if g is a polynomial function.*

The most common approach of constructing kernelization algorithms is to design a set of reduction rules [34]. Roughly speaking, a *reduction rule* is a rule that transforms a problem instance into an equivalent instance with smaller size or other benefits. Here, two input instances (I, k) and (I', k') of a parameterized problem L are equivalent if (I, k) belongs to L if and only if (I', k') belongs to L. Once there is a set of reduction rules, a kernelization algorithm exhaustively applies all of them, typically in a predefined order. Reduction rule is defined as follows.

Definition 4 (Reduction rule)**.** *A* reduction rule *for a parameterized problem* $L \subseteq \Sigma^* \times \mathbb{N}$ *is a function* $\varphi\colon \Sigma^* \times \mathbb{N} \to \Sigma^* \times \mathbb{N}$*, such that, for an instance* $(I, k) \in \Sigma^* \times \mathbb{N}$*, the function* φ *is computable in polynomial time with respect to both* k *and the size of* I*, and returns an equivalent instance* (I', k')*.*

The definition of reduction rule is relatively open. However, we often look for reduction rules that reduce the input size or the parameter.

The tight connection between kernelization and parameterized complexity represents the fact that if there is a kernelization algorithm K for a decidable parameterized problem L, then clearly there is an FPT algorithm for L; a simple FPT algorithm for L can be the one that solves the kernel of K by brute-force. What is more, the opposite direction of the claim, that is, if there is an FPT algorithm for L, then there is also a kernelization algorithm for L, is easy to prove as well. However, it was shown by Bodlaender et al. [35] that even if there is a kernelization algorithm for an FPT problem, it does not imply that the problem admits polynomial size kernel.

1.3 Motivation

It is well known that Max Cut, like many other NP-complete problems, is in FPT when parameterized by treewidth [16]. It was unknown whether Max Cut parameterized by treewidth admits a polynomial kernel, until Bodlaender et al. [35] came with a framework for determining whether problems have polynomial kernels or not. From their work, it is not difficult to see that both the Max Cut problem and even the Simple Max Cut problem parameterized by treewidth do not admit a polynomial kernel, unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$. Also, Lokshtanov et al. [36] showed that Max Cut (together with some other problems) parameterized by treewidth does not admit an FPT algorithm with a better time than $2^{\mathrm{tw}(G)} \cdot n^{O(1)}$, where $\mathrm{tw}(G)$ is treewidth of a given graph G, unless the *strong exponential time hypothesis* [37, 38] fails.

This fact brings a motivation to study another parameters. One of them is the vertex cover number proposed to study by Fellows et al. [39]. Here, a *vertex cover* of a graph G is a subset of vertices $X \subseteq V(G)$, such that at least one endvertex of every edge of G belongs to X. A *minimum vertex cover* of a graph G is a vertex cover of G of the smallest possible size. A *vertex cover number* of a graph G is the size of a minimum vertex cover of G, denoted $\mathrm{vc}(G)$. It is easy to show that, for any graph, treewidth never exceed vertex cover number. It makes vertex cover number a "stronger" parameter than treewidth, since bounded vertex cover number implies bounded treewidth. What is more, a vertex cover of any graph of size at most two times greater than vertex cover number of the graph can be easily obtained by a simple approximation algorithm. The algorithm first finds a maximal matching of a graph G, which can be done by a greedy algorithm in linear time with respect to the edge set size of G, and then returns all endvertices of that matching as

a vertex cover of G. This algorithm was discovered independently by Gavril and Yannakakis [40, p. 134].

While it is very unlikely that there is a polynomial kernel for Simple Max Cut parameterized by treewidth, it is interesting to ask if the problem admits a polynomial kernel when parameterized by other parameters. The vertex cover number represents a very suitable candidate.

1.4 Contribution

This thesis presents a new kernelization algorithm for the Simple Max Cut problem parameterized by vertex cover number with a polynomial kernel size as proof of Theorem 1.

Theorem 1. *Simple Max Cut admits a kernel with $O(\mathrm{vc}(G)^4)$ vertices, where G is the input graph.*

The proof of Theorem 1 and the description of the algorithm is presented in Chapter 3. In chapter 2, we provide a comprehensive overview of the Max Cut problem.

Chapter 2

Review

2.1 Applications

The Max Cut problem has applications in various industries. In VLSI circuit desing, the important problem is to minimize the number of vertical interconnection areas, which can be represented as the Max Cut problem [5, 6]. Another application for Max Cut can be found in statistical physics for the analysis of spin-glass models [5, 7]. Both of these applications are fully described in the survey of Barahoma et al. [5]. In cluster analysis, the task is to divide input data into groups, where data in the same group have some similar properties. It is possible to model the task as the Max Cut problem; an algorithm for clustering based on solving the Max j-Cut problem is provided in [8].

2.2 Tractability

The Max Cut problem began to receive a lot of attention with the release of Karp's 21 NP-complete problems. However, it is also necessary to mention Cook's paper [41], where the problem of determining whether a given propositional formula is tautology (known as the Satisfiability problem) was proven to be NP-complete as the first of all problems. Karp, in his famous paper [3], polynomially reduced 21 classical decision NP problems on each other, including the Max Cut problem, while the one of them was the Satisfiability problem and thus, he showed that all 21 problems are NP-complete.

It is well-known that even the Simple Max Cut problem is NP-complete. It was originally proven by Garey, Johnson, and Stockmeyer [4]. An elegant proof of that was also provided by Poljak and Tuza [42]. In their proof, the known NP-hard problem of finding a maximum independent set of a given graph was reduced to the Simple Max Cut problem. In the same paper, Tuza and Poljak gave a polynomial reduction from the Simple Max Cut

problem to the Simple Max Cut problem for graphs with maximum degree 3. It is worth pointing out that Simple Max Cut for graphs with maximum degree 3 was originally proven to be NP-complete by Papadimitriou and Yannakakis [18].

Unless P = NP, there is no hope for a polynomial time algorithm for (unrestricted) Max Cut or Simple Max Cut. Due to this fact, Max Cut is considered as an *intractable* problem. The simple brute-force solution, for a graph with n vertices and m edges, takes $O(m \cdot 2^n)$, since there is 2^n different vertex partitioning. Although there is a more sophisticated exponential time algorithm for Max Cut running in $\widetilde{O}\ (2^{m/4})$, where the notation $\widetilde{O}$ suppresses polynomial factors in any parameters [43], it is obvious that such an algorithm is not practically usable for large inputs. For this reason, it becomes interesting to investigate the problem restricted on special classes of graphs with some constraint properties.

2.3 Special classes of graphs

As long as one deals with an intractable problem, the important aspect of algorithms is mainly whether they run in polynomial time.

Firstly, it is a fact that solving Max Cut for bipartite graphs is trivial. Perhaps the most significant result in studies of Max Cut for restricted graphs was provided by Hadlock [14]. In his work, Hadlock proposed a polynomial time algorithm for the Max Cut problem for planar graphs. It was shown by Wimer [16] that Max Cut is polynomial solvable for graphs with bounded treewidth. Both algorithms will be described in this section.

Bodlaender and Jansen examined the Simple Max Cut problem on several classes of graphs. In their paper [17], they proved that Simple Max Cut remains NP-complete for chordal graphs, undirected path graphs, 2-split graphs, tripartite graphs and graphs that are the complement of a bipartite graph. They provided polynomial reductions from the Max 2-Sat problem, which was proven to be NP-complete by Garey, Johnson, and Stockmeyer [4], to the Simple Max Cut problem for chordal graphs and for undirected path graphs. The NP-completeness of the Simple Max Cut problem for 2-split graphs was proven by a reduction from the (unrestricted) Simple Max Cut problem. Then, the Simple Max Cut problem for 2-split graphs was reduced to the Simple Max Cut problem for tripartite graphs and for graphs that are the complement of a bipartite graph. They also presented a polynomial time algorithm for the Simple Max Cut problem for cographs, based on dynamic programming.

Another interesting result was given by Grötschel and Nemhauser [44]. Namely, they showed that Max Cut for graphs with bounded size of the longest odd cycle is polynomial solvable. Lastly, it was shown by Arbib [45] that Max Cut for line graphs is also polynomial solvable.

Planar graphs

It is well-known that MAX CUT is polynomial solvable when the input graph is required to be planar. Hadlock [14] established an interesting way to think about the MAX CUT problem; he showed that a set of edges C is a cut in a graph G if and only if a subset of edges $E(G) \setminus C$ is an odd-circuit cover in G. Here, an *odd-circuit cover* is a subset of edges S of a graph G, whose removal from G produces a graph free of odd-circuits (i. e., a bipartite graph). Therefore, the problem of finding a maximum cut can be treated as the minimum odd-circuit cover problem. However, this text describes the algorithm in a slightly different way; the idea and the notation that will be used is based on that of Poljak and Tuza [42].

A *dual graph* of a planar graph G is a multigraph G^* that has a vertex for each face of G. An edge $\{u, v\}$, $u, v \in V(G^*)$, belongs to the edge set of G^*, if face that corresponds to u shares a common edge in G with face that corresponds to v. S^* denotes a subset of edges of a dual representation of a graph G that corresponds to a subset of edges S in G. For a multigraph G, a subgraph H with all vertices of even degree is called an *Eulerian subgraph*. An Eulerian subgraph H of a multigraph G is a *maximum Eulerian subgraph* of G if the total weight of the edge set of H is at least the total weight of the edge set of any Eulerian subgraph of G.

The algorithm needs a dual graph of the input graph. One can construct a dual graph of the input graph easily from a planar embedding of the graph. There is an algorithm for planar embedding that runs in linear time [46].

Lemma 1. *For a graph G and its dual graph G^*, a subset of edges $C \subseteq E(G)$ is a cut in G if and only if a multigraph $(V(G^*), C^*)$ is an Eulerian subgraph of G^*.*

Proof. Suppose that the multigraph $(V(G^*), C^*)$ contains a vertex v with an odd degree. Let J be the set of edges of a cycle in G bordering the face that corresponds to the vertex v. It is easy to see that the size of any cut in any cycle is even. Thus, $|J \cap C|$ must be even, which is a contradiction with the assumption that v has an odd degree in $(V(G^*), C^*)$.

The other direction of the equivalence can be proved in a very similar way. □

Lemma 1 provides a strong result: A cut with the total weight as large as possible in a graph corresponds to the set of edges of a maximum Eulerian subgraph of its dual graph. A maximum Eulerian subgraph of a dual graph can be found in the following way. Let P_G be a disjoint collection of paths with vertices of odd degree (in a dual graph G^*) as endvertices, using each once as an endvertex, and with minimum sum of path total weights. Then, a maximum Eulerian subgraph in G^* can be obtained by removing all edges of G^* that are contained in the collection of paths P_G. The problem of finding

the collection of paths P_G can be easily reduced to the Maximum Matching problem of a complete graph [14].

Since there is a polynomial time algorithm for the Maximum Matching problem [47], there is also a polynomial time algorithm for Max Cut for planar graphs.

Graphs with bounded treewidth

Another class of graphs where the Max Cut problem becomes polynomial solvable are graphs with bounded treewidth. The existence of a polynomial time algorithm for this problem was proven by Wimer [16]. The algorithm is based on dynamic programming. In this section, we describe how the algorithm works for unweighted graphs. The idea of the algorithm and the notation that will be used is based on the work of Bodlaender [17]. Since we are dealing with the Simple Max Cut problem, a graph is undirected and unweighted.

Definition 5 (Tree decomposion [20])**.** *A* tree decomposion *of a graph G is a pair $(\{X_i \mid i \in I\}, T = (I, F))$, where $\{X_i \mid i \in I\}$ is a collection of subsets of $V(G)$, and $T = (I, F)$ is a tree, such that the following conditions hold:*

- $\bigcup_{i \in I} X_i = V(G)$,
- *for every edge $\{u, v\} \in E(G)$, there is a node $i \in I$, with $u, v \in X_i$, and*
- *for every vertex $v \in V(G)$, the induced subgraph of T with nodes $\{i \mid v \in X_i\}$ is connected.*

The *width* of a tree decomposition $(\{X_i \mid i \in I\}, T = (I, F))$ is the size of the largest set of $\{X_i \mid i \in I\}$ minus 1. The *treewidth* of a graph G, denoted $\mathrm{tw}(G)$, is the minimum width over all possible tree decompositions of G.

Definition 6 (Nice tree decomposion)**.** *A* nice tree decomposion *is a tree decomposion $(\{X_i \mid i \in I\}, T = (I, F))$, such that the following rules are satisfied:*

- *T is a rooted binary tree,*
- *if a node $i \in I$ has two children $j, k \in I$ in T, then $X_i = X_j = X_k$, and*
- *if a node $i \in I$ has one child $j \in I$ in T, then either $X_i \subset X_j$ and $|X_j \setminus X_i| = 1$ or $X_j \subset X_i$ and $|X_i \setminus X_j| = 1$.*

The first step of the algorithm is, given a graph G and its treewidth k, to find a nice tree decomposion of G of width equal to k; as long as k is bounded by a constant, a nice tree decomposion of width k with at most $4 \cdot |V(G)|$ nodes can be obtained in time $O(|V(G)|)$ [48].

Since a nice tree decomposion $(\{X_i \mid i \in I\}, T = (I, F))$ of a given graph G is obtained, the computation of the dynamic programming table $tab_{X_i}[A]$ with $A \subseteq X_i$ can start. For $J_i = \{j \in I \mid j = i \text{ or } j \text{ is a descendant of } i \text{ in } T\}$, let $Y_i = \bigcup_{j \in J_i} X_j$. The table $tab_{X_i}[A]$ holds the maximum size of cut edges in the induced subgraph Y_i of G, where vertices from A are in one partition and vertices from $X_i \setminus A$ are in the other partition. It will be shown how the table can be computed efficiently. The computation goes from leafs to the root of T. When the table for a node $i \in I$ is computed, four cases can occur.

- If i is a leaf in T, then

$$tab_{X_i}[A] = |E(A, X_i \setminus A)|.$$

- If i has one child j, such that $X_i = X_j \cup \{v\}$, then

$$tab_{X_i}[A] = tab_{X_j}[A \cap X_i] + \begin{cases} |N(v) \cap (X_i \setminus A)| & v \in A, \\ |N(v) \cap A| & v \notin A. \end{cases}$$

- If i has one child j, such that $X_j = X_i \cup \{v\}$, then

$$tab_{X_i}[A] = \max\{tab_{X_j}[A \cup \{v\}], tab_{X_j}[A]\}.$$

- If i has two children j and k, then

$$tab_{X_i}[A] = tab_{X_j}[A] + tab_{X_k}[A] - |E(A, X_i \setminus A)|.$$

If i is the root of T, then the size of a maximum cut in G is the maximum of $tab_{X_i}[A]$ for $A \subseteq X_i$.

It is easy to see that the computation of $tab_{X_i}[A]$ takes $O(1)$ time. Thus, the computation over the whole tree T can be done in time $O(|V(T)|)$, where $|V(T)|$ is at most $4 \cdot |V(G)|$.

2.4 Approximation algorithms

When there is no need for the optimal solution, very powerful framework for dealing with intractable optimization problems represents approximation algorithms. A p-approximation algorithm for an optimization problem is a polynomial time algorithm that returns a solution that is close to the optimal solution. The aim of a p-approximation algorithm for a maximization problem is to find a solution that is at least p-times the optimal value. The concept of approximation algorithms was introduced by Garey, Graham, and Ullman [12] and Johnson [13].

A very simple approximation algorithm for Simple Max Cut is the Local Search algorithm that guarantees at least 0.5 times the optimal value,

presented by Papadimitriou [49]. The algorithm starts with an arbitrary partition (U, W). Then, the algorithm exhaustively switches partitions of vertices, as long as it increases the total weight of the cut between U and W.

One of the best-known approximation algorithms for Max Cut is the one of Goemans and Williamson [10] based on semidefinite programming. Their algorithm guarantees the expected value of a solution to be at least 0.878 times the optimal value. It was shown by Khot et al. [11] that the algorithm of Goemans and Williamson is optimal, unless the *unique games conjecture* fails. In practise, the algorithm has a complex design and its computation time may be prohibitive when input graphs have more than 1000 vertices [50]. A simpler version of the algorithm of Goemans and Williamson was given by Bertoni, Campadelli, and Grossi [50]. They provided an experimental proof that their algorithm has a better average performance than the algorithm of Goemans and Williamson. However, the algorithm of Bertoni, Campadelli, and Grossi guarantees only at least 0.39 times the optimal value.

2.5 FPT algorithms

One can observe that the algorithm for graphs with bounded treewidth proposed by Wimer [16] is in fact an FPT algorithm.

A well-studied parameterization of Max Cut is the parameterization by the solution size ℓ (here, the cut size). There is a trivial polynomial kernel for that parameterization. First, given an instance (G, ℓ) of Simple Max Cut, it is easy to see that if $\ell \leq |E(G)|/2$, then there is a cut with size at least ℓ in G (i. e. (G, ℓ) is Yes-instance). Otherwise, we have a kernel with at most $2 \cdot \ell$ edges and $2 \cdot \ell + 1$ vertices.

An interesting result came from Mahajan and Raman [51] who showed that Max Cut parameterized by the cut size can be solved in $O(\ell \cdot 4^{\ell} + n + m)$. Lately, for the same parameterization, Rodriguez [52] found an algorithm running in $O(1.414^{\ell} \cdot n^{c})$, for some constant c.

Fedin and Kulikov [53] introduced an FPT algorithm for Max Cut parameterized by the edge set size of a given graph G running in $O(2^{m/4} \cdot \text{poly}(m))$.

Another FPT algorithm for Max Cut was given by Crowston, Jones, and Mnich [54]; their algorithm finds any cut of size $\frac{m}{2} + \frac{n-1}{4} + j$ in time $2^{O(j)} \cdot n^4$, where $j \in \mathbb{N}$ is the parameter. Moreover, they proved that their algorithm is optimal, unless the *strong exponential time hypothesis* fails.

Chapter 3

Current work

3.1 Preliminaries

Unless otherwise stated, all graphs are simple, which means that they are unweighted, undirected, without loops and multiple edges. A *graph* G is a pair (V, E), where V is a nonempty set of *vertices* and E is a set of *edges*. The set of all vertices of the graph G is denoted $V(G)$. Similarly, the set of all edges of the graph G is denoted $E(G)$. For a graph G and subsets of vertices $A \subseteq V(G)$ and $B \subseteq V(G)$, the subset of edges $\{\{u, v\} \in E(G) \mid u \in A, v \in B\}$ is denoted $E(A, B)$.

For a graph G and a pair of vertices $u, v \in V(G)$, the vertex u is a *neighbor* of the vertex v if $\{u, v\} \in E(G)$. If the vertex u is a neighbor of the vertex v, we say u and v are *adjacent*. An edge e is *incident* with a vertex v if $v \in e$. The set of all neighbors of a vertex v in a graph G is denoted $N_G(v)$ and $\deg_G(v) = |N_G(v)|$. We omit the subscript if the graph G is clear from the context.

For a graph G, a graph H with $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$ is a *subgraph* of G. A subgraph H of a graph G with $E(H) = \{\{u, v\} \in E(G) \mid u \in V(H) \wedge v \in V(H)\}$ is an *induced subgraph* of G. For a subset of vertices A of a graph G, the operation $G \setminus A$ produces an induced subgraph G' of G with $V(G') = V(G) \setminus A$. For a subset of edges C of a graph G, the operation $G - C$ produces a graph $(V(G), E(G) \setminus C)$.

A *path graph* is a graph $(\{v_0, \ldots, v_m\}, \{\{v_i, v_{i+1}\} \mid i = 0, \ldots, m-1\}$. If a set of vertices $A = \{v_1, \ldots, v_m\}$ is said to be a *path*, then it is meant a path graph P with $V(P) = \{v_1, \ldots, v_m\}$. A *path in a graph* G is a subgraph P of a graph G such that P is a path graph. For a graph G and a pair of vertices $u, v \in V(G)$, a *path between* u *and* v is a path P in the graph G, such that $u, v \in V(P)$ and $\deg_P(u) = \deg_P(v) = 1$.

A graph G is *connected* if there is a path in G between every pair of vertices of G. We emphasize the following definitions.

Definition 7 (Partition)**.** *A* partition *of a graph G is a pair (U, W), where $U \subset V(G)$ and $W \subset V(G)$ are nonempty subsets of vertices, such that $U \cup W = V(G)$ and $U \cap W = \emptyset$.*

Definition 8 (Cut)**.** *A* cut *in a graph G with a partition (U, W) of G is a subset of edges $E(U, W)$.*

Definition 9 (Maximum cut)**.** *A cut C in a graph G is a* maximum cut *in G if $|C'| \leq |C|$ for any cut C' in G.*

Definition 10 (Vertex cover)**.** *A* vertex cover *of a graph G is a subset of vertices $X \subseteq V(G)$, such that at least one endvertex of every edge of G belongs to X.*

A *minimum vertex cover* of a graph G is a vertex cover of G of the smallest possible size. A *vertex cover number* of a graph G is the size of a minimum vertex cover of G, denoted $\mathrm{vc}(G)$. An *independent set* of a graph G is a subset of vertices $B \subseteq V(G)$, such that $\{u, v\} \notin E(G)$ for every pair of vertices $u, v \in B$.

3.2 Polynomial kernel for Max Cut parameterized by vertex cover number

In this section, we propose a new kernelization algorithm for Simple Max Cut parameterized by vertex cover number. This section is divided into two parts: First, we construct one reduction rule. The second part describes the algorithm whose idea is to exhaustively apply the reduction rule.

For simplicity, in what follows, if a vertex cover X of the given graph is clear from the context, a positive integer k refers to the size of the vertex cover X. Similarly, a set of vertices B corresponds to the complement of the vertex cover X in the given graph G.

Let us recall the problem we are dealing with.

Simple Max Cut	
Input:	A graph G and a positive integer ℓ,
Question:	Is there a cut C in G with $\|C\| \geq \ell$?

The reduction rule

Observations 1 and 2 point on basic, but important properties of the vertex cover and its independent set.

Observation 1. *Let G be a graph with a vertex cover X. Every vertex $b \in B$ satisfies* $\deg(b) \leq k$.

Observation 1 is trivially true, since vertices of an independent set can be adjacent only with vertices from the vertex cover.

Observation 2. *Let G be a graph with a vertex cover X. For the induced subgraph G' of G with $V(G') = X$, $|E(G')| \leq \binom{k}{2}$.*

In other words, Observation 2 states that there are at most $\binom{k}{2}$ edges between vertices of the vertex cover. Indeed, if $E(G')$ is as large as possible, then G' is a complete graph with $|E(G')| = \binom{k}{2}$.

Consider a cut between the vertex cover and the complement of the vertex cover. It turns out that the size of this cut is always larger than cuts, where vertices of the vertex cover with more than $\binom{k}{2}$ common neighbors are in different partitions.

Lemma 2. *Let G be a graph with a vertex cover X. Consider a partition $(S, V(G) \setminus S)$ of the graph G with a maximum cut C. If there is a pair of vertices $u, v \in V(G)$, such that $|N(u) \cap N(v)| > \binom{k}{2}$, then both $u, v \in S$ or $u, v \in V(G) \setminus S$.*

Proof. Suppose for a contradiction that either $u \in S \wedge v \in V(G) \setminus S$ or $v \in S \wedge u \in V(G) \setminus S$. Consider a set of paths

$$\mathcal{P} = \{\{u, w, v\} \mid w \in N(u) \cap N(v)\}.$$

For each $i \in \{1, \ldots, q\}$, q is the size of the set $\mathcal{P}$, there is a path $P^{\mathrm{i}} \in \mathcal{P}$ with exactly one edge $e_i \in E(P^{\mathrm{i}})$, such that $e_i \notin C$. Otherwise u and v are in the same partition. Let K be a set of such edges, that is, $K = \{e_1, \ldots, e_q\}$. It follows that $|K| > \binom{k}{2}$. Consider a cut $C' = E(X, B)$. We prove, that $|C'| > |C|$. According to Observation 2, we get the following inequation

$$|C'| \geq |E(G)| - \binom{k}{2} > |E(G)| - |K| \geq |C|.$$

Therefore, the cut C is not a maximum cut of G. □

Lemma 2 raises a reasonable question: What is the property of some vertex v from the vertex cover, such that there is a vertex u that has more than $\binom{k}{2}$ common neighbors with v? Lemma 3 shows natural, but important generalization of the answer.

Lemma 3. *Let G be a graph with a vertex cover X. Consider a subset of vertices $Y \subset X$ with $v \in Y$. If*

$$|E(N(v) \cap B, X \setminus Y)| > (k-1) \cdot \binom{k}{2},$$

then there exists $u \in X \setminus Y$, such that $|N(v) \cap N(u)| > \binom{k}{2}$.

Proof. Assume that every vertex $u \in X \setminus Y$ satisfies $|N(v) \cap N(u)| \leq \binom{k}{2}$. Hence, there are at most $|X \setminus Y| \cdot \binom{k}{2}$ edges in $E(N(v) \cap B, X \setminus Y)$. This gives a contradiction, since $|X \setminus Y| < k$. □

Reduction rule 1. *Let (G, ℓ) be an instance of Simple Max Cut. Let X be a vertex cover of G with $v \in X$, such that $\deg(v) > (k-1) \cdot (\binom{k}{2} + 1)$. Consider a set of vertices*

$$A = \{v\} \cup \{u \in X \mid \binom{k}{2} < |N(v) \cap N(u)|\}.$$

Let $b \in N(v) \cap B$ be a vertex, such that $|N(b) \cap A| = \deg(b)$. Transform (G, ℓ) into $(G \setminus \{b\}, \ell - \deg(b))$.

The proof of Reduction rule 1 consists of two parts. First, it will be shown that, according to Lemma 3, the vertex b must exists.

Lemma 4. *The vertex b from Reduction rule 1 exists.*

Proof. Consider a set of edges $M = E(N(v) \cap B, X \setminus A)$. Assume for a contradiction that $\deg(b) > |N(b) \cap A|$. Then $|N(b) \setminus A| > 0$ and a vertex $u \in N(b) \setminus A$ exists. Consider an edge $e = \{b, u\}$. It follows, that $e \in M$. The size of M is increased at least by 1 by every vertex of $N(v) \cap B$. Since $|N(v) \cap X| \leq (k-1)$, it follows that $|N(v) \cap B| > (k-1) \cdot \binom{k}{2}$. The size of M is the following

$$|M| \geq |N(v) \cap B| > (k-1) \cdot \binom{k}{2}.$$

It is easy to see that $A \subset X$ and therefore, according to Lemma 3, there exists a vertex $w \in X \setminus A$, such that $|N(v) \cap N(w)| > \binom{k}{2}$. This is a contradiction, since $w \notin A$. □

Now, everything is prepared for the proof of the Reduction rule 1 safeness.

Lemma 5. *Reduction rule 1 is safe.*

Proof. Consider a partition $(S, V(G) \setminus S)$ of the graph G with a maximum cut C. According to Lemma 2, the vertex b has all its neighbors in one partition. Let S be such a partition, that is, $A \subseteq S$. Consequently, $b \in V(G) \setminus S$, otherwise the cut $C \cup E(\{b\}, A)$ contains more edges than C. Thus, edges $E(\{b\}, A)$ are in the maximum cut C and instances (G, ℓ) and $(G \setminus \{b\}, \ell - \deg(b))$ are equal. □

Claim 1. *Let (G, ℓ) be an instance of* Simple Max Cut. *Let X be a vertex cover of G. After exhaustive application of Reduction rule 1 on the instance (G, ℓ), we obtain an instance $(G', \ell - p)$, such that*

$$|V(G')| \leq k + k \cdot (k-1) \cdot (\binom{k}{2} + 1),$$

where $p \in \mathbb{N}_0$.

Proof. According to Reduction rule 1, every vertex from X has at most $(k-1) \cdot (\binom{k}{2} + 1)$ neighbors. Hence, the size of B is at most $k \cdot (k-1) \cdot (\binom{k}{2} + 1)$. The size of X is k and the inequation is satisfied. □

The kernelization algorithm

The idea of the algorithm is to exhaustively apply Reduction Rule 1. In the following pseudocode, we use the RAM model [55] and graphs are represented as adjacency lists.

The input of the algorithm are an instance of Simple Max Cut (G, ℓ) and a vertex cover X of the graph G. Recall that there is the simple approximation algorithm for the Minimum Vertex Cover problem that, for a given graph G, runs in time $O(|E(G)|)$ and returns a vertex cover of G with size at most two times greater than the size of a minimum vertex cover of G [40, p. 134].

Therefore, we presume that the size of the input vertex cover X is at most two times greater than vertex cover number of G. That is, we presume that $k \leq 2 \cdot \mathrm{vc}(G)$.

Algorithm 1

Input: An instance of Simple Max Cut (G, ℓ), a vertex cover X of G,
Output: An instance of Simple Max Cut $(G', \ell - p)$.

```
1: G' ← G
2: p ← 0
3: for v ∈ X do
4:     while deg(v) > (k − 1) · ((k choose 2) + 1) do
5:         A ← {v}
6:         for u ∈ X \ {v} do
7:             if |N(v) ∩ N(u)| > (k choose 2) then
8:                 A ← A ∪ {u}
9:             end if
10:        end for
11:        for b ∈ N(v) ∩ B do
12:            if deg(b) = |N(b) ∩ A| then
13:                G' ← G' \ {b}
14:                p ← p + deg(b)
15:                break
16:            end if
17:        end for
18:    end while
19: end for
```

Claim 2. *Algorithm 1 returns an instance $(G', \ell - p)$ with $|V(G')| \in O(k^4)$.*

Proof. Since the notion of Algorithm 1 is to exhaustively apply Reduction rule 1, the proof follows from Claim 1. □

Claim 3. *Algorithm 1 runs in time $O(k^3 \cdot n^3)$.*

Proof. The *for* loop on the line 3 will run exactly k times. Since each iteration of the *while* loop on the line 4 removes one neighbor of the vertex v, it will be executed $O(n)$ times. The *for* loop on the line 6 will run $O(k)$ times. The check of the condition on the line 7 requires time $O(n^2)$. The *for* loop on the line 11 costs $O(n)$ and the condition on line 12 can be checked in time $O(k^2)$. We get the following running time:

$$O(k \cdot n \cdot (k \cdot n^2 + n \cdot k^2)) = O(k^3 \cdot n^3).$$

□

One can observe that Theorem 1 was already proven. Since k never exceed n, Algorithm 1 runs in polynomial time with respect to the size of the input graph G. Under the assumption that $k \leq 2 \cdot \mathrm{vc}(G)$, Algorithm 1 returns the kernel with $O((2 \cdot \mathrm{vc}(G))^4) = O(\mathrm{vc}(G)^4)$ vertices.

Conclusion

This thesis provided a kernelization algorithm for the Simple Max Cut problem that returns a kernel with $O(\mathrm{vc}(G)^4)$ vertices. The result emphasizes the significance of vertex cover number parameterization and supports the idea that vertex cover number might be a suitable parameter wherever treewidth fails. Due to this result, two possible generalizations arise.

Firstly, one can ask for generalization of the algorithm for (edge) weighted graphs, that is, to find a polynomial kernel for the Max Cut problem. Unfortunately, since Observation 2 holds only when the graph is unweighted, the way of dealing with this generalization shall be different.

Secondly, the concept of vertex cover can be generalized to *j-path vertex cover* (j-PVC). Here, j-PVC for a graph G is a subset of vertices $U \subseteq V(G)$, such that every path in G with j edges contains at least one vertex from U. One can observe that 1-PVC coincides with vertex cover number. Now, when we know that there is polynomial kernel for the Simple Max Cut problem parameterized by j-PVC when j is equal to 1, it becomes interesting to ask whether it holds for other values of j. We leave this question open for further research.

Bibliography

[1] Goldschmidt, O.; Hochbaum, D. S. Polynomial algorithm for the k-cut problem. In *[Proceedings 1988] 29th Annual Symposium on Foundations of Computer Science*, IEEE, 1988, ISBN 978-0-8186-0877-3, pp. 444–451.

[2] Ford, L. R.; Fulkerson, D. R. Maximal flow through a network. In *Classic papers in combinatorics*, Springer, 2009, ISBN 978-0-8176-4841-1, pp. 243–248.

[3] Karp, R. M. Reducibility among combinatorial problems. In *Complexity of computer computations*, Springer, 1972, ISBN 978-1-4684-2001-2, pp. 85–103.

[4] Garey, M. R.; Johnson, D. S.; et al. Some simplified NP-complete problems. In *Proceedings of the sixth annual ACM symposium on Theory of computing*, 1974, ISBN 978-1-4503-7423-1, pp. 47–63.

[5] Barahona, F.; Grötschel, M.; et al. An application of combinatorial optimization to statistical physics and circuit layout design. *Operations Research*, volume 36, no. 3, 1988: pp. 493–513, doi:10.1287/opre.36.3.493.

[6] Liers, F.; Nieberg, T.; et al. Via Minimization in VLSI Chip Design-Application of a Planar Max-Cut Algorithm. *Department of Computer Science, Faculty of Mathematics and Natural Sciences, Cologne University, Technical report*, 2011: pp. 1–15. Available from: `http://e-archive.informatik.uni-koeln.de/id/eprint/630`

[7] Galluccio, A.; Loebl, M.; et al. Optimization via enumeration: a new algorithm for the max cut problem. *Mathematical Programming*, volume 90, no. 2, 2001: pp. 273–290, doi:10.1007/PL00011425.

[8] Poland, J.; Zeugmann, T. Clustering pairwise distances with missing data: Maximum cuts versus normalized cuts. In *International Confer-*

ence on Discovery Science, Springer, Springer Berlin Heidelberg, 2006, ISBN 978-3-5404-6493-8, pp. 197–208.

[9] Rendl, F.; Rinaldi, G.; et al. Solving max-cut to optimality by intersecting semidefinite and polyhedral relaxations. *Mathematical Programming*, volume 121, no. 2, 2010: pp. 307–335, doi:10.1007/s10107-008-0235-8.

[10] Goemans, M. X.; Williamson, D. P. .879-approximation algorithms for MAX CUT and MAX 2SAT. In *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, 1994, ISBN 978-0-8979-1663-9, pp. 422–431.

[11] Kindler, G.; O'Donnell, R.; et al. Optimal inapproximability results for MAX-CUT and other 2-variable CSPs? *Electronic Colloquium on Computational Complexity (ECCC)*, volume 37, 2005, doi:10.1137/S0097539705447372.

[12] Garey, M.; Graham, R.; et al. An analysis of some packing algorithms. In *Combinatorial Algorithms (Courant Computer Science Symposium)*, 9, 1972, ISBN 978-0-9174-4803-4, pp. 39–47.

[13] Johnson, D. S. Approximation algorithms for combinatorial problems. In *Proceedings of the fifth annual ACM symposium on Theory of computing*, 1973, ISBN 978-1-4503-7430-9, pp. 38–49.

[14] Hadlock, F. Finding a maximum cut of a planar graph in polynomial time. *SIAM Journal on Computing*, volume 4, no. 3, 1975: pp. 221–225, ISSN 1095-7111.

[15] Dorfman, Y.; Orlova, G. Finding the maximum cut in a planar. *Engineering Cybernetics*, volume 10, 1975: pp. 502–506.

[16] Wimer, T. V. *Linear Algorithms on K-Terminal Graphs*. Dissertation thesis, Clemson University, 1987.

[17] Bodlaender, H. L.; Jansen, K. On the complexity of the maximum cut problem. *Nordic Journal of Computing*, volume 7, no. 1, 2000: pp. 14–31, ISSN 1236-6064.

[18] Papadimitriou, C. H.; Yannakakis, M. Optimization, approximation, and complexity classes. *Journal of computer and system sciences*, volume 43, no. 3, 1991: pp. 425–440, ISSN 0022-0000.

[19] Cygan, M.; Fomin, F. V.; et al. *Parameterized algorithms*, volume 4. Springer, 2015, ISBN 978-3-3192-1274-6.

[20] Robertson, N.; Seymour, P. D. Graph minors. II. Algorithmic aspects of tree-width. *Journal of algorithms*, volume 7, no. 3, 1986: pp. 309–322, doi:10.1016/0196-6774(86)90023-4.

[21] Bodlaender, H. L. Treewidth: characterizations, applications, and computations. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, Springer, 2006, ISBN 978-3-5404-8382-3, pp. 1–14.

[22] Courcelle, B. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Information and computation*, volume 85, no. 1, 1990: pp. 12–75, ISSN 0890-5401.

[23] Bodlaender, H. L. A tourist guide through treewidth. In *Developments in Theoretical Computer Science*, volume 6, CRC Press, 1994, ISBN 978-2-8812-4961-7, pp. 1–20.

[24] Bodlaender, H. L. Discovering treewidth. In *International Conference on Current Trends in Theory and Practice of Computer Science*, Springer, 2005, ISBN 978-3-5402-4302-1, pp. 1–16.

[25] Reed, B. A. Algorithmic aspects of tree width. In *Recent advances in algorithms and combinatorics*, Springer, 2003, ISBN 978-0-3879-5434-9, pp. 85–107.

[26] Kloks, T. *Treewidth: computations and approximations*, volume 842. Springer Science & Business Media, 1994, ISBN 978-3-5405-8356-1.

[27] Downey, R. G.; Fellows, M. R. Fixed-parameter tractability and completeness I: Basic results. *SIAM Journal on Computing*, volume 24, no. 4, 1995: pp. 873–921, doi:10.1137/S0097539792228228.

[28] Downey, R. G.; Fellows, M. R. Fixed-parameter tractability and completeness II: On completeness for W [1]. *Theoretical Computer Science*, volume 141, no. 1-2, 1995: pp. 109–131, doi:10.1016/0304-3975(94)00097-3.

[29] Downey, R.; Fellows, M. Fixed-Parameter Tractability and Completeness III: Some Structural Aspects of the W Hierarchy. In *Complexity Theory: Current Research*, Cambridge University Press, 1993, ISBN 978-0-5214-4220-6, p. 191–225.

[30] Downey, R. G.; Fellows, M. R. *Parameterized complexity*. Springer Science & Business Media, 2012, ISBN 978-1-4612-0515-9.

[31] Downey, R. G.; Fellows, M. R. *Fundamentals of parameterized complexity*, volume 4. Springer, 2013, ISBN 978-1-4471-5559-1.

[32] Niedermeier, R. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006, ISBN 978-0-1985-6607-6.

[33] Flum, J.; Grohe, M. *Parameterized Complexity Theory*. Springer-Verlag, 2006, ISBN 978-3-5402-9952-3.

[34] Fomin, F. V.; Lokshtanov, D.; et al. *Kernelization: theory of parameterized preprocessing*. Cambridge University Press, 2019, ISBN 978-1-1074-1515-7.

[35] Bodlaender, H. L.; Downey, R. G.; et al. On problems without polynomial kernels. *Journal of Computer and System Sciences*, volume 75, no. 8, 2009: pp. 423–434, ISSN 0022-0000.

[36] Lokshtanov, D.; Marx, D.; et al. Lower bounds based on the Exponential Time Hypothesis. *Bulletin of the EATCS*, volume 105, 2011: pp. 41–72, doi:10.1007/978-3-319-21275-3_14.

[37] Impagliazzo, R.; Paturi, R. On the complexity of k-SAT. *Journal of Computer and System Sciences*, volume 62, no. 2, 2001: pp. 367–375, ISSN 0022-0000.

[38] Impagliazzo, R.; Paturi, R.; et al. Which problems have strongly exponential complexity? In *Proceedings 39th Annual Symposium on Foundations of Computer Science (Cat. No. 98CB36280)*, IEEE, 1998, ISBN 978-0-8186-9172-7, pp. 653–662.

[39] Fellows, M. R.; Lokshtanov, D.; et al. Graph layout problems parameterized by vertex cover. In *International Symposium on Algorithms and Computation*, Springer, 2008, ISBN 978-3-5409-2182-0, pp. 294–305.

[40] Papadimitriou, C. H.; Steiglitz, K. *Combinatorial optimization: algorithms and complexity*. Courier Corporation, 1998, ISBN 978-0-4864-0258-1.

[41] Cook, S. A. The Complexity of Theorem-Proving Procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, Association for Computing Machinery, 1971, ISBN 978-1-4503-7464-4, p. 151–158.

[42] Poljak, S.; Tuza, Z. Maximum cuts and largest bipartite subgraphs. In *Combinatorial optimization: papers from the DIMACS Special Year*, volume 20, American Mathematical Soc., 1995, ISBN 978-1-4704-3978-1, pp. 181–244.

[43] Scott, A. D.; Sorkin, G. B. Faster Algorithms for MAX CUT and MAX CSP, with Polynomial Expected Time for Sparse Instances. In *Approximation, Randomization, and Combinatorial Optimization.. Algorithms and Techniques*, Springer Berlin Heidelberg, 2003, ISBN 978-3-5404-5198-3, pp. 382–395.

[44] Grötschel, M.; Nemhauser, G. L. A polynomial algorithm for the max-cut problem on graphs without long odd cycles. *Mathematical Programming*, volume 29, no. 1, 1984: pp. 28–40, doi:10.1007/BF02591727.

[45] Arbib, C. A polynomial characterization of some graph partitioning problems. *Information processing letters*, volume 26, no. 5, 1988: pp. 223–230, doi:10.1016/0020-0190(88)90144-5.

[46] Boyer, J. M.; Myrvold, W. J. On the cutting edge: simplified O (n) planarity by edge addition. *Journal of Graph Algorithms and Applications*, volume 8, no. 3, 2004: pp. 241–273, doi:10.7155/jgaa.00091.

[47] Edmonds, J. Maximum matching and a polyhedron with 0, 1-vertices. *Journal of research of the National Bureau of Standards B*, volume 69, no. 125-130, 1965: pp. 55–56, doi:10.6028/jres.069b.013.

[48] Bodlaender, H. L.; Kloks, T. Better algorithms for the pathwidth and treewidth of graphs. In *International Colloquium on Automata, Languages, and Programming*, 18, Springer, 1991, ISBN 978-3-5405-4233-9, pp. 544–555.

[49] Papadimitriou, C. H. *Computational complexity*. John Wiley and Sons Ltd., 2003, ISBN 978-0-2015-3082-7.

[50] Bertoni, A.; Campadelli, P.; et al. An approximation algorithm for the maximum cut problem and its experimental analysis. *Discrete applied mathematics*, volume 110, no. 1, 2001: pp. 3–12, doi:10.1016/S0166-218X(00)00299-7.

[51] Mahajan, M.; Raman, V. Parameterizing above Guaranteed Values: MaxSat and MaxCut. *Journal of Algorithms*, volume 31, 1997: pp. 335–354, doi:10.1006/jagm.1998.0996.

[52] Rodríguez, E. P. *Systematic kernelization in FPT algorithm design*. Dissertation thesis, The University of Newcastle, 2005.

[53] Kulikov, A.; Fedin, S. A $2^{|E|/4}$-time Algorithm for MAX-CUT. *Journal of Mathematical Sciences*, volume 126, 2005, doi:10.1007/s10958-005-0101-7.

[54] Crowston, R.; Jones, M.; et al. Max-cut parameterized above the Edwards-Erdős bound. In *International Colloquium on Automata, Languages, and Programming*, Springer, 2012, ISBN 978-3-6423-1584-8, pp. 242–253.

[55] Cook, S. A.; Reckhow, R. A. Time bounded random access machines. *Journal of Computer and System Sciences*, volume 7, no. 4, 1973: pp. 354–375, doi:10.1016/S0022-0000(73)80029-7.