Utrecht University

Refining Approximation Algorithms on Planar Graphs

Master Thesis Computing Science

Author: Jens Hoppenreijs

Supervisor: Dr. Jesper Nederlof
Second supervisor: Dr. Erik Jan van Leeuwen

Graduate School of Natural Sciences
Utrecht University

July 2023

Abstract

A result by Marx shows that there does not exist a PTAS for Maximum Independent Set on planar graphs running in $2^{O(1/\epsilon)^{1-\delta}} n^{O(1)}$ time, for any $\delta > 0$, assuming ETH holds. We refine this result by showing that the same problem does not admit a PTAS running in $2^{o(1/\epsilon)} n^{O(1)}$ time, assuming Gap-ETH holds. This refined lower bound is tight in the sense that it matches the running time of the best known PTAS for Maximum Independent Set on planar graphs by Baker in its reliance on ϵ. We observe that this does not exclude an algorithm that yields $(1-\epsilon)$-approximations for only specific values of ϵ in $2^{o(1/\epsilon)} n^{O(1)}$ time, and expand upon this observation to show the exact algorithm for Maximum Independent Set on planar graphs is optimal for $\epsilon \leq O(1/\sqrt{n})$. Additionally, we present polynomial-space PTAS's for the planar versions of Maximum Independent Set, Maximum Weighted Independent Set, Minimum Dominating Set, and Minimum Vertex Cover.

Contents

1 Introduction

A well-known hypothesis in the field of computational complexity is that P $\neq$ NP. This hypothesis implies that problems in the class NP cannot be solved optimally in a running time that is polynomial with respect to the size of the input. As it turns out, many of the problems studied in the field of theoretical computer science are NP-complete, meaning they are NP-hard and in NP. Some examples include Subset Sum, Knapsack, and Independent Set. Assuming that P $\neq$ NP holds, we cannot hope to solve any of these problems optimally in polynomial time. As a consequence, large instances of these problems may not be solvable using exact algorithms at all.

Instead, a common approach for solving NP-complete problems is by using approximation algorithms. An approximation algorithm is an algorithm of which the value of the resulting solution is only guaranteed to be within some factor of the optimal solution. In return for this concession on the quality of the solution, approximation algorithms have running times that are polynomial in the size of the input. A simple example of a 2-approximation algorithm for the Euclidean Traveling Salesman problem is to compute the minimum spanning tree of a graph, and visiting each vertex by traversing the tree in depth-first order. The order in which vertices encountered forms a tour that is at most twice the length of the optimal tour [1]. This result was improved upon by Christofides and Serdyukov who independently of each other, in 1976 and 1978 respectively, presented an algorithm producing a tour that is at most $\frac{3}{2}$ times the length of the optimal tour [2; 3].

There also exist approximation algorithms in which the approximation factor is a parameter that is used within the algorithm itself. A *polynomial-time approximation scheme* (PTAS) is a $(1-\epsilon)$-approximation algorithm, for each value $0 < \epsilon < 1$, with a running time that is commonly of the form $n^{O(1/\epsilon)}$. This means the running time is polynomial in the size of the instance n, for fixed values of ϵ. In the case of minimization problems, a PTAS yields $(1+\epsilon)$-approximations instead. As an example, in 1998 Arora presented a PTAS for the Euclidean versions of Traveling Salesman and Minimum Steiner Tree [4]. Some problems admit efficient polynomial-time approximation schemes (EPTAS), which are even faster in their reliance on n.

There are problems that do not admit a PTAS or EPTAS, unless P = NP, such as the Maximum Clique problem. In this graph problem, the goal is to find a subset of vertices in the graph of maximum cardinality, such that each vertex in the subset has an edge to all other vertices in the subset. Maximum Clique is closely related to another problem called Maximum Independent Set, in which the goal is to find a subset of the vertices of maximum cardinality, such that none of the selected vertices are adjacent. Since a clique in the original graph is an independent set in the complement graph, any algorithm for Maximum Independent Set can also be used to solve instances of Maximum Clique, as was shown by Gallai [5] in 1959. These two problems are widely studied, and have applications in the fields of synthetic biology, chemistry, bioinformatics, and the study of social networks [6; 7]. It has been shown that for any constant $\epsilon > 0$, there does not exist a polynomial-time $O(n^{\epsilon-1})$-approximation algorithm for Maximum Clique, unless P = NP [8; 9]. This means it is hard to approximate Maximum Independent Set as well.

Fortunately, it is possible to do better on special classes of graphs, such as *planar graphs*. Planar graphs have an embedding that can be drawn in the plane without any edges crossing. This gives planar graphs certain properties that can be used to develop more efficient approximation algorithms, and more efficient algorithms in general. A common method when designing approximation schemes for planar graph problems is to use a vertex separator. This is a subset of a graph's vertices of bounded size, the removal of which splits the graph into two or more components. A well-known technique for obtaining vertex separators of bounded size was introduced by Lipton and Tarjan in 1979 [10]. In their *planar separator theorem*, they showed that for any n-vertex planar graph there is a vertex set S of size $O(\sqrt{n})$, such that when S is removed from the graph the remaining components each contain at most $2n/3$ vertices. This can be used for solving combinatorial problems efficiently with a divide-and-conquer approach [11]. Additionally, such a balanced separator is useful when designing approximation schemes, as it is an efficient way of splitting a problem into subproblems.

In 1983, a new technique for separating planar graphs was proposed by Baker, who formally published the technique in a paper in 1994 [12]. Baker's technique works by partitioning a planar graph into k-outerplanar components on which problems can be solved efficiently. Baker showed that in such k-outerplanar components Maximum Independent Set can be solved optimally in $O(8^k n)$ time, where n is the number of vertices in the component. Merging the solutions of different components yields a $k/(k+1)$-approximation of the maximum independent set of the original graph.

In 1986, Robertson and Seymour introduced the concept of the *treewidth* of a graph [13], though it had previously been discovered as the dimension of a graph by Bertelè and Brioschi [14]. This concept is especially useful when designing algorithms for graphs with bounded treewidth, such as planar graphs. In 1988, Bodlaender showed that a number of problems that are NP-complete on general graphs can be solved efficiently on graphs with bounded treewidth, by dynamic programming on the tree decomposition of a graph [15].

Another useful method for designing approximation schemes on planar graphs was proposed by Frederickson in 1987 [16], who showed that the planar separator theorem can be used to obtain an *r-division* of a graph. This is a division of an n-vertex planar graph into $\Theta(n/r)$ regions of $O(r)$ vertices each, and $O(\sqrt{r})$ boundary vertices each. A recent example of such r-divisions being used for designing approximation algorithms is a paper by Goodrich et al. [17], who used r-divisions to create an approximation algorithm for the Tracking Paths problem. Other methods commonly used for designing approximation schemes include the use of kernels to reduce the size of the problem instance, and linear programming [18].

An obvious question is whether the running times of existing approximation schemes can be improved. In 1997, Cesati and Trevisan showed that there are problems which admit a PTAS, but do not admit an EPTAS unless W[P] = FPT [19]. For problems that do admit an EPTAS, we can investigate whether the dependence on ϵ can be improved. In 2006, Huang and Chen showed that there does not exist an $2^{o(\sqrt{1/\epsilon})}n^{O(1)}$ EPTAS for the planar versions of Maximum Independent Set, Minimum Vertex Cover, and Minimum Dominating Set, unless all SNP problems have subexponential-time algorithms. This lower bound was improved upon by Marx, who showed that those same problems do not admit an EPTAS running in $2^{O(1/\epsilon)^{1-\delta}}n^{O(1)}$ time, for any $\delta > 0$, assuming ETH holds. An alternative approach to improving existing approximation schemes is to reduce their space usage. Many PTAS's, including Baker's PTAS for Maximum Independent Set [12], use dynamic programming subroutines that require space that is exponential in the size of the input. Reducing this space usage to polynomial space could greatly improve the practical applications for such approximation schemes.

In this thesis, we prove a tight lower bound on a PTAS for Maximum Independent Set on planar graphs. We follow a sequence of reductions by Marx, but use a different hypothesis as starting point, and modify the reductions to obtain the refined lower bound. Our result is formalized in the following theorem.

Theorem 1.1. *There does not exist a PTAS for* Maximum Independent Set *on planar graphs running in $2^{o(1/\epsilon)}n^{O(1)}$ time, unless Gap-ETH fails.*

This lower bound implies that the best known PTAS for Maximum Independent Set on planar graphs by Baker [12] is optimal in its reliance on ϵ. However, this does not exclude a $(1-\epsilon)$-approximation algorithm for Maximum Independent Set on planar graphs running in $2^{o(1/\epsilon)}n^{O(1)}$ time for only specific values of ϵ. Expanding upon this observation, we refine our lower bound and show that the following theorem holds.

Theorem 1.2. *For any $\epsilon \leq O(1/\sqrt{n})$, there does not exist a $(1-\epsilon)$-approximation algorithm for* Maximum Independent Set *on n-vertex planar graphs running in $2^{o(\sqrt{n})}$ time, unless Gap-ETH fails.*

By this theorem, a $(1-\epsilon)$-approximation algorithm for Planar Maximum Independent Set must run in at least $2^{O(\sqrt{n})}$ time, for any $\epsilon \leq O(1/\sqrt{n})$, assuming Gap-ETH holds. This running time matches the running time of the exact algorithm for Maximum Independent Set on planar

graphs, which implies that the exact algorithm is optimal for obtaining $(1-\epsilon)$-approximations of maximum independent sets on planar graphs, for any $\epsilon \leq O(1/\sqrt{n})$, assuming Gap-ETH holds.

We also present polynomial-time approximation schemes that use only polynomial space for a number of problems on planar graphs. The following theorem contains an example of one such result.

Theorem 1.3. *There is a PTAS for* Minimum Dominating Set *on planar graphs running in* $2^{O(1/\epsilon)}\epsilon^2 n + O(n^3)$ *time and using polynomial space.*

To our knowledge, these results were not previously known. Other planar graph problems we present a polynomial-space PTAS for include Maximum Independent Set, Maximum Weighted Independent Set, and Minimum Vertex Cover.

In Section 2, we introduce preliminary concepts that will be used throughout the rest of the thesis. In Section 3, we prove the tight lower bound for a PTAS for Maximum Independent Set on planar graphs. In Section 4, we present polynomial-space PTAS's for a number of planar graph problems. In Section 5, we conclude by summarizing our results, and giving some suggestions for future research.

2 Preliminaries

In this section, a number of preliminary concepts that will be used throughout the rest of the thesis are introduced.

2.1 Graphs

A graph $G = (V, E)$ is a data structure represented by a pair, where V is a set of vertices (or nodes) and E is a set of edges between the vertices. An edge is a connection between a pair of vertices (u, v), which are also called the endpoints of an edge. An edge is directed if it is represented by an ordered pair of vertices. If the pair of vertices is unordered, then the edge is undirected. A graph in which any edge is directed is called a directed graph. Otherwise, the graph is called an undirected graph. A weighted graph is a graph in which the vertices or edges have a weight assigned to them. The weight of a vertex $v \in V$ is defined by a weight function $w(v)$. An unweighted graph can be seen as a special case of a weighted graph, where for each vertex $v \in V$ it holds that $w(v) = \frac{1}{|V|}$. An undirected graph is called a connected graph (or connected component), if between any two vertices $u, v \in V$ there is a subset of edges in E that forms a path from u to v. In this thesis, unless otherwise specified, a graph refers to a graph that is undirected, unweighted, and connected. Similarly, a weighted graph refers to a graph that is undirected, weighted, and connected. Unless otherwise specified, the vertex weights of a weighted graph add up to 1.

An *embedding* of a graph G is a drawing or visualization of G into the plane. An embedding is planar if it does not contain any intersecting edges. A *plane graph* is a graph that has a planar embedding. A *planar graph* is a plane graph along with a planar embedding of the graph. In many theorems in this thesis, we assume we are given a planar graph, meaning we do not need to take computing an embedding into account. However, the embedding of a plane graph can be computed in linear time [20], and so these theorems would not change if we were given a plane graph instead.

Planar graphs have structural properties that can be used to create algorithms with bounded complexity. For example, using Euler's formula it can be shown that a planar graph with n vertices has at most $3n - 6$ edges.

A *tree* is a special case of a plane graph that does not contain cycles. This property allows for efficient dynamic programming algorithms. A *path graph* is a special case of a tree, in which the vertices form an ordering $v_1, ..., v_n$, such that the only edges are $(v_1, v_2), ..., (v_{n-1}, v_n)$, i.e. the graph forms a path.

A method of classifying a planar graph is by the number of *levels* its embedding consists of. The level 1 vertices of a planar graph are those vertices that lie on the exterior face. Level i vertices in a planar graph are the vertices that lie on the exterior face after removing the vertices of levels $1, ..., i-1$. A k-outerplanar graph is a planar graph in which all of the vertices are of level k or less. The *outerplanarity* of a graph is the number of levels in its embedding, meaning a k-outerplanar graph has an outerplanarity of k. The concept of k-outerplanar graphs is useful, as there exist algorithms that have a running time bounded by k.

2.2 Linear Programming

Linear programming is a technique for modeling and optimising problems that can be formulated using linear constraints and a linear objective function. A linear program (LP) consists of a set of variables, a set of constraints on the variables, and an objective function (or value function). The constraints act as boundaries of the solution space, within which a valid solution must be located. This valid solution space is called the feasible region of a linear program. The optimal solution to a linear program is then the solution within the feasible region that has the optimal value by the objective function.

The following is an example of an LP for finding two positive numbers x and y, such that $x - y$ is as great as possible, and given that their sum must be at most 3.14.

$$\begin{aligned} \text{maximize} \quad & x - y \\ \text{subject to} \quad & x + y \leq 3.14 \\ & x, y \geq 0 \end{aligned}$$

Note that since the constraint on the variables x and y individually is only that their value must be at least zero, they can take both integer and real values. There are algorithms for solving linear programming problems in polynomial time, such as an $O(n^{2.5})$-time algorithm by Vaidya [21]. There is a special case of linear programming called *integer linear programming*, in which variables can only hold integer values. The following is an example of an integer linear program (ILP) for the Knapsack problem, in which the goal is to, given a set of n items $1 \leq i \leq n$ with value c_i and weight w_i, select a subset of items of maximum total value, such that their weight does not exceed the sack capacity B. For each of the items we introduce a binary decision variable x_i that is equal to 1 if we select item i to be placed in the sack, and 0 otherwise.

$$\begin{aligned} \text{maximize} \quad & \Sigma_{i=1}^{n} x_i c_i \\ \text{subject to} \quad & \Sigma_{i=1}^{n} x_i w_i \leq B \\ & x_i \in \{0, 1\} \qquad\qquad \forall i = 1, ..., n \end{aligned}$$

Due to the binary constraint on the variables x_i, this problem is not optimally solvable in polynomial time, as can be shown by reduction from Subset Sum. In general, integer linear programming is NP-complete [22]. The constraints of an ILP can be relaxed. One such method of constraint relaxation is an *LP relaxation*, in which the binary constraints are changed to a real number constraint instead. In this case, the constraint $x_i \in \{0, 1\}$ would be replaced with $x_i > 0$. Such relaxations can be useful for obtaining approximate solutions to ILP problems.

2.3 Approximation Schemes

An α-approximation of an instance of a maximization problem with optimum OPT is a solution to the instance with value at least $\alpha \cdot OPT$. In the case of minimization problems, the value of the solution should be at most $\alpha \cdot OPT$ instead. The convention is that for maximization problems $\alpha < 1$, and for minimization problems $\alpha > 1$. An α-approximation algorithm is an algorithm that yields α-approximations for some problem.

A *polynomial-time approximation scheme* (PTAS) for a maximization problem is a $(1-\epsilon)$-approximation algorithm, for each value $0 < \epsilon < 1$, with a running time of the form $n^{f(1/\epsilon)}$, for any computable

function f. This means the running time is polynomial in the size of the instance n, for fixed values of ϵ. In the case of minimization problems, a PTAS yields $(1+\epsilon)$-approximations instead.

An *efficient polynomial-time approximation scheme* (EPTAS) for a maximization problem is a $(1-\epsilon)$-approximation algorithm, for each value $0 < \epsilon < 1$, with a running time of the form $2^{f(1/\epsilon)}n^{O(1)}$, for any computable function f. That is, a running time exponential in ϵ and polynomial in the size of the instance n, independent of ϵ. Since the class of problems that admit an EPTAS is a strict subset of the class of problems that admit a PTAS, an EPTAS can also be referred to as a PTAS. This will be done throughout the rest of this thesis as well.

It can be shown that a problem admits a PTAS by providing an approximation-preserving reduction to it from a problem that admits a PTAS. An *L-reduction* is an approximation-preserving reduction that is defined as follows [23].

Definition 2.1 (L-reduction)**.** *A pair of polynomial-time functions f and g forms an L-reduction from an optimization problem $\mathcal{A}$ to an optimization problem $\mathcal{B}$, if there are constants $\alpha, \beta > 0$, such that the following conditions hold for each instance x of $\mathcal{A}$:*

- *if x is an instance of problem $\mathcal{A}$, then $f(x)$ is an instance of problem $\mathcal{B}$,*
- *$OPT(f(x)) \leq \alpha OPT(x)$,*
- *if y is a solution to $f(x)$ with value c_y, then $g(y)$ is a solution to x with value c_x,*
- *$|OPT(x) - c_x| \leq \beta |OPT(f(x)) - c_y|$.*

2.4 Treewidth

A *tree decomposition* of a graph $G = (V, E)$ is a tree in which each node contains subset of V. The nodes in a tree decomposition are also called *bags*. A tree decomposition maintains certain features of its respective graph in the form of three properties.

1. For each vertex $v \in V$, there is a bag X_i such that $v \in X_i$
2. For each edge $(u, v) \in E$, there is a bag X_i such that $u, v \in X_i$
3. If a bag X_j lies on a path between bags X_i and X_k, then $X_i \cap X_k \subseteq X_j$

Note that by these properties, every graph has a trivial tree decomposition consisting of a single bag containing all vertices. Tree decompositions give an indication of how similar a graph is to a tree. More specifically, we say that the *treewidth* of a tree decompositions is one less than the size of its largest bag. The treewidth of a graph is then the lowest treewidth over all its tree decompositions. Generally speaking, the lower the treewidth of a graph the more similar it is to a tree. A useful property of tree decompositions is that for any bag X_i in a tree decomposition T of a graph G, the vertices in X_i form a separator in G between any vertices not in a connected subgraph of bags in $T - X_i$. Given an n-vertex planar graph G of treewidth t, a tree decomposition of G with treewidth $O(t)$ and size $O(nt)$ can computed in $O(nt^2 \log t)$ time [24].

2.5 Treedepth

A *treedepth decomposition* of a graph $G = (V, E)$ is a spanning tree T on V, such that for each edge $(u, v) \in E$, either u is an ancestor of v in T, or v is an ancestor of u in T. The *treedepth* of a treedepth decomposition is the height of the spanning tree. Note that every graph has a trivial treedepth decomposition of a path graph on its vertices with treedepth $|V|$. The treedepth of a graph is the minimum treedepth over all its treedepth decompositions. This property indicates, roughly speaking, how similar a graph is to a star graph.

Treedepth decompositions and tree decompositions are related. An example of such relation is that if a graph has treewidth t, then this implies a treedepth of at most $t \log n$. A simple argument shows this is the case. A bag X_j in a tree decomposition forms a separator in its respective graph, meaning adjacent bags X_i and X_k cannot have edges between any vertices in $X_i \setminus X_j$ and $X_k \setminus X_j$. Given a tree decomposition T that is rooted at some bag X_r, such that T is a balanced tree, a

treedepth decomposition can be constructed by placing all vertices from X_r in a path graph. This process can be repeated recursively for the children of X_r, connecting each path graph to the one a layer above. Each path graph has height t, and there are at most $O(\log n)$ layers of recursion, resulting in a treedepth decomposition of treedepth $O(t \log n)$.

2.6 Computational Problems

A propositional logic formula in *conjunctive normal form* (CNF) is a conjunction of clauses that each consist of a disjunction of literals, where a literal is a variable or its negation. A CNF formula is satisfiable if there exists a true-false assignment to its variables, such that in each clause one of the literals has the value true. A 3CNF formula is a special case, in which each clause contains at most 3 literals. The satisfiability problem 3SAT is defined as follows:

Definition 2.2 (3SAT)**.** *Given a 3CNF formula ϕ, determine whether ϕ is satisfiable.*

There is an optimization variant of 3SAT called the MAX3SAT problem. Given an instance of 3SAT, the goal of MAX3SAT is to maximize the number of clauses that are satisfied. An m-clause CNF formula is α-satisfiable if there is an assignment to its variables which satisfies at least αm clauses.

Definition 2.3 (MAX3SAT)**.** *Given a 3CNF formula ϕ, find an assignment γ to the variables of ϕ that maximizes the number of satisfied clauses in ϕ.*

Definition 2.4 (Maximum Independent Set)**.** *Given a graph $G = (V, E)$, find a subset $S \subseteq V$ of maximum cardinality, such that there does not exist a pair $u, v \in S$ with $(u, v) \in E$.*

In the weighted version Maximum Weighted Independent Set the goal is to find a subset of maximum weight instead. Throughout this thesis, we refer to the planar graph version of Maximum Independent Set both as Maximum Independent Set on planar graphs and Planar Maximum Independent Set, and these are to be seen as identical.

Definition 2.5 (Minimum Dominating Set)**.** *Given a graph $G = (V, E)$, find a subset $D \subseteq V$ of minimum cardinality, such that for each vertex $v \in V \setminus D$ there exists a vertex $u \in D$, such that $(u, v) \in E$.*

Definition 2.6 (Minimum Vertex Cover)**.** *Given a graph $G = (V, E)$, find a subset $V' \subseteq V$ of minimum cardinality, such that for each edge $(u, v) \in E$ it holds that $\{u, v\} \cap V' \geq 1$.*

A minimum vertex cover in a graph is always the complement of a maximum independent set, and the complement of a maximum independent set in a graph is always a minimum vertex cover [5].

2.7 Planar Separator Theorem

The *planar separator theorem* is a theorem proven by Lipton and Tarjan, stating that any planar graph has a balanced vertex separator of bounded size [10]. The theorem is formally defined as follows.

Definition 2.7 (Planar separator theorem)**.** *Let $G = (V, E)$ be any planar graph with n vertices. Then V can be partitioned into three sets A, B, C, such that no edge has an endpoint in both A and B, the sets A and B each contain at most $(2/3)n$ vertices, and C contains at most $2\sqrt{2}\sqrt{n}$ vertices.*

This theorem can be used for solving combinatorial problems efficiently with a divide-and-conquer approach. Lipton and Tarjan use their planar separator theorem to show that a maximum independent set in an n-vertex planar graph can be computed in $2^{O(\sqrt{n})}$ time [11]. They achieve this by recursively splitting a planar graph into components separated by a set of vertices C of size $O(\sqrt{n})$, computing all independent sets on C, and checking the unions with the other components.

Lipton and Tarjan also use their planar separator theorem to prove the existence of a PTAS for the Maximum Independent Set problem in planar graphs with running time $2^{O(1/\epsilon)^2} n + O(n \log n)$

and using only polynomial space [11]. The running time is greater than Baker's approach, but the polynomial space usage is an improvement over Baker's exponential space.

2.8 r-Division

An *r-division* is a division of an n-vertex planar graph into $\Theta(n/r)$ regions of $O(r)$ vertices each and $O(\sqrt{r})$ boundary vertices each. A *boundary vertex* is a vertex that belongs to two or more regions. All other vertices in the r-division belong to only a single region and only have edges to vertices within their region. The total number of boundary vertices in an r-division is $O(n/\sqrt{r})$. The planar separator theorem can be used to create an r-division of a planar graph, as was shown by Frederickson [16]. The r-division of an n-vertex planar graph can be computed in $O(n \log n)$ time [16].

2.9 Kernels

Kernelization is a technique used for improving the efficiency of algorithms, by applying preprocessing rules on a graph prior to running an algorithm on it. The resulting graph is called a *kernel*, and is reduced in size in such a way that a solution for the kernel also resembles a solution for the original graph. The size of a kernel is bounded in some way that increases the efficiency of algorithms. Formally, a kernelization is defined as follows [25].

Definition 2.8 (Kernelization)**.** *A kernelization for a parameterized problem L is a polynomial-time algorithm that for any instance (I, k) of L outputs an equivalent instance (I', k'), such that (I', k') is a yes-instance iff (I, k) is a yes-instance. Furthermore, it should hold that $|I'|, k' \leq f(k)$, for some computable function f.*

If f is a linear function, then the size of the kernel is linear in the size of the problem solution, and the kernel is called a *linear problem kernel*.

2.10 Baker's Technique

Baker's Technique is a technique for designing polynomial-times approximation schemes on planar graphs [12]. It works by decomposing planar graphs into k-outerplanar components, and then solving a problem independently on each component. The value of k can be a function of the parameter ϵ that is used in PTAS's. Each component is defined such that it spans k levels of the graph, making it k-outerplanar. Depending on the problem at hand, adjacent components can either have some levels of overlap with one another, or a level is removed from the graph between them. A brief example is given below of Baker's technique applied to obtain a PTAS for Maximum Independent Set on planar graphs, as was presented in Baker's paper [12]. We include the proof as it provides insight into how Baker's technique works. The adaptation of this proof using tree decompositions is described in Chapter 10.2 of the book *The Design of Approximation Algorithms* [26]. This representation uses a method of obtaining tree decompositions that has a different time complexity than what was described in Section 2.4.

Theorem 2.9. *There exists a PTAS for* Maximum Independent Set *on n-vertex planar graphs running in $2^{O(1/\epsilon)}n^2$ time.*

Proof. Given a planar graph G and $0 \leq \epsilon \leq 1$, let k the smallest positive integer such that $1/k \leq \epsilon$. For each value $0 \leq i \leq k-1$, let S_i be the set of vertices of level i mod k, and let G_i be the graph obtained by removing the vertices from S_i. Note that each graph G_i consists of connected components that are at most k-outerplanar.

In each connected component, a maximum independent set can be computed in time $2^{O(k)}n^2$. This is done by dynamic programming on its tree decomposition in time $2^{O(t)}|V'|$ [26], where t is the treewidth of the connected component and $|V'|$ is the size of the tree decomposition. The treewidth is bounded by $O(k)$, since the treewidth of a k-outerplanar graph is at most $3k-1$ [27]. The size of the tree decomposition $|V'|$ is bounded by $O(n)$, though computing it takes $O(n^2)$ time [26].

The maximum independent set of each graph G_i can be found by taking the union of the maximum independent sets of its connected components. These independent sets are never in conflict with each other since the components are disconnected. The approximated maximum independent set of the original graph G is then the maximum over all graphs G_i.

This yields an independent set with weight at least a factor $(1-\epsilon)$ of the weight of the maximum independent set of the original graph. Let OPT be the weight of the maximum independent set of G. The sets S_i partition G into k disjoint sets. At least one of these sets has at most average weight. In other words: there exists a set S_j with weight at most $\frac{1}{k}OPT$. The maximum independent set obtained on G_j is then at least $(1-\frac{1}{k})OPT$. Since k was defined such that $1/k$ is at most ϵ, this results in a solution with weight at least $(1-\epsilon)OPT$.

The running time of this algorithm is $2^{O(k)}n^2$ per connected component in each graph G_i. Each vertex appears at most once across all components in a graph G_i, so the running time per G_i remains $2^{O(k)}n^2$. Since there are k graphs G_i, the total running time is $k \cdot 2^{O(k)}n^2 = 2^{O(1/\epsilon)}n^2$. □

2.11 Exponential Time Hypothesis

The Exponential Time Hypothesis (ETH) [28] is a hypothesis stating that the running time of any deterministic algorithm for 3SAT must be at least exponential in the number of variables.

Definition 2.10 (ETH)**.** *There is no $2^{o(n)}$-time algorithm for the* 3SAT *problem, where n is the number of variables.*

By the Sparsification Lemma [29], this is equivalent to the assumption that there is no such algorithm running in $2^{o(n+m)}$ time, where m is the number of clauses. ETH can be used to prove lower bounds on many satisfiability problems using reductions. More specifically, we can exclude an algorithm with running time $2^{o(f(|x|))}$ for some problem X, by providing a reduction from 3SAT to X with output size $O(g(n+m))$, where f is the inverse of g.

Aside from its direct claim about 3SAT, ETH also implies that, for any $\delta > 0$, MAX3SAT cannot be approximated to a constant factor in time $2^{O(n^{1-\delta})}$ [30].

2.12 Gap Exponential Time Hypothesis

The *Gap Exponential Time Hypothesis* (Gap-ETH) is a hypothesis introduced by Dinur [31]. This hypothesis states a that any algorithm distinguishing between an n-variable 3SAT formula with at most Dn clauses being satisfiable and not 0.9-satisfiable must run in at least 2^{cn} time, for constants $c, D > 0$. The Sparsification Lemma [29] can again be used to show that this is equivalent to the assumption that there is no such algorithm running in $O(2^{c(n+m)})$ time, where m is the number of clauses. An equivalent hypothesis called the *Exponential Time Hypothesis for Approximating* MAX3SAT (ETHA) was introduced independently by Manurangsi and Raghavendra [32]. This hypothesis states that no algorithm running in $O(2^{cm})$ time can distinguish between an m-clause 3SAT formula being satisfiable and at most $(1-\epsilon)$-satisfiable, for constants $\epsilon, c > 0$. Throughout this thesis, the formulation we use of this hypothesis is more similar to that of ETHA. However, we refer to the hypothesis as Gap-ETH, as this name for the hypothesis seems to be more widely known. The hypothesis is formally defined as follows.

Definition 2.11 (Gap-ETH)**.** *There exist constants $\epsilon, c > 0$ such that, given an m-clause* 3SAT *formula ϕ, there does not exist an $O(2^{cm})$-time algorithm that can distinguish between ϕ being satisfiable and ϕ being at most $(1-\epsilon)$-satisfiable.*

The existence of a linear-size PCP would imply that ETH and Gap-ETH are equivalent hypotheses [31]. Gap-ETH does not explicitly state that a randomized algorithm which can distinguish between the two cases of 3SAT formulas also cannot run in $O(2^{cm})$ time, for some constant $c < 0$. However, it is generally accepted to apply this lower bound to randomized algorithms as well, since the existence of such algorithm would still be very unexpected and significant.

3 Tight Lower Bound on a PTAS for Planar Maximum Independent Set

In this section, we prove that there does not exist a $2^{o(1/\epsilon)}n^{O(1)}$-time PTAS for Maximum Independent Set on planar graphs, assuming the Gap Exponential Time Hypothesis holds. This refined lower bound implies that the best known PTAS by Baker [12] running in $2^{O(1/\epsilon)}n^{O(1)}$ time is optimal in its reliance on ϵ. The proof is done by a sequence of reductions, as shown in Figure 1. Assuming Gap-ETH, there does not exist an algorithm that can distinguish between an m-clause 3SAT formula being satisfiable and at most $(1-\epsilon)$-satisfiable in $O(2^{cm})$ time, for some $\epsilon, c > 0$. A reduction to MAX2SAT implies that there also does not exist an algorithm that can distinguish between m-clause 2SAT formula being α_1-satisfiable and not α_2-satisfiable in $2^{o(m)}$ time, for some $1 > \alpha_1 > \alpha_2 > 0$. This in turn implies that there does not exist a PTAS for Matrix Tiling with $D = 2$ in $2^{o(1/\epsilon)}k^{O(1)}$ time. A final reduction proves that there does not exist a PTAS for Maximum Independent Set on planar graphs in $2^{o(1/\epsilon)}n^{O(1)}$ time. The reductions in Section 3.2 and Section 3.3 closely follow the reductions as presented by Marx [30], who proved an almost tight bound for approximating Maximum Independent Set on planar graphs assuming ETH. In our presentation of the proofs, we use a slightly modified notation, perform an example reduction, and include figures for increased clarity.

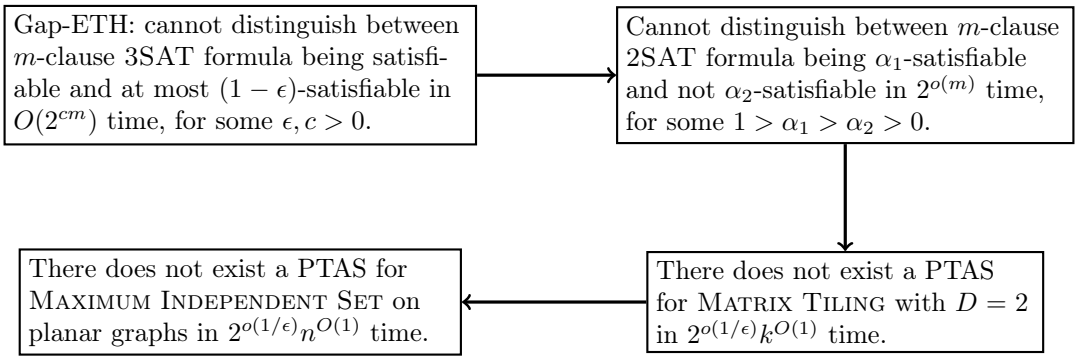

Figure 1: Sequence of reductions from Gap-ETH to Planar Maximum Independent Set.

3.1 Reducing 3SAT to MAX2SAT

In this section, we prove a reduction from 3SAT to MAX2SAT. We first show that, when running an algorithm on an n-variable 3CNF formula, we can assume that each variable appears at most a constant $d > 0$ times for an additional $2^{o(n)}$ time. This will be done in two steps, and will be important later when reducing MAX2SAT to Matrix Tiling.

We can reduce the number of clauses of an at most α_1-satisfiable 3CNF formula, obtaining an at most α_2-satisfiable 3CNF formula, where the difference between α_1 and α_2 is very small with very high probability. To prove this, we use the Multiplicative Chernoff bound, which states that

$$Pr[|X - \mu| \geq \delta\mu] \leq 2e^{-\delta^2\mu/3},$$

where X is the sum of a set of independent random variables taking values in $\{0, 1\}$, $\mu = \mathbb{E}[X]$ is the expected value of X, and $0 \leq \delta \leq 1$ is a chosen value.

Lemma 3.1. *Given an at most α_1-satisfiable* 3CNF *formula ϕ with m clauses and n variables, let ϕ' be an at most α_2-satisfiable formula with $\overline{m} = m \cdot p$ clauses in expectation, obtained by keeping each clause of ϕ independently with a probability of $p = min(cn/m, 1)$, for some constant c. Then the probability that the difference between α_1 and α_2 exceeds some small constant k is $\frac{2}{(e/2)^n}$.*

Proof. Given an at most α_1-satisfiable 3CNF formula ϕ with m clauses and n variables, let $0 \leq val_{\gamma,\phi} \leq 1$ (the value of γ w.r.t ϕ) denote the fraction of the clauses that an assignment γ

satisfies in ϕ. We obtain an at most α_2-satisfiable 3CNF formula ϕ' by keeping each clause of ϕ independently with a probability of p. The expected number of clauses in ϕ' is $\overline{m} = m \cdot p$. Let $X = val_{\gamma,\phi'} \cdot \overline{m}$ be the number of satisfied clauses in ϕ' under some assignment γ. The expected value of X is then $\mu = val_{\gamma,\phi} \cdot \overline{m}$. Then by the Chernoff bound:

$$\begin{aligned} Pr[|val_{\gamma,\phi'} \cdot \overline{m} - val_{\gamma,\phi} \cdot \overline{m}| \geq \delta \cdot val_{\gamma,\phi} \cdot \overline{m}] &= Pr[|val_{\gamma,\phi'} - val_{\gamma,\phi}| \geq \delta \cdot val_{\gamma,\phi}] \\ &\leq 2e^{-\delta^2 \cdot val_{\gamma,\phi} \cdot \overline{m}/3}. \end{aligned}$$

Let $\delta = \frac{k}{val_{\gamma,\phi}}$, where k is a small constant. Then by substituting δ we have that:

$$\begin{aligned} Pr[|val_{\gamma,\phi'} - val_{\gamma,\phi}| \geq k] &\leq 2e^{-(\frac{k}{val_{\gamma,\phi}})^2 \cdot val_{\gamma,\phi} \cdot \overline{m}/3} \\ &= 2e^{-k \cdot \frac{k}{val_{\gamma,\phi}} \cdot \overline{m}/3}. \end{aligned}$$

We have that $\frac{k}{val_{\gamma,\phi}} \geq k$, since $0 \leq val_{\gamma,\phi} \leq 1$. By setting $p = \min(cn/m, 1)$ we have that $\overline{m} \leq cn$, for some constant c. We can always choose the values of k and c such that $k^2 \cdot c = 3$. For example, we can set $k = 0.0001$ and $c = 3000000$. Without loss of generality, let us assume that this is the case to simplify the inequality. Substituting $\frac{k}{val_{\gamma,\phi}}$ and $\overline{m}$ in the inequality above gives:

$$\begin{aligned} Pr[|val_{\gamma,\phi'} - val_{\gamma,\phi}| \geq k] &\leq 2e^{-k^2 \cdot cn/3} \\ &= 2e^{-n} \\ &= \frac{2}{e^n}. \end{aligned}$$

The maximum of $val_{\gamma,\phi}$ and $val_{\gamma,\phi'}$ over all assignments γ is α_1 and α_2 respectively. There are 2^n possible assignments γ. By the union bound, the probability that for any γ the difference between $val_{\gamma,\phi}$ and $val_{\gamma,\phi'}$ exceeds k is at most:

$$\begin{aligned} Pr[|\alpha_1 - \alpha_2| > k] &\leq \frac{2}{e^n} \cdot 2^n \\ &= \frac{2}{(e/2)^n}. \end{aligned}$$

□

Using Lemma 3.1 we now show that, when running an algorithm on a 3CNF formula, we can assume each variable occurs at most a constant $d > 0$ times in exchange for some additional time usage. We formalize this in the following lemma.

Lemma 3.2. *When running an algorithm on a* 3CNF *formula ϕ with n variables, we can assume that each variable appears in ϕ at most a constant $d > 0$ times, for an additional factor $2^{o(n)}$ time.*

Proof. Let ϕ be an at most α_1-satisfiable 3CNF formula with n variables and m clauses. Let ϕ' be an at most α_2-satisfiable 3CNF formula with $\overline{m} = m \cdot p$ clauses in expectation, obtained by keeping each clause of ϕ with a probability of $p = min(cn/m, 1)$, for some constant c. By Lemma 3.1, the difference between α_1 and α_2 is at most a small constant, with probability $1 - \frac{2}{(e/2)^n}$. Let A be an algorithm that runs on ϕ'. There are at most $3\overline{m}/d$ variables that appear more than d times. We can branch on these variables by fixing their value as either 0 or 1, which creates $2^{3\overline{m}/d}$ subproblems. We have that $2^{3\overline{m}/d} = 2^{o(n)}$, since for each 2^{bn} $(b > 0)$, there is a constant d such that $3\overline{m}/d \leq 3cn/d < bn$. We run A on each of the $2^{o(n)}$ subproblems. When running A on a subproblem we can assume each variable appears at most d times. □

The results in the rest of the section rely on Lemma 3.2, which uses probability. It should therefore be noted that the results in this section hold only with high probability. Gap-ETH is still applicable, as was described in Section 2.12.

We now show that a fast algorithm for MAX2SAT would also be fast at distinguishing between a 3SAT formula being satisfiable and being at most $(1-\epsilon)$-satisfiable. This is done by a reduction from 3SAT to MAX2SAT. A formula is *simple* if it does not contain unsatisfiable or duplicated clauses. Since the clauses are disjunctions, the only unsatisfiable clauses are empty clauses. A clause is a duplicate of another clause if it contains the same literals in the same order. For example, the duplicates of a clause $(a \vee b)$ are: $(\neg a \vee b)$, $(a \vee \neg b)$, and $(\neg a \vee \neg b)$.

As a first attempt, let us follow a reduction by Garey et al. [33]. For each clause $C_i = (a \vee b \vee c)$ in a 3CNF formula ϕ, we create the following set of clauses: $S_i = \{(a), (b), (c), (r_i), (\neg a \vee \neg b), (\neg a \vee \neg c), (\neg b \vee \neg c), (a \vee \neg r_i), (b \vee \neg r_i), (c \vee \neg r_i)\}$. An assignment that satisfies C_i can be extended to an assignment that satisfies 7 of the clauses in S_i, and an assignment that does not satisfy C_i can be extended to an assignment that satisfies at most 6 of the clauses in S_i. Note that if there are clauses C_i and C_j that both contain some variable this will result in duplicated clauses. We modify this reduction to prove the following theorem.

Theorem 3.3. *There are constants $1 > \alpha_1 > \alpha_2 > 0$ such that if there is an algorithm that can distinguish between α_1-satisfiable and not α_2-satisfiable* 2SAT *formulas in time $2^{o(m)}$ (where m is the number of clauses), then Gap-ETH fails. Furthermore, we can assume that the formula is simple and a variable appears at most d times for some constant $d > 0$.*

Proof. We prove that this is the case by reduction from 3SAT to MAX2SAT. Given a 3SAT formula ϕ with m clauses that is either satisfiable or at most $(1-\epsilon)$-satisfiable, for some $\epsilon > 0$. By Lemma 3.2, we can assume each variable appears at most a constant $d > 0$ times, for an additional factor $2^{o(n)}$ time.

For each clause $C_i = (a \vee b \vee c)$ in ϕ we create the following set of clauses: $S_i = \{(a \vee \neg x_i), (b \vee \neg y_i), (c \vee \neg z_i), (x_i), (y_i), (z_i)(r_i), (\neg x_i \vee \neg y_i), (\neg x_i \vee \neg z_i), (\neg y_i \vee \neg z_i), (x_i \vee \neg r_i), (y_i \vee \neg r_i), (z_i \vee \neg r_i)\}$. This will never result in duplicated clauses since the original variables only appear in a clause with variables that are unique to the clause C_i. An assignment to the variables that satisfies C_i can be extended to an assignment that satisfies 10 of the clauses in S_i, and an assignment that does not satisfy C_i can be extended to an assignment that satisfies at most 9 of the clauses in S_i. A conjunction of the clauses in all sets S_i yields a formula ϕ' of $m' = 13m$ clauses. This formula ϕ' is simple because it does not contain unsatisfiable or duplicated clauses.

Let $\mathcal{A}$ be an algorithm that distinguishes between α_1-satisfiable and not α_2-satisfiable 2SAT formulas in time $2^{o(m')}$, for all $1 > \alpha_1 > \alpha_2 > 0$. We can run $\mathcal{A}$ on ϕ' with $\alpha_1 = \frac{10}{13}$ and $\alpha_2 = \frac{9}{13} + (1-\epsilon)/13 + v$, where $v < \frac{1}{m}$ is some small value [1]. If ϕ' is α_1-satisfiable, then ϕ is satisfiable. If ϕ' is not α_2-satisfiable, then ϕ is at most $(1-\epsilon)$-satisfiable. This means $\mathcal{A}$ can be used to distinguish between a 3SAT formula being satisfiable and being at most $(1-\epsilon)$-satisfiable in $O(2^{cm})$ time for each $\epsilon, c > 0$, which contradicts Gap-ETH. □

3.2 Reducing MAX2SAT to Matrix Tiling

In this section, we show a reduction from MAX2SAT to a special case of a problem called Matrix Tiling with $D = 2$. We will also prove a lower bound on a PTAS for Matrix Tiling with $D = 2$.

The Matrix Tiling problem is similar to the more commonly used Grid Tiling problem, but it allows for solutions that satisfy the constraints for only some of the cells by using ★ values as wildcards. The Matrix Tiling problem is defined by Marx as follows [30].

Definition 3.4 (Matrix Tiling)**.** *Given integers k, D, and k^2 nonempty sets $S_{i,j} \subseteq Z_D \times Z_D$ $(1 \leq i, j \leq k)$, where Z_D denotes the set $\{0, 1, ..., D-1\}$.*

[1] We add v, since otherwise ϕ' being not α_2-satisfiable would prove ϕ is not $(1-\epsilon)$-satisfiable, as opposed to proving ϕ is at most $(1-\epsilon)$-satisfiable.

For each $1 \leq i, j \leq k$, find a value $s_{i,j} \in S_{i,j} \cup \bigstar$ such that:

- *If $s_{i,j} = (a_1, a_2)$ and $s_{i,j+1} = (b_1, b_2)$, then $a_1 = b_1$.*
- *If $s_{i,j} = (a_1, a_2)$ and $s_{i+1,j} = (b_1, b_2)$, then $a_2 = b_2$.*

And maximize the number of pairs (i, j) $(1 \leq i, j \leq k)$ with $s_{i,j} \neq \bigstar$.

Note that any Matrix Tiling instance has an optimum of at least $k^2/2$, which can be obtained by selecting an arbitrary value in each cell with set $S_{i,j}$ for which i and j are either both odd or both even, and selecting $\bigstar$ in all other cells.

(1,2)	(3,1) (1,0) (2,3)	(1,0) (2,2)
(2,2) (0,1)	(0,3) (1,1)	(0,2) (3,0)
(3,1) (0,2)	(0,2) (1,1) (3,3)	(0,0) (3,2)

$i \uparrow$

$j \longrightarrow$

Figure 2: Example Matrix Tiling instance.

An example of a Matrix Tiling instance with $k = 3$ and $D = 4$ is shown in Figure 2. In this instance, the cell in the bottom-left of the matrix contains the set $S_{1,1} = \{(3, 1), (0, 2)\}$. Cells that are horizontally adjacent should have the first component of their selected pair in common, and cells that are vertically adjacent should have the second component of their selected pair in common. The optimum of this instance is 8, since any solution has at most 8 cells of which the selected value is not $\bigstar$. To verify this, first observe that $s_{3,1} = (1, 2)$, since it is the only pair in $S_{3,1}$. Then for $S_{1,1}$ and $S_{2,1}$ we must select $s_{1,1} = (0, 2)$ and $s_{2,1} = (2, 2)$ respectively, to avoid selecting a $\bigstar$. Now note that in $S_{2,2}$ we must select a pair that has 2 as its first component, since it is horizontally adjacent to $s_{2,1} = (2, 2)$. However, $S_{2,2}$ does not contain any such pair, and so we set $s_{2,2} = \bigstar$. No further $\bigstar$'s are required to select values in the rest of the cells.

3.2.1 Reduction Proof

We show that an instance of MAX2SAT can be reduced to a special case of Matrix Tiling with $D = 2$ in the theorem below. The reduction closely follows the reduction by Marx [30], but uses slightly different notation. We also include an example reduction after the proof.

Theorem 3.5. *There is a polynomial-time algorithm that, given a simple* 2SAT *formula ϕ with m clauses where each variable appears in at most d clauses, constructs an instance of* Matrix Tiling *with parameters $k = O(d^2 m)$ and $D = 2$ such that if t is the minimum number of unsatisfied clauses in ϕ, then the optimum of the constructed instance is $k^2 - t$.*

Proof. Let ϕ be a simple 2SAT formula with m clauses and n variables ($n \leq 2m$), where each variable appears in at most d clauses.

Let $\delta = x_1, ..., x_n$ be an ordering on the variables of ϕ. Let $d_i \leq d$ be the number of variables that a variable x_i appears together in a clause with, and let $z = 4d$. We are going to replace each variable x_i in δ with the segment of x_i. The *segment* of x_i is an ordered sequence of variables, containing x_i itself and $2zd_i$ new variables $x_{s,i,l}$. The label $1 \leq s \leq n$ corresponds to the index in δ of a variable x_s that appears together in a clause with x_i. The label i corresponds to the index in δ of the variable x_i that the segment belongs to. The label $1 \leq l \leq 2z$ indicates that the variable is the l'th of that variable in the segment. The segment is constructed in such a way that all variables with $l \leq z$ are before x_i in the segment, and all variables with $z + 1 \leq l \leq 2z$ are after x_i in the segment. We replace each variable in the ordering δ with its segment to obtain a new sequence X of length $k \leq (2zd + 1)n$.

We construct an instance of Matrix Tiling with k as defined above and $D = 2$. The rows and columns of the matrix are indexed by the variables in sequence X or by the indices of these variables in X. We say that each variable $x_{s,i,l}$ is a *copy* of x_s. A variable in X *represents* a variable x_s in δ if it is a copy of x_s or x_s itself. We define the sets $S_{i,j}$ according to the following rules:

1. If ϕ contains a clause $(x_i \vee x_j)$, then the set S_{x_i,x_j} contains those pairs from $Z_D \times Z_D = \{(0,0),(0,1),(1,0),(1,1)\}$ that correspond to an assignment that satisfies $(x_i \vee x_j)$.
2. If row i and column j correspond to two variables in X that represent that same variable from δ, then $S_{i,j} = \{(0,0),(1,1)\}$.
3. Otherwise $S_{i,j} = \{(0,0),(0,1),(1,0),(1,1)\}$.

This concludes the construction of the Matrix Tiling instance. All steps can be done in polynomial time. We will now show that a MAX2SAT instance with optimum $m-t$ corresponds to a Matrix Tiling instance with optimum $k^2 - t$. For the other direction, we show that a Matrix Tiling solution with at most t ★'s corresponds to an assignment of ϕ that satisfies at least $m-t$ clauses.

Assume that γ is an optimal assignment to the variables of ϕ that satisfies $m - t$ clauses. The pairs $s_{i,j}$ are selected in accordance with the variables from ϕ that the i'th and j'th variables in X represent. If the i'th variable in X represents x_1 and the j'th variable in X represents x_2, then $s_{i,j} = (\gamma(x_1), \gamma(x_2))$. If the set $S_{i,j}$ was defined by rule 1, then it must contain the pair $(\gamma(x_1), \gamma(x_2))$, unless the values of x_1 and x_2 correspond to an assignment that does not satisfy the clause. In this case we set $s_{i,j} = ★$. If the set $S_{i,j}$ was defined by rule 2 or rule 3, then it always contains $(\gamma(x_1), \gamma(x_2))$. Since the number of unsatisfied clauses is t, the number of ★'s is at most t.

For the other direction, let us assume we have a solution for Matrix Tiling with value at least $k^2 - t$, where t is the number of ★'s. We say that a variable of X is *bad* if the row or column corresponding to that variable contains at least one ★. Otherwise we say that the variable of X is *good*. If the i'th variable of X is good, then row i contains no ★'s, meaning the selected pairs in row i must have their first component in common. Since we have that $S_{i,i} = \{(0,0),(1,1)\}$, this value is also shared in the second component of column i. We associate with each good variable its shared value.

We say that a variable of ϕ is *spoiled* if at least z of its copies are bad. We construct an assignment γ of ϕ as follows: if a variable x_i of ϕ is spoiled then we set its value in γ arbitrarily. Otherwise, we set it to the value associated with the good copies of x_i. We now prove that this constructed assignment γ satisfies at least $m - t$ clauses of ϕ.

The *cross* of a clause $(x_i \vee x_j)$ consists of those cells in row x_i where the column represents x_j and those cells in column x_j where the row represents x_i. Crosses of different clauses are disjoint since there are no duplicated clauses.

Claim 1. *For every unsatisfied clause $(x_i \vee x_j)$, at least one of the following holds:*

1. *the cell in row x_i of column x_j contains a ★*

2. *x_i or x_j is spoiled*

3. *the cross of the clause contains at least two ★'s*

Proof. Consider a clause $(x_i \vee x_j)$ such that (1)-(3) do not hold. Less than z of x_i's copies are bad, since x_i is not spoiled (2). This means there is an $1 \leq l_1 \leq z$ such that variable x_{i,j,l_1} is good and an $z+1 \leq l_2 \leq 2z$ such that variable x_{i,j,l_2} is good. There is at most one ★ in row x_i between column x_{i,j,l_1} and x_{i,j,l_2} (3). This means in row x_i there is either no ★ between columns x_{i,j,l_1} and x_j or between columns x_j and x_{i,j,l_2}. Since $s_{x_i,x_j} \neq$ ★ (1), in either case s_{x_i,x_j} must have $\gamma(x_i)$ as its first component. The same argument can be made to show that s_{x_i,x_j} must have $\gamma(x_j)$ as its second component. The pair $(\gamma(x_i), \gamma(x_j))$ is in S_{x_i,x_j}, by how the sets $S_{i,j}$ are defined, which means that the clause $(x_i \vee x_j)$ is satisfied by γ. This shows that if none of (1)-(3) are true for some clause, then the clause must be satisfied. This implies that if a clause is not satisfied, at least one of (1)-(3) must hold. □

Claim 2. *At most t clauses of ϕ are not satisfied by γ.*

Proof. Let t_0 be the number of clauses for which (1) holds. These t_0 ★'s do not affect whether (2) or (3) holds for other clauses, so we only need to investigate the effect of the remaining $t - t_0$ ★'s.

A ★ can make at most two variables of X bad, so the number of spoiled variables in ϕ is at most $2(t-t_0)/z$. As a result, (2) holds for at most $d \cdot 2(t-t_0)/z = d \cdot 2(t-t_0)/4d = (t-t_0)/2$ clauses.

Additionally, (3) holds for at most $(t-t_0)/2$ clauses. This is because the crosses are disjoint, and two ★'s are required to make (3) hold for a clause.

In total the number of unsatisfied clauses is at most $t_0 + (t-t_0)/2 + (t-t_0)/2 = t_0 - 0.5t_0 - 0.5t_0 + 0.5t + 0.5t = t$. □

In summary, we have proven that there exists a polynomial-time algorithm that reduces a simple 2SAT formula ϕ with m clauses, where each variable appears in at most d clauses, to an instance M of Matrix Tiling with $D = 2$ and $k = O(d^2 m)$. We also showed that if ϕ has an assignment γ that satisfies $m - t$ clauses, then M has an optimum of at least $k^2 - t$, and that a solution for M with value $k^2 - t$ can be transformed into an assignment of ϕ that satisfies at least $m - t$ clauses. □

3.2.2 Example Reduction

In this example, we slightly simplify our notation of the variables in the segments. We now use $x_{ij,l}$ to express what was previously denoted as $x_{i,j,l}$. For example, we use $a_{2,5}$ to express $x_{1,2,5}$, where 1 is the index of a in the ordering δ. If 2 is the index of b in δ, then $a_{2,5}$ is the 5th copy of a in the segment of b.

Let $\phi = (a \vee b) \wedge (\neg b \vee a) \wedge (\neg a \vee c) \wedge (\neg c \vee \neg a)$ be a 2SAT formula. This formula does not contain duplicated clauses by our definition of a duplicated clause. We have that $d = 4$, since each variable appears in at most 4 clauses. Note that an assignment of the variables satisfies at most 3 clauses. Let $\delta = a, b, c$ be an ordering on the variables. We have that $z = 16$. The segments of the variables are then:

- segment of a: $b_{1,1}, ..., b_{1,16}, c_{1,1}, ..., c_{1,16}, a, b_{1,17}, ..., b_{1,32}, c_{1,17}, ..., c_{1,32}$
- segment of b: $a_{2,1}, ..., a_{2,16}, b, a_{2,17}, ..., a_{2,32}$
- segment of c: $a_{3,1}, ..., a_{3,16}, c, a_{3,17}, ..., a_{3,32}$

By concatenating the segments we obtain the sequence X of length $k = 131$. We define the sets $S_{i,j}$, for $1 \leq i, j \leq k$, according to the rules described in Theorem 3.5. An example of each rule being applied is given:

1. $S_{a,c} = \{(0,0), (0,1), (1,1)\}$, since there is a clause $(\neg a \vee c)$.

2. $S_{a,a_{2,7}} = \{(0,0),(1,1)\}$, since a and $a_{2,7}$ both represent a.
3. $S_{b_{1,1},a_{2,18}} = \{(0,0),(0,1),(1,0),(1,1)\}$, since $b_{1,1}$ and $a_{2,18}$ represent different variables, and there is no clause $(b_{1,1} \vee a_{2,18})$.

The resulting Matrix Tiling instance is shown in Figure 3.

We now give an example of obtaining a solution to a Matrix Tiling instance from a solution to a MAX2SAT instance, resuming the example we used for the reduction. Let $\gamma = a:1, b:1, c:0$ be an assignment of the variables in ϕ. This assignment satisfies 3 clauses. We select a pair in each cell as described in Theorem 3.5. For the cell in row a column b, we should select the pair $(1,1)$. This pair is in the set $S_{a,b}$, since this set contains all pairs corresponding to satisfying assignments of $(a \vee b)$. For the cell in row a column c, we should select the pair $(1,0)$. This pair is not in the set $S_{a,c}$ since it does not correspond to a satisfying assignment of $(\neg a \vee c)$, so we select a ★ instead. For all other cells, we select the values of the variables that the respective row and column indices represent. This is always possible because either the row and column represent the same pair, meaning they have the same value, or they represent different pairs, meaning any combination of values can be selected in the cell. Since there is 1 clause not satisfied by γ, the number of ★'s in the corresponding Matrix Tiling solution is at most 1.

3.2.3 Matrix Tiling Lower Bound

Using the reduction of Theorem 3.5 in combination with Theorem 3.3, we can prove a lower bound on the running time of a PTAS for Matrix Tiling with $D = 2$.

Theorem 3.6. *There does not exist a PTAS for* Matrix Tiling *with* $D = 2$ *running in* $2^{o(1/\epsilon)}k^{O(1)}$ *time, unless Gap-ETH fails.*

Proof. Let ϕ be a 2SAT formula with m clauses. We can use the algorithm from Theorem 3.5 to create an instance M of Matrix Tiling with $k = O(d^2 m)$ and $D = 2$. If ϕ is α_1-satisfiable, then the optimum M is at least $k^2 - (1-\alpha_1)m$. If ϕ is not α_2-satisfiable, then the optimum of M is less than $k^2 - (1-\alpha_2)m$.

Let $\mathcal{A}$ be a PTAS for Matrix Tiling with $D = 2$ running in $2^{o(1/\epsilon)}k^{O(1)}$ time. To make the desired distinction, we want the following: if an instance of Matrix Tiling has an optimum of at least $k^2 - (1-\alpha_1)m$, then an $(1-\epsilon)$-approximation should yield a solution with value at least $k^2 - (1-\alpha_2)m$. It should then hold that:

$$(1-\epsilon)(k^2 - (1-\alpha_1)m) \geq k^2 - (1-\alpha_2)m.$$

We can use $\mathcal{A}$ on M with $\epsilon \leq \frac{(\alpha_1-\alpha_2)m}{k^2-(1-\alpha_1)m}$ to distinguish between M having an optimum of at least $k^2 - (1-\alpha_1)m$ and an optimum of less than $k^2 - (1-\alpha_2)m$, in $2^{o(1/\epsilon)}k^{O(1)}$ time. This in turn distinguishes between ϕ being α_1-satisfiable and being not α_2-satisfiable in $2^{o(1/\epsilon)}k^{O(1)} = 2^{o(m)}$ time, which by Theorem 3.3 contradicts Gap-ETH. □

3.3 Reducing Matrix Tiling to Planar Maximum Independent Set

In this section, we show an L-reduction from Matrix Tiling with $D = 2$ to Planar Maximum Independent Set, and use this reduction to prove a lower bound on a PTAS for Planar Maximum Independent Set. We conclude this section with some additional notes on what this proof means for the optimality of the exact algorithm, when performing $(1-\epsilon)$-approximations for specific values of ϵ.

Theorem 3.7. *There is a polynomial-time algorithm that, given an instance* M *of* Matrix Tiling *with* $D = 2$*, constructs an instance of* Planar Maximum Independent Set *such that if* $k^2 - t$ *is the optimum of* M*, where* t *is the number of* ★*'s, then the optimum of the constructed instance is* $(c+1)k^2 - t$.

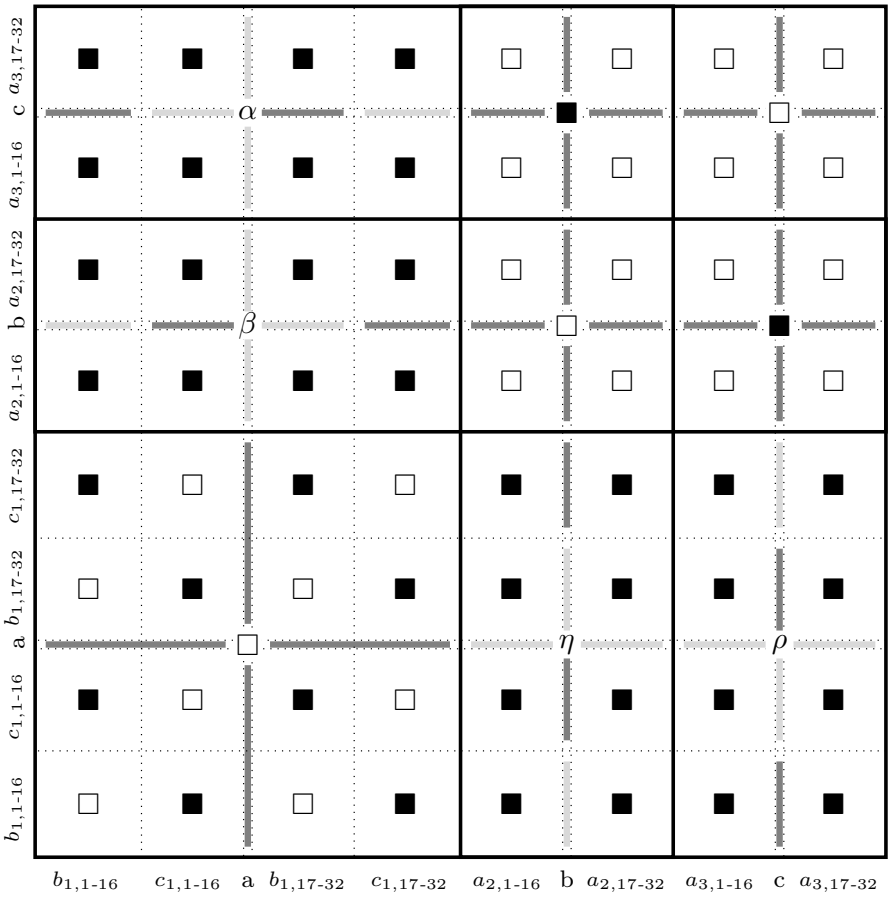

Figure 3: Example constructed MATRIX TILING instance. Shorthand notation is used on the axes; e.g. $b_{1,1-16}$ denotes $b_{1,1}, ..., b_{1,16}$. Black lines separate segments, and dotted lines separate parts of segments. The sets $S_{i,j}$ of individual cells are not shown, with the exception of cells of which both row i and column j correspond to a variable in ϕ. The sets of other cells are given per region of cells. The square symbols denote the following sets: ■ $= \{(0,0),(0,1),(1,0),(1,1)\}$, and □ $= \{(0,0),(1,1)\}$. Dark gray-shaded regions are equivalent to regions with ■, and light gray-shaded regions are equivalent to regions with □. Cells that correspond to a clause in ϕ have unique letters with the following meaning: $\alpha = \neg c \vee \neg a$, $\beta = \neg b \vee a$, $\eta = a \vee b$, and $\rho = \neg a \vee c$. The sets $S_{i,j,}$ corresponding to these cells contain those pairs from $Z_D \times Z_D$ that satisfy the respective clause; e.g. the set in cell α is $\{(0,0),(0,1),(1,0)\}$.

Proof. Each of the k^2 cells in an instance of Matrix Tiling with $D = 2$ can be represented by a gadget. These gadgets are constructed in such a way that an independent set in a gadget can be slightly greater when it corresponds to a pair in the corresponding cell. The gadgets are then connected in a way that that enforces the Matrix Tiling constraints.

The foundation of the gadget consists of a set of 8 vertices on its outer border in the following order: $B = \{a_0, a_1, b_0, b_1, c_0, c_1, d_0, d_1\}$. The edges between these vertices are: $(a_0, a_1), (b_0, b_1), (c_0, c_1)$, and (d_0, d_1). For each gadget $G_{i,j,}$ the set $B_{i,j}$ is a copy of B. The center of a gadget $G_{i,j}$ contains a set of fully connected vertices $C_{i,j} \subseteq \{v_{(0,0)}, v_{(0,1)}, v_{(1,0)}, v_{(1,1)}\}$ corresponding to the pairs in $S_{i,j}$. Note that $C_{i,j}$ cannot be empty since $S_{i,j}$ is a nonempty set. A subset $B'_{i,j} \subset B_{i,j}$ *represents* a pair $(x, y) \in Z_2 \times Z_2$ if $B'_{i,j} = \{a_x, c_x, b_y, d_y\}$. For example, the set $\{a_0, c_0, b_1, d_1\}$ represents the pair $(0, 1)$. There are edges between a vertex $u \in B_{i,j}$ and a vertex $v_{(x,y)} \in C_{i,j}$ if u is not part of the set of vertices that represents (x, y). For example, there is an edge between a_0 and $v_{(1,0)}$ since $(1, 0)$ is represented by $\{a_1, c_1, b_0, d_0\}$ which does not contain a_0. An example of a gadget $G_{i,j}$ can be seen in Figure 4, with $S_{i,j} = \{(0, 0), (1, 0), (1, 1)\}$.

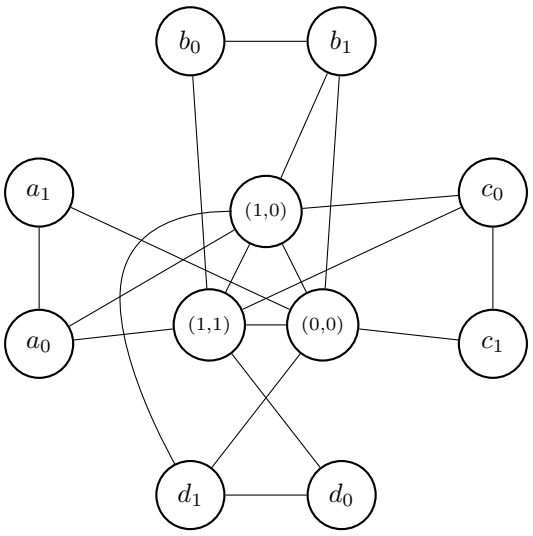

Figure 4: Example gadget of a Planar Maximum Independent Set instance.

Note that this gadget is not a planar graph, since it contains a subgraph that is a subdivision of $K_{3,3}$. It can be made planar by using planar crossover gadgets of constant size [33]. A crossover gadget of 22 vertices can be seen in Figure 5. The crossover gadget is constructed in such a way that its maximum independent set can always include either V_1 (resp. V_2) or V'_1 (resp. V'_2), but does not need to include both.

Table 1 shows the size of the maximum independent set in the crossover gadget when different combinations of V_1, V'_1, V_2, and V'_2 are included. An independent set of size 9 can be achieved by including only V_1 or V'_1 and only V_2 or V'_2. When an edge crossing is replaced by a crossover gadget, this enforces the behaviour that only one of the endpoints of the replaced edges are selected. As such, introducing them does not affect the maximum independent set in the rest of the graph.

	$\varnothing$	$V_1 \mid V'_1$	$V_1 \cup V'_1$
$\varnothing$	7	8	8
$V_2 \mid V'_2$	8	9	9
$V_2 \cup V'_2$	7	8	9

Table 1: Size of a maximum independent set in the crossover gadget with specific vertices included.

All gadgets $G_{i,j}$ can use the same number of crossover gadgets to ensure they are of the same size. Let g be the number of crossover gadgets used in each $G_{i,j}$. The way a gadget $G_{i,j}$ is constructed means that if the vertices $B'_{i,j} \subset B_{i,j}$ selected in an independent set represent a pair

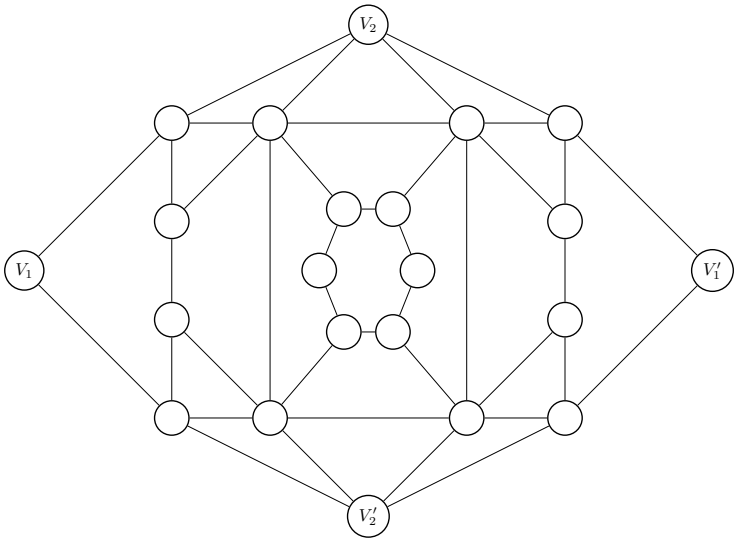

Figure 5: Crossover gadget for Maximum Independent Set.

from $S_{i,j}$, then there is a $v_{(x,y)} \in C_{i,j}$ such that no vertex from $B'_{i,j}$ is adjacent to $v_{(x,y)}$. This means the independent set can be extended by including $v_{(x,y)}$. The maximum independent set of each gadget $G_{i,j}$ is then $c = 4 + 9g$, or $c + 1$ if the vertices in $B'_{i,j}$ represent a pair in the set $S_{i,j}$. There always exists some $B'_{i,j}$ for which this is the case, since the sets $S_{i,j}$ cannot be empty. Additionally, if a gadget $G_{i,j,}$ has an independent set of size $c + 1$, then it must be the case that its set $B'_{i,j}$ represents a pair in $S_{i,j}$, since otherwise at least one vertex would be adjacent to the added vertex from $C_{i,j}$. By the above, the following claim must hold.

Claim 3. *For each gadget $G_{i,j}$ the following holds:*

1. *The size of the maximum independent set is $c + 1$.*
2. *If $B'_{i,j}$ represents a pair in $S_{i,j}$, then the independent set can be extended to size $c + 1$.*
3. *If $B'_{i,j}$ does not represent a pair in $S_{i,j}$, then the independent set can be extended to size c.*
4. *If the gadget has an independent set of size $c+1$, then its $B'_{i,j}$ must represent a pair in $S_{i,j}$.*

The constraints of Matrix Tiling are enforced by creating edges between c_0 and c_1 of $G_{i,j}$ and a_1 and a_0 of $G_{i,j+1}$ respectively, for each $1 \leq i \leq k$, $1 \leq j < k$. Similarly, there are edges between d_0 and d_1 of $G_{i,j}$ and b_1 and b_0 of $G_{i+1,j}$ respectively, for each $1 \leq i < k$, $1 \leq j \leq k$. This concludes the construction of the Maximum Independent Set instance. A part of the constructed instance can be seen in Figure 6. The center of each gadget depends on the corresponding cell, but will look similar to what is shown in Figure 4.

We now show that if the optimum of a Matrix Tiling instance with $D = 2$ is $k^2 - t$, then the optimum of the corresponding constructed Planar Maximum Independent Set instance is $(c + 1)k^2 - t$. Let M be an instance of Matrix Tiling with $D = 2$ and an optimum of $k^2 - t$, where t is the number of ★'s, and let G be an instance of Planar Maximum Independent Set obtained by reduction from M.

For each selected pair $s_{i,j} \in S_{i,j} \neq$ ★, we include in the independent set of $G_{i,j}$ those vertices in $B_{i,j}$ that represent $s_{i,j}$. By Claim 3:2, the independent set of this gadget can then be extended to size $c + 1$. Note that the independent sets in neighbouring gadgets chosen this way are not in conflict with each other. Selected pairs $s_{i,j}$ and $s_{i,j+1}$ must have their first component in common. Then in both $G_{i,j}$ and $G_{i,j+1}$ either a_0 and c_0 or a_1 and c_1 are selected. There is no edge between c_0 of $G_{i,j}$ and a_0 of $G_{i,j+1}$, nor is there an edge between c_1 of $G_{i,j}$ and a_1 of $G_{i,j+1}$, meaning the

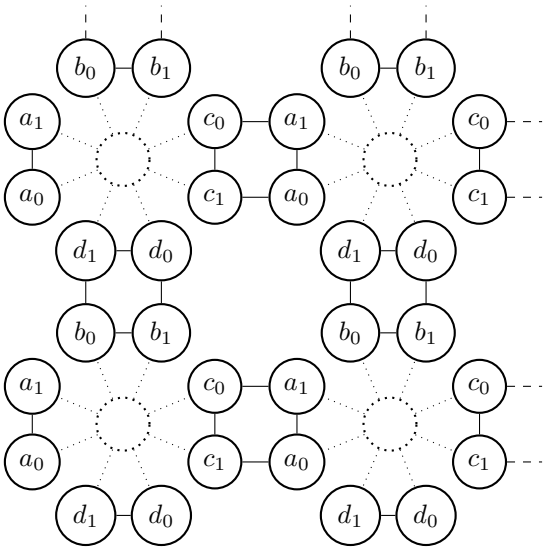

Figure 6: Partially constructed Maximum Independent Set instance.

selected vertices are not adjacent. The same argument can be made to show that selected vertices in $G_{i,j}$ and $G_{i+1,j}$ are not adjacent.

For each selected pair $s_{i,j} \in S_{i,j} = \bigstar$, we include in the independent set of $G_{i,j}$ those vertices in $B_{i,j}$ that do not conflict with neighbouring gadgets. There are always 4 such vertices, since a neighbouring gadget cannot, for example, have both a_0 and a_1 selected in its independent set. By Claim 3:3, we can extend the independent set of this gadget to size c. The resulting independent set of G is then of size $(c+1)(k^2-t)+ct = (c+1)k^2 - t$.

For the other direction, let I be an independent set of size $(c+1)k^2 - t$. By Claim 3:1, each gadget has an independent set of size at most $c+1$. If a gadget $G_{i,j}$ has an independent set of size $c+1$, we have $B'_{i,j} = \{a_x, c_x, b_y, d_y\}$ and set $s_{i,j} = (x, y)$. By Claim 3:4, this pair must be in $S_{i,j}$. There are $k^2 - t$ gadgets for which this is the case. For the other t gadgets we set the values in their corresponding cells to $\bigstar$. This results in a Matrix Tiling solution with value at least $k^2 - t$. □

Theorem 3.8. *There does not exist a PTAS for* Maximum Independent Set *on planar graphs running in* $2^{o(1/\epsilon)}n^{O(1)}$ *time, unless Gap-ETH fails.*

Proof. We show that this is the case by L-reduction from Matrix Tiling with $D = 2$ to Planar Maximum Independent Set. Let M be an instance of Matrix Tiling with $D = 2$ of size $O(k^2)$, and with optimum $k^2 - t$. By Theorem 3.7, we can construct an instance G of Planar Maximum Independent Set of size $n = O(k^2)$, and with optimum $(c+1)k^2 - t$. Here, t is the number of tiles with a $\bigstar$ or the number of gadgets with a maximum independent set of size c. Let A be a PTAS for Planar Maximum Independent Set running in $2^{o(1/\epsilon)}n^{O(1)}$ time. Any solution to M that is obtained by approximating the maximum independent set on G, and then transforming the solution to M, should have a value at least that of approximating a solution to M directly. Starting with an approximation on G with value

$$(1-\epsilon)((c+1)k^2 - t) = (c+1)k^2 - (\epsilon(c+1)k^2 + (1-\epsilon)t),$$

we obtain a solution to M with value

$$k^2 - (\epsilon(c+1)k^2 + (1-\epsilon)t) = (1-\epsilon)(k^2 - t) - \epsilon c k^2.$$

It should then hold that

$$(1-\epsilon)(k^2 - t) - \epsilon c k^2 \geq (1 - b\epsilon)(k^2 - t),$$

for some constant $b > 0$. Since any Matrix Tiling instance has an optimum of at least $k^2/2$, we have that $k^2 - t \geq k^2/2$. Then the inequality above holds for any constant $b \geq 2c + 1$, implying that any given $\epsilon > 0$ can be divided by $2c+1$ to obtain the desired approximation. This means A can be used on G to obtain $(1-\epsilon)$-approximations of M in $2^{o(1/\epsilon)}n^{O(1)} = 2^{o(1/\epsilon)}k^{O(1)}$ time, which by Theorem 3.6 contradicts Gap-ETH. □

We have seen that there does not exist a PTAS for Planar Maximum Independent Set running in $2^{o(1/\epsilon)}n^{O(1)}$ time. However, this does not exclude an algorithm that yields $(1-\epsilon)$-approximations for only specific values of ϵ in $2^{o(1/\epsilon)}n^{O(1)}$ time. In the proof of Theorem 3.6, we showed that a PTAS for Matrix Tiling with $D = 2$ can distinguish between a 2SAT formula being α_1-satisfiable and not α_2-satisfiable using any $\epsilon \leq \frac{(\alpha_1-\alpha_2)m}{k^2-(1-\alpha_1)m}$, for constants $1 > \alpha_1 > \alpha_2 > 0$. Recall that $k = O(d^2m)$, which is $O(m)$, since d is a constant. Then we have that

$$\begin{aligned}\epsilon &\leq \frac{(\alpha_1 - \alpha_2)m}{k^2 - (1-\alpha_1)m} \\ &= O(\frac{1}{k}) \\ &= O(\frac{1}{\sqrt{n}}),\end{aligned}$$

which implies that, independently of the values of α_1 and α_2, there is some $\epsilon \in O(1/\sqrt{n})$ that provides a sufficient approximation to make the desired distinction. The proven lower bound then states that for any such ϵ, a $(1-\epsilon)$-approximation algorithm must run in at least $2^{O(1/\epsilon)}n^{O(1)}$ time. Note that substituting $\epsilon = O(1/\sqrt{n})$ yields a running time of $2^{O(\sqrt{n})}n^{O(1)} = 2^{O(\sqrt{n})}$, which is the running time of the exact algorithm[2]. The lower bound does not hold for $\epsilon \leq o(1/\sqrt{n})$, since for each of these ϵ, the exact algorithm yields a $(1-\epsilon)$-approximation in $2^{o(1/\epsilon)}n^{O(1)}$ time. For these ϵ, the running time of a $(1-\epsilon)$-approximation algorithm is bounded by the running time of the exact algorithm instead, assuming Gap-ETH. We formalize this in the following theorem, by which the exact algorithm for Maximum Independent Set on planar graphs is optimal for obtaining $(1-\epsilon)$-approximations, for any $\epsilon \leq O(1/\sqrt{n})$.

Theorem 3.9. *For any $\epsilon \leq O(1/\sqrt{n})$, there does not exist a $(1-\epsilon)$-approximation algorithm for* Maximum Independent Set *on n-vertex planar graphs running in $2^{o(\sqrt{n})}$ time, unless Gap-ETH fails.*

4 Polynomial-space PTAS on Planar Graphs

In this section, we present polynomial-time approximation schemes that use polynomial space for the planar versions of Maximum Independent Set, Maximum Weighted Independent Set, Minimum Dominating Set, and Minimum Vertex Cover. Throughout this section, polynomial space refers to space usage that is polynomial in both ϵ and n, where ϵ is the parameter used in a PTAS, and n is the number of vertices in a graph. Furthermore, it should be noted that any algorithm running in polynomial time must always use polynomial space. We will only explicitly mention that a step used in a PTAS uses polynomial space where this may not be directly obvious.

4.1 Maximum Independent Set

Theorem 4.1. *There is a PTAS for* Maximum Independent Set *on planar graphs running in $2^{O(1/\epsilon)}\epsilon^2 n + O(n \log n)$ time and using polynomial space.*

Proof. Given an n-vertex planar graph G and $\epsilon > 0$. Let $r = 1/\epsilon^2$. We can obtain an r-division of G, with $\Theta(n/r) = \Theta(\epsilon^2 n)$ regions of $O(r) = O(1/\epsilon^2)$ vertices each and $O(\sqrt{r}) = O(1/\epsilon)$ boundary vertices each. The total number of boundary vertices is then $O(\epsilon n)$.

[2]This implies that the exact algorithm is optimal under Gap-ETH, which is not surprising since this algorithm has also been proven to be optimal under ETH.

Let OPT be the size of the maximum independent set in G. We have that $\frac{1}{4}n \leq OPT$, since all planar graphs have a 4-coloring. Removing the $O(\epsilon n)$ boundary vertices from the graph then decreases the optimum by at most ϵOPT, because $\epsilon\frac{1}{4}n \leq \epsilon OPT$ and $\epsilon\frac{1}{4}n = O(\epsilon n)$.

We remove all boundary vertices from the graph. For each of the $\Theta(\epsilon^2 n)$ regions, we compute a maximum independent set on its remaining $O(1/\epsilon^2)$ vertices. This can be done in $2^{O(\sqrt{1/\epsilon^2})} = 2^{O(1/\epsilon)}$ time, and using polynomial space [11]. The approximated maximum independent set I of G is then the union over the independent sets of all regions.

Since removing boundary decreases the optimum by at most ϵOPT, we have that $|I| \geq (1-\epsilon)OPT$. The r-division can be computed in $O(n \log n)$ time. Removing the boundary vertices takes $O(n)$ time. Computing a maximum independent set takes $2^{O(1/\epsilon)}$ time per region. The total running time is then $2^{O(1/\epsilon)}\epsilon^2 n + O(n \log n)$. All steps use polynomial space. □

4.2 Maximum Weighted Independent Set

The approach used in the proof of Theorem 4.1 does not work in the weighted case. This is because r-divisions rely on the planar separator theorem, which does not work on weighted graphs. The PTAS by Baker [12], described in Theorem 2.9, still works in the weighted case. Unfortunately, this PTAS uses exponential space, meaning we need to take a refined approach to obtain a polynomial-space PTAS for Maximum Weighted Independent Set. We first show that we can find a balanced vertex separator of size $O(t)$ in a weighted planar graph of treewidth t in polynomial time. We then use this balanced separator to prove the existence of a PTAS for Maximum Weighted Independent Set on planar graphs running in $2^{O(1/\epsilon^2)} + O(n^3)$ time and using polynomial space.

The following lemma and its proof are considered to be standard knowledge in the field. In Section 7.6.1 of the book *Parameterized Algorithms*, a similar lemma along with its proof can be found [25]. We include our own proof for this lemma regardless, as it provides insight as to why the PTAS works, and allows us to more accurately analyze the running time.

Lemma 4.2. *Given an n-vertex weighted planar graph $G = (V, E)$ and a tree decomposition T of G with treewidth t and size $O(nt)$, a separator $C \subset V$ of size $O(t)$ can be found in $O(nt)$ time and using polynomial space, such that in $G[V \setminus C]$ each connected component has weight at most $2/3$.*

Proof. Given an n-vertex weighted planar graph $G = (V, E)$ and a tree decomposition T of G with treewidth t and size $O(nt)$. If the sum of the vertex weights in any bag is at least $1/3$, then we return this bag as separator. Otherwise, the separator can be found by dynamic programming.

We root T at an arbitrary vertex r. To avoid confusion with the weight of vertices, let the *mass* of a bag in T be the mass of its children, plus the weight of its own vertices, minus the weight of the vertices it has in common with its parent. For each leaf bag, compute its mass by summing the weight of its vertices, and subtracting the weight of the vertices it has in common with its parent bag. For each parent bag compute its mass by summing the mass of its children and the weight of its own vertices, and again subtracting the weight of the vertices it has in common with its parent bag. We repeat this process until the mass of some bag is at least $1/3$. Then the parent of this bag can be selected as separator. If no such bag is found before r is reached, then r itself must be a balanced separator, since all of its child bags have mass less than $1/3$. It cannot be the case that the mass of a bag is less than $1/3$ in one step and the mass of its parent is more than $2/3$ in the next step, since there is no bag of which the weight of its vertices sum to at least $1/3$.

The found separator is of size $O(t)$, as T has treewidth $O(t)$. Computing the mass of a bag takes $O(t)$ time. There are $O(n)$ bags, so the running time of the dynamic programming is $O(nt)$. The total running time is $O(nt)$. The space usage is $O(nt)$, since T is of size $O(nt)$ and a single value is stored per bag. □

Theorem 4.3. *There is a PTAS for* Maximum Weighted Independent Set *on planar graphs running in $2^{O(1/\epsilon^2)} + O(n^3)$ time and using polynomial space.*

Proof. Given an n-vertex weighted planar graph G and $\epsilon > 0$. Let $c > 0$ be a constant, such that Baker's PTAS runs in $2^{c/\epsilon}n$ time. Then if $n > 2^{c/\epsilon}$, Baker's PTAS [12] can be used to approximate a maximum weighted independent set in $O(2^{c/\epsilon}n) = O(n^2)$ time, since the $2^{O(1/\epsilon)}$ space usage is polynomial in n. We therefore assume $n \leq 2^{c/\epsilon}$, which implies $\log n \leq c/\epsilon$.

Let $k = 1/\epsilon$. We use Baker's technique to split G into k-outerplanar connected components in linear time [12], for each $0 \leq i \leq k-1$, as described in Theorem 2.9. From this we obtain k graphs G_i, each consisting of k-outerplanar connected components. In each component, we compute a maximum weighted independent set as follows.

Let G' be a connected component of size n'. As G' is k-outerplanar it has treewidth $O(k)$ [27]. We compute a tree decomposition T' of treewidth $O(k)$ and size $O(n'k)$. By Lemma 4.2, we can find a balanced separator C of size $O(k)$ in $O(n'k)$ time. Removing C from G' leaves components $G'_1, ..., G'_p$. These components are all at most k-outerplanar, and have weight at most 2/3. For each of the $2^{O(k)}$ independent sets in C, take the union with the maximum weighted independent sets in $G'_1, ..., G'_p$, which are computed by recursion. Note that this uses polynomial space, since only the largest found independent set needs to be stored when iterating over the possible independent sets in C. Recomputing the tree decomposition in each recursive step is not required, as removing C from the tree decomposition leaves valid tree decompositions for each component $G'_1, ..., G'_p$. Additionally, for each component $G'_1, ..., G'_p$, Lemma 4.2 only needs to be applied once, as the separator does not change based on the current selected independent set in C. As a base case, we can compute a maximum weighted independent set on a component of which the tree decomposition consists of two bags in $2^{O(\sqrt{k})}$ time and using polynomial space [11]. We return the maximum over all independent sets of C as maximum weighted independent set of G', and return the union over the maximum weighted independent sets of all components as maximum weighted independent set of G_i. We take the maximum weighted independent set over all G_i to obtain the approximated maximum weighted independent set I of G.

Let OPT be the weight of a maximum weighted independent set in G. Removing levels $i \mod k$ decreases the optimum by at most $1/k \cdot OPT = \epsilon OPT$, for some $0 \leq i \leq k-1$, as was shown in Theorem 2.9. Then for some $0 \leq i \leq k-1$ it must hold that $w(I) \geq (1-\epsilon)OPT$, where $w(I)$ is the combined weight of all vertices in I.

Computing the levels of vertices in G and computing the graphs G_i takes $O(nk)$ time. The graphs G_i each consist of $O(n/k)$ k-outerplanar components to compute a maximum weighted independent set on. The components have an average of $n' = O(n/k)$ vertices.

For each component, Lemma 4.2 is applied a constant number of times in each layer of recursion. There are $O(\log n')$ layers of recursion. Note that each bag of the initial tree decomposition appears at most once per layer in the recursion, so applying Lemma 4.2 adds $O(n'k \log n') \leq O(n'^2)$ time per component. We also compute a tree decomposition in each component of treewidth $O(k)$ in $O(n'k^2 \log k) \leq O(n'^2 \log k)$ time and using polynomial space [24]. In each recursive step, we iterate over $2^{O(k)}$ possible independent sets of the separator, which takes $(2^{O(k)})^{\log n'}$ time and polynomial space per component. The total running time per component is then $(2^{O(k)})^{\log n'} + O(n'^2 \log k)$.

The sum over all n' in any G_i is at most n, since each vertex appears at most once across all components. Then the running time per G_i is $(2^{O(k)})^{\log n} + O(n^2 \log k) \leq 2^{O(1/\epsilon^2)} + O(n^2 \log k)$. The total running time over all k graphs G_i is then $2^{O(1/\epsilon^2)}k + O(n^2 k \log k) \leq 2^{O(1/\epsilon^2)} + O(n^3)$. In the case Baker's PTAS is used, the running time is $O(n^2)$ instead, which is dominated by the $O(n^3)$ term. All steps use polynomial space. □

4.3 Minimum Dominating Set

In [12] Baker discusses a number of problems that admit a PTAS using a similar technique as the approximation scheme for Maximum Independent Set. One of these problems is Minimum Dominating Set on planar graphs. Marzban and Gu [34] show that the approximation ratio achieved by Baker's application of the technique on Minimum Dominating Set is not bounded

by a constant factor. They modify the application of Baker's technique to obtain a PTAS for Minimum Dominating Set on planar graphs running in $2^{O(1/\epsilon)}n$ time.

4.3.1 Modifying an Existing PTAS

This PTAS works by first decomposing a planar graph G into $k+2$-outerplanar components, where the two levels on the boundary of each component overlap with its adjacent components. There are k ways of performing this decomposition, each causing different levels of vertices to be in the overlapping part. The *interior* vertices of a component are defined as being those vertices that are not on the boundary of the component. Then for each component, an exact algorithm is used to compute a set of minimum cardinality that dominates all its interior vertices. Note that any exact algorithm for Minimum Dominating Set could find such set by adding a vertex that is adjacent to all non-interior vertices, and another vertex only adjacent to that vertex, prior to running the algorithm. The exact algorithm used by Marzban and Gu runs in $O(2^b n)$ time and exponential space, where b is the branchwidth of the graph. The union over the dominating sets of the components is taken as approximated minimum dominating set of G. Marzban and Gu then show that for one of the k decompositions, the ratio between the minimum dominating set and found approximation is at most $1 + 2/k$.

We present a modification of the PTAS by Marzban and Gu that has increased time complexity in its reliance on ϵ, but only uses polynomial space. The proof of the correctness of this PTAS is largely similar to that of Marzban and Gu [34].

Theorem 4.4. *There is a PTAS for* Minimum Dominating Set *on planar graphs running in* $2^{O(1/\epsilon^2)}n^2$ *time and using polynomial space.*

Proof. Given an m-outerplanar graph $G = (V, E)$ on $n = |V|$ vertices. Let $c > 0$ be a constant, such that the PTAS by Marzban and Gu [34] runs in $2^{c/\epsilon}n$ time. Then if $n > 2^{c/\epsilon}$, the PTAS by Marzban and Gu PTAS [34] can be used to approximate a minimum dominating set in $O(2^{1/\epsilon}n) = O(n^2)$ time, since the $2^{O(1/\epsilon)}$ space usage is polynomial in n. We therefore assume $n \leq 2^{c/\epsilon}$, which implies $\log n \leq c/\epsilon$.

Compute vertex sets $V_1, ..., V_m$ representing the levels of G. Let $k = 1/\epsilon$. For each $2 \leq i < k + 2$, decompose G into $r = O(m/k)$ components $G_{i,1}, ..., G_{i,r}$ that are at most $(k+2)$-outerplanar, where levels $i - 1 \mod k + 2$ and $i \mod k + 2$ in G are the overlapping parts of adjacent components.

Let the *boundary* vertices of a component be those vertices in the component of which the level in the original graph was at least as high or at least as low as all other vertices in the component. All other vertices in that component are the *interior* vertices of that component. Note that a $(k + 2)$-outerplanar graph has treewidth $O(k)$, which implies a treedepth of at most $O(k \log n)$ (see Section 2.5). For each $0 \leq j \leq r$, compute a set $S_{i,j}$ of minimum cardinality that dominates every interior vertex in component $G_{i,j}$. This can be done in $O(3^d n^2)$ time and using polynomial space, using an algorithm by Belbasi and Fürer [35], where d is the treedepth of a graph. Let $S_i = \bigcup_{j=0}^{r} S_{i,j}$. We take the set S_i with minimum cardinality over all $2 \leq i < k + 2$ as the approximated minimum dominating set S.

We now show that the desired approximation ratio holds. Let D be a minimum dominating set of G. We need to show that $|S| \leq (1 + \epsilon)|D|$. For each $2 \leq i < k + 2$ and $0 \leq j \leq r$, let $D_{i,j} = D \cap S_{i,j}$. Here $D_{i,j}$ denotes a set containing those vertices from D that are in component j when G is decomposed into components according to i. Note that the interior vertices of any component cannot be dominated by vertices outside of that component. Then since D dominates every vertex in G, it must hold that $D_{i,j}$ dominates every interior vertex of $G_{i,j}$. Since $S_{i,j}$ is a set of minimum cardinality that also dominates every interior vertex in $G_{i,j}$, we have that

$$|S_{i,j}| \leq |D_{i,j}|.$$

We also know that $|S_i|$ is at most sum over $|S_{i,j}|$, for $0 \leq j \leq r$. It can be less due to some selected

vertices overlapping.

$$|S_i| \leq \sum_{j=0}^{r} |S_{i,j}| \leq \sum_{j=0}^{r} |D_{i,j}|.$$

For each $2 \leq i < k+2$, since the vertices on levels $i-1 \mod k+2$ and $i \mod k+2$ appear in two components we have that

$$\sum_{j=0}^{r} |D_{i,j}| \leq |D| + \sum_{j=0}^{r-1} |D \cap V_{j \cdot k+i-1}| + |D \cap V_{j \cdot k+i}|.$$

Note that there must be some $2 \leq i < k+2$ such that the overlapping parts of the components has an average number of vertices in common with D, meaning

$$\min_{i=2}^{k+1}\{\sum_{j=0}^{r-1} |D \cap V_{j \cdot k+i-1}| + |D \cap V_{j \cdot k+i}|\} \leq \frac{2|D|}{k}.$$

Since the approximated minimum dominating set S has minimum cardinality over all S_i, we have that

$$|S| \leq \sum_{j=0}^{r} |D_{i,j}| \leq |D| + \frac{2|D|}{k}.$$

Then the approximated minimum dominating set S has size at most $|S| \leq (1+2/k)|D| = (1+2\epsilon)|D|$. Note that this approximation ratio becomes $|S| \leq (1+\epsilon)|D|$ by, given some ϵ, simply using $\epsilon/2$.

Decomposing G into $k+2$-outerplanar connected components, and taking the union over their dominating sets can be done in $O(kn)$ time. A minimum dominating set in a component can be computed in $O(3^d n^2) = 2^{O(k \log n)} n^2 \leq 2^{O(1/\epsilon^2)} n^2$ time. For each of the $O(k)$ decompositions of G, each vertex appears in at most two components. The total running time is then $2^{O(1/\epsilon^2)} n^2$. All steps use polynomial space. □

4.3.2 Improved PTAS Using a Kernel

Using a different method, the running time of the above algorithm can be improved while maintaining polynomial space usage. We use an algorithm that is bounded in the treedepth of a graph, as well as a linear problem kernel for Minimum Dominating Set by Alber et al. [36]. We first prove that planar graphs have bounded treedepth, then introduce the kernel and show that it preserves approximation, and finally give a proof for the PTAS itself.

The lemma below is known, but we include our own proof for the sake of completeness, and to provide insight as to why the PTAS works.

Lemma 4.5. *A planar graph on n vertices has a treedepth decomposition of treedepth $O(\sqrt{n})$ that can be found in $O(n \log n)$ time.*

Proof. Let $G = (V, E)$ be a planar graph on $n = |V|$ vertices. By the planar separator theorem every planar graph has a separator $C \subset V$ of size $2\sqrt{2}\sqrt{n}$ that can be found in $O(n)$ time, such that removing C from the graph leaves connected components of size at most $2n/3$. Note that since C is a separator, the vertices in different components cannot have an edge between them, meaning they are not required to be ancestors in the treedepth decomposition. We form a path graph using the vertices from C, with arbitrary vertices r and l as root and leaf of the path graph respectively. We repeat this process by recursion on the remaining connected components in G until each component is of size $O(\sqrt{n})$. The root of each path graph is connected to the leaf of the path graph one recursive layer higher. The treedepth of the treedepth decomposition is then

$$\sum_{i=1}^{\log n} 2\sqrt{2}\sqrt{(2/3)^i n} \;=\; 2\sqrt{2}\sqrt{n} \sum_{i=1}^{\log n} \sqrt{(2/3)^i} \;=\; O(\sqrt{n}).$$

Since in each layer of recursion each vertex appears at most once, the total time it takes to compute the treedepth decomposition is $O(n \log n)$. □

We now move to explaining the linear problem kernel for Minimum Dominating Set on planar graphs by Alber et al. [36]. The kernel is achieved by repeated application of two reduction rules, as formalized the following theorem.

Theorem 4.6. *For a planar graph $G = (V, E)$ which is reduced with respect to Rules 1 and 2, we get $|V| \leq 335\gamma(G)$, i.e., the* Dominating Set *problem on planar graphs admits a linear problem kernel.*

The proof of this theorem can be found in Section 3 of [36]. We first give the definition of the rules, and then show that the rules are valid L-reductions.

The reduction rules use some notation related to the neighbourhood of vertices. Given a planar graph $G = (V, E)$, let $N(v)$ be the neighbourhood of some vertex $v \in V$, and let $N[v] = N(v) \cup \{v\}$. We can partition $N(v)$ into three sets by defining

$$\begin{aligned} N_1(v) &= \{u \in N(v) : N(u) \setminus N[v] \neq \emptyset\}, \\ N_2(v) &= \{u \in N(v) \setminus N_1(v) : N(u) \cap N_1(v) \neq \emptyset\}, \\ N_3(v) &= N(v) \setminus (N_1(v) \cup N_2(v)). \end{aligned}$$

In other words, $N_1(v)$ contains those vertices in $N(v)$ that are adjacent to one or more vertices outside of $N[v]$, $N_2(v)$ contains those vertices in $N(v)$ that are not in $N_1(v)$ and are adjacent to a vertex in $N_1(v)$, and $N_3(v)$ contains those vertices in $N(v)$ that are not in $N_1(v)$ or $N_2(v)$. The first reduction rule is then defined as follows:

Rule 4.3.1. *If $N_3(v) \neq \emptyset$, for some vertex v, then*

- *remove $N_2(v)$ and $N_3(v)$ from G and*
- *add a new vertex v' with the edge $\{v, v'\}$ to G.*

Intuitively, the size of the minimum dominating set does not change by applying Rule 1, as it enforces choosing a vertex that must be the best option for dominating the vertices in its neighbourhood. This is formalized in Lemma 4.7, of which the proof can be found in [36].

Lemma 4.7. *Let $G = (V, E)$ be a graph and let $G' = (V', E')$ be the resulting graph after having applied Rule 2 to G. Then $\gamma(G) = \gamma(G')$.*

The second reduction rule considers vertices in pairs and uses similar definitions as the first rule. Let $N(v, w) = N(v) \cup N(w)$. We can partition $N(v, w)$ into three sets by defining

$$\begin{aligned} N_1(v, w) &= \{u \in N(v, w) : N(u) \setminus N[v, w] \neq \emptyset\}, \\ N_2(v, w) &= \{u \in N(v, w) \setminus N_1(v, w) : N(u) \cap N_1(v, w) \neq \emptyset\}, \\ N_3(v, w) &= N(v, w) \setminus (N_1(v, w) \cup N_2(v, w)), \end{aligned}$$

with similar meanings to the definitions of $N_1(v)$, $N_2(v)$, and $N_3(v)$. The second reduction rule is then defined as follows.

Rule 4.3.2. *Consider $v, w \in V (v \neq w)$ and suppose that $N_3(v, w) \neq \emptyset$. Suppose that $N_3(v, w)$ cannot be dominated by a single vertex from $N_2(v, w) \cup N_3(v, w)$.*

Case 1. *If $N_3(v, w)$ can be dominated by a single vertex from $\{v, w\}$:*

1.1. *If $N_3(v, w) \subseteq N(v)$ as well as $N_3(v, w) \subseteq N(w)$:*

- *remove $N_3(v, w)$ and $N_2(v, w) \cap N(v) \cap N(w)$ from G and*
- *add two new vertices z, z' and edges $\{v, z\}$, $\{w, z\}$, $\{v, z'\}$, $\{w, z'\}$ to G.*

1.2. *If $N_3(v, w) \subseteq N(v)$, but not $N_3(v, w) \subseteq N(w)$:*

- *remove $N_3(v, w)$ and $N_2(v, w) \cap N(v)$ from G and*

- *add a new vertex v' and the edge $\{v, v'\}$ to G.*

1.3. *If $N_3(v, w) \subseteq N(w)$, but not $N_3(v, w) \subseteq N(v)$:*

- *remove $N_3(v, w)$ and $N_2(v, w) \cap N(w)$ from G and*
- *add a new vertex w' and the edge $\{w, w'\}$ to G.*

Case 2. *If $N_3(v, w)$ cannot be dominated by a single vertex from $\{v, w\}$:*

- *remove $N_3(v, w)$ and $N_2(v, w)$ from G and*
- *add two new vertices v', w' and edges $\{v, v'\}$, $\{w, w'\}$ to G.*

Similar to the application of Rule 1, it is intuitive that the size of the minimum dominating set does not change by applying Rule 2. If we suppose that $N_3(v, w)$ cannot be dominated by a single vertex from $N_2(v, w) \cup N_3(v, w)$, then a combination of v and w must be the best option for dominating $N_3(v, w)$. This is formalized in Lemma 4.8, of which the proof can be found in [36].

Lemma 4.8. *Let $G = (V, E)$ be a graph and let $G' = (V', E')$ be the resulting graph after having applied Rule 2 to G. Then $\gamma(G) = \gamma(G')$.*

We now prove that the kernel of Theorem 4.6 preserves approximation.

Lemma 4.9. *The reduction rules used in the kernel of Theorem 4.6 are L-reductions.*

Proof. Given a planar graph $G = (V, E)$ on $n = |V|$ vertices, let $\gamma(G)$ be the size of the minimum dominating set of G. We apply the kernel of Theorem 4.6 to obtain a planar graph G' with $n' \leq 335\gamma(G')$ vertices, where $\gamma(G')$ is the minimum dominating set of G'. By Lemma 4.7 and Lemma 4.8 we have that $\gamma(G') = \gamma(G)$. Let D' be a dominating set obtained by running a $(1 + \epsilon)$-approximation algorithm on G'. We have that $|D'| \leq (1 + \epsilon)\gamma(G') = (1 + \epsilon)\gamma(G)$, so the approximation holds.

To obtain the approximated minimum dominating set D of G, let $D = D' \cap V(G)$. Then for any gadget vertex x'_1, add its corresponding vertex $x_1 \in V(G)$ to D instead. □

We now use the kernel to obtain a PTAS for Minimum Dominating Set on planar graphs that uses only polynomial space.

Theorem 4.10. *There is a PTAS for* Minimum Dominating Set *on planar graphs running in $2^{O(1/\epsilon)}\epsilon^2 n + O(n^3)$ time and using polynomial space.*

Proof. Given a planar graph $G = (V, E)$ on $n = |V|$ vertices, and $\epsilon > 0$. Let $\gamma(G)$ be the size of the minimum dominating set of G. We can use the kernel of Theorem 4.6 to obtain a planar graph G' with $n' = O(\gamma(G))$ vertices. By Lemma 4.9 this kernel preserves approximation.

Let $r = 1/\epsilon^2$. We can obtain an r-division of G', with $\Theta(\epsilon^2 n')$ regions of $O(1/\epsilon^2)$ vertices each and $O(1/\epsilon)$ boundary vertices each. The total number of boundary vertices is then $O(\epsilon n')$.

The regions of $O(1/\epsilon^2)$ vertices are planar graphs and have treedepth $O(\sqrt{1/\epsilon^2}) = O(1/\epsilon)$, by Lemma 4.5. We can compute a minimum dominating set in each region in $O(3^{O(1/\epsilon)}(1/\epsilon^2)^2) = 2^{O(1/\epsilon)}$ time and using polynomial space, using an algorithm by Belbasi and Fürer [35].

We take the union over the minimum dominating sets of the regions and all boundary vertices as approximated minimum dominating set D. Including all boundary vertices in the dominating set increases the dominating set of G by at most $O(\epsilon n') = O(\epsilon\gamma(G))$, since $n' = O(\gamma(G))$. We then have that $|D| \leq (1 + \epsilon)\gamma(G)$.

The kernel takes $O(n^3)$ time to run on planar graphs [36]. Computing the r-division takes $O(n \log n)$ time. Computing the minimum dominating set on the regions takes $2^{O(1/\epsilon)} \cdot \Theta(\epsilon^2 n) = 2^{O(1/\epsilon)}\epsilon^2 n$ time. Finally, computing D takes $O(n)$ time. The total running time is then $2^{O(1/\epsilon)}\epsilon^2 n + O(n^3)$. All steps use polynomial space. □

4.4 Minimum Vertex Cover

We present a PTAS for Minimum Vertex Cover on planar graphs using polynomial space. The PTAS uses a linear problem kernel based on a theorem by Nemhauser and Trotter [37]. We use a representation of the kernel as shown in Chapter 6.1 of the book *Kernelization: Theory of Parameterized Preprocessing* [18], which refers to Khuller [38] and Chen et al. [39] for the representation of proof of the theorem and its application to obtain a kernel respectively. We first introduce the theorem by Nemhauser and Trotter [37], then show how this theorem can be used to obtain a linear problem kernel for Minimum Vertex Cover, and finally give a proof for the PTAS.

An instance of Minimum Vertex Cover can be formulated as an ILP. Given a graph $G = (V, E)$, for each $v \in V$, we create a decision variable x_v that has a value of 1 if v is in the vertex cover, and 0 otherwise. The ILP formulation is then as follows:

$$\begin{aligned} \text{minimize} \quad & \Sigma_{v \in V} x_v & \\ \text{subject to} \quad & x_u + x_v \geq 1 & \forall (u, v) \in E \\ & x_v \in \{0, 1\} & \forall v \in V \end{aligned}$$

The first constraint ensures that for each edge, at least one of its endpoints is in the vertex cover. We now apply an LP relaxation to the ILP, by allowing the decision variables to take any value $0 \leq x_v \leq 1$. Note that the $x_v \leq 1$ part of the constraint can be omitted, since minimizing over the x_v will always result in their values being at most 1 for an optimal solution. The LP formulation is then as follows:

$$\begin{aligned} \text{minimize} \quad & \Sigma_{v \in V} x_v & \\ \text{subject to} \quad & x_u + x_v \geq 1 & \forall (u, v) \in E \\ & 0 \leq x_v & \forall v \in V \end{aligned}$$

Let $L(G)$ be LP formulation as described above with respect to a graph G. An optimal solution to $L(G)$ consists of a value between 0 and 1 for each x_v. We partition the vertices $V(G)$ into three sets, based on the value of their decision variable:

- V_0 contains those $v \in V$ for which $x_v < \frac{1}{2}$,
- $V_{\frac{1}{2}}$ contains those $v \in V$ for which $x_v = \frac{1}{2}$,
- V_1 contains those $v \in V$ for which $x_v > \frac{1}{2}$.

The theorem by Nemhauser and Trotter states that, given these sets of vertices for an optimal solution of $L(G)$, there exists a minimum vertex cover that consists of all vertices in V_1 and zero or more vertices in $V_{\frac{1}{2}}$. We give proofs for this theorem, and subsequently its application to yield a linear problem kernel, for completeness and increased insight as to why the kernelization works, closely following the proofs as described in [18].

Theorem 4.11 (Nemhauser-Trotter's Theorem)**.** *There is a minimum vertex cover OPT of G such that $V_1 \subseteq OPT \subseteq V_1 \cup V_{\frac{1}{2}}$.*

Proof. Given a graph G on n vertices, let OPT be a minimum vertex cover of G such that $V_0 \cap OPT \neq \emptyset$. Note that any vertex in V_0 can only have neighbours in V_1, since for any edge (u, v) it must hold that $x_u + x_v \geq 1$. This means that by removing V_0 from OPT and including all vertices of V_1 in OPT, we maintain a vertex cover. Let $OPT' = (OPT \setminus V_0) \cup V_1$ be this vertex cover. Note that OPT' satisfies $V_1 \subseteq OPT' \subseteq V_1 \cup V_{\frac{1}{2}}$. If it holds that $|V_0 \cap OPT| \geq |V_1 \setminus OPT|$, then OPT' is a minimum vertex cover of G.

We claim that $|V_0 \cap OPT| < |V_1 \setminus OPT|$ cannot occur. We will prove that this is the case by assuming $|V_0 \cap OPT| < |V_1 \setminus OPT|$ holds and showing this leads to a contradiction.

Let $\epsilon = \min\{|x_v - \frac{1}{2}| \mid v \in V_0 \cup V_1\}$ be the smallest difference of the value of a decision variable in $V_0 \cup V_1$ with $\frac{1}{2}$. For each vertex $v \in V$, we create a new decision variable y_v according to the following rules:

- if $v \in V_1 \setminus OPT$, then $y_v = x_v - \epsilon$,
- if $v \in V_0 \cap OPT$, then $y_v = x_v + \epsilon$,
- otherwise $y_v = x_v$.

We assumed that $|V_0 \cap OPT| < |V_1 \setminus OPT|$, so there are more decision variables from which ϵ was subtracted than decision variables to which ϵ was added, which implies it holds that

$$\Sigma_{v \in V} y_v < \Sigma_{v \in V} x_v.$$

We now show that the decision variables y_v form a valid solution to $L(G)$, by verifying that for each edge $(u, v) \in E(G)$ it holds that $y_u + y_v \geq 1$.

Consider any edge $(u, v) \in E$ such that at least one endpoint is in $V_1 \setminus OPT$. Other edges will always still satisfy the constraint, since the decision variables of their endpoints can only have increased in value or remained the same. Let u be the endpoint in $V_1 \setminus OPT$. Note that v must be in OPT, since OPT is a vertex cover. We consider the three different cases for v:

- if $v \in V_1$, then $y_u + y_v = x_u - \epsilon + x_v \geq \frac{1}{2} + \frac{1}{2} = 1$,
- if $v \in V_0$, then $y_u + y_v = x_u - \epsilon + x_v + \epsilon = x_u + x_v \geq 1$,
- if $v \in V_{\frac{1}{2}}$, then $y_u + y_v = x_u - \epsilon + x_v \geq \frac{1}{2} + \frac{1}{2} = 1$.

In each case, the edges still satisfy the constraint. This means the decision variables y_v form a valid solution to $L(G)$ with lower cost than x_v did, which contradicts the assumption that OPT is a minimal vertex cover. Then it must hold that $|V_0 \cap OPT| \geq |V_1 \setminus OPT|$, which proves that OPT' is a minimum vertex cover that satisfies $V_1 \subseteq OPT' \subseteq V_1 \cup V_{\frac{1}{2}}$. □

We now use Theorem 4.11 to show that Vertex Cover admits a linear problem kernel on general graphs.

Theorem 4.12. Vertex Cover *admits a kernel with at most* $2|S|$ *vertices, where* S *is a minimum vertex cover.*

Proof. Given a graph G, let $L(G)$ be the corresponding Minimum Vertex Cover linear program. We obtain the vertex sets V_0, $V_{\frac{1}{2}}$, and V_1 by solving $L(G)$. Let S be a minimum vertex cover in G. By Theorem 4.11, it can be assumed that $V_1 \subseteq S \subseteq V_1 \cup V_{\frac{1}{2}}$. Let $G' = G[V_{\frac{1}{2}}]$. For each vertex $v \in V_{\frac{1}{2}}$ it holds that $x_v = \frac{1}{2}$. Since the sum over all such x_v is at most $|S|$, we have that $|V(G')| \leq 2|S|$. □

Given a graph G with n vertices and m edges, the kernel of Theorem 4.12 can be computed in $O(m\sqrt{n})$ time [18]. We now use the kernel to obtain a PTAS for Minimum Dominating Set on planar graphs that uses only polynomial space.

Theorem 4.13. *There is a PTAS for* Minimum Vertex Cover *on planar graphs running in* $2^{O(1/\epsilon)}\epsilon^2 n + O(n^{1.5})$ *time and using polynomial space.*

Proof. Given a planar graph $G = (V, E)$ with $n = |V|$ vertices and m edges, and some $\epsilon > 0$. Let $\tau(G)$ be the size of a minimum vertex cover of G. Let $L(G)$ be the Minimum Vertex Cover linear program of G. Solving $L(G)$ yields vertex sets V_0, $V_{\frac{1}{2}}$, and V_1. By Theorem 4.12, we can obtain a kernel $G' = G[V_{\frac{1}{2}}]$ with $V(G') = n' \leq 2\tau(G)$ vertices in $O(n\sqrt{m})$ time.

Let $r = 1/\epsilon^2$. We can obtain an r-division of G', with $\Theta(\epsilon^2 n')$ regions of $O(1/\epsilon^2)$ vertices each and $O(1/\epsilon)$ boundary vertices each. The total number of boundary vertices is then $O(\epsilon n')$.

The regions of $O(1/\epsilon^2)$ vertices are planar graphs. We find a minimum vertex cover in each region by computing a maximum independent set in $2^{O(\sqrt{1/\epsilon^2})} = 2^{O(1/\epsilon)}$ time [11], and taking its compliment.

We take as the approximated minimum vertex cover S: vertex set V_1, the union over the vertex covers of the regions, and all boundary vertices. By Theorem 4.11, we can include V_1 in S and compute the minimum vertex cover on G', without increasing the size of the minimum vertex cover. Including all boundary vertices in the vertex cover increases the vertex cover of G by at most $O(\epsilon n') = O(\epsilon \tau(G))$, since $n' = O(\tau(G))$. We then have that $|S| \leq (1+\epsilon)\tau(G)$.

The kernel takes $O(m\sqrt{n}) \leq O(n^{1.5})$ time to compute [18]. Computing the r-division takes $O(n \log n)$ time. Computing a minimum vertex cover on the regions takes $2^{O(1/\epsilon)} \cdot \Theta(\epsilon^2 n) = 2^{O(1/\epsilon)}\epsilon^2 n$ time. Finally, computing S takes $O(n)$ time. The total running time is then $2^{O(1/\epsilon)}\epsilon^2 n + O(n^{1.5})$. All steps use polynomial space. □

5 Conclusion

A result by Marx [30] shows that there does not exist a PTAS for Maximum Independent Set on planar graphs running in $2^{O(1/\epsilon)^{1-\delta}} n^{O(1)}$ time, for any $\delta > 0$, assuming ETH holds. We refined this result by proving that there does not exist a PTAS for Maximum Independent Set on planar graphs running in $2^{o(1/\epsilon)} n^{O(1)}$ time, assuming Gap-ETH holds. The refined lower bound matches running time of the best known PTAS for Maximum Independent Set on planar graphs by Baker [12], running in $2^{O(1/\epsilon)} n$ time. This implies the lower bound is tight, and the running time of the PTAS by Baker is optimal in its reliance on ϵ. We also showed that the exact algorithm for Maximum Independent Set on planar graphs running in $2^{O(\sqrt{n})}$ time is optimal for obtaining $(1-\epsilon)$-approximations, for any $\epsilon \leq O(1/\sqrt{n})$. Many PTAS's, including the one by Baker, have space usage exponential in the size of the input. We presented PTAS's that only use polynomial space for a number of planar graph problems. We showed there is a PTAS for Maximum Independent Set on planar graphs running in $2^{O(1/\epsilon)}\epsilon^2 n + O(n \log n)$ time and using polynomial space, using an r-division and a lower bound on the maximum independent set in planar graphs. We also showed there is a PTAS for Maximum Weighted Independent Set on planar graphs running in $2^{O(1/\epsilon^2)} + O(n^3)$ time and using polynomial space, using Baker's technique and a vertex separator using tree decompositions. Additionally, we modified a PTAS for Minimum Dominating Set on planar graphs by Marzban and Gu [34] to have polynomial space usage, in exchange for an increase time complexity. This was done by replacing the exact algorithm used with one that has a running time bounded in the treedepth of the graph. Using a different method, we then improved upon the reliance of this PTAS on ϵ, by using a linear problem kernel for Minimum Dominating Set to obtain a PTAS running in $2^{O(1/\epsilon)}\epsilon^2 n + O(n^3)$ time and using polynomial space. Finally, we showed there is a PTAS for Minimum Vertex Cover on planar graphs running in $2^{O(1/\epsilon)}\epsilon^2 n + O(n^{1.5})$ time and using polynomial space.

We suggest a number of possible future research directions.

1. The existence of a linear-size PCP would imply that ETH is equivalent to Gap-ETH. A proof that such PCP construction exists would mean that our proof on the lower bound of a PTAS for Maximum Independent Set on planar graphs is valid assuming ETH, as opposed to the stronger assumption of Gap-ETH.

2. In Section 3.3 we observed that, when $(1-\epsilon)$-approximating the Maximum Independent Set on planar graphs, the exact algorithm is optimal for a certain range of values of ϵ. This observation could be further investigated to see whether modified reductions from MAX2SAT to Matrix Tiling with $D = 2$, or from Matrix Tiling with $D = 2$ to Planar Maximum Independent Set, could increase the range of values of ϵ for which this observation holds. Additionally, a similar analysis can be performed for other problems with tight lower bounds.

3. The polynomial-space PTAS for Maximum Independent Set on planar graphs has a running time that is linear in the exponent, while the same PTAS for Maximum Weighted Independent Set is quadratic in the exponent. An interesting question is whether the running time of the latter can be improved such that it matches the running time of the former in its reliance on ϵ.

4. We showed the existence of polynomial-space PTAS's for a number of planar graph problems. The running times of these PTAS's seem to frequently match the running times of their exponential-space counterparts in their reliance on ϵ. Further research could show the existence of polynomial-space PTAS's for other planar graph problems with running times similar to existing exponential-space PTAS's.

References

[1] Eugene L. Lawler. The traveling salesman problem: A guided tour of combinatorial optimization. *Journal of the Operational Research Society*, 37(5):535–536, May 1986. ISSN 1476-9360. doi: 10.1057/jors.1986.93. URL https://doi.org/10.1057/jors.1986.93.

[2] Nicos Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. *Operations Research Forum*, 3(1):20, Mar 2022. ISSN 2662-2556. doi: 10.1007/s43069-021-00101-z. URL https://doi.org/10.1007/s43069-021-00101-z.

[3] Anatoliy I. Serdyukov. О некоторых экстремальных обходах в графах (On some extremal walks in graphs). *Upravlyaemye Sistemy*, pages 76–79, 1978. URL http://nas1.math.nsc.ru/aim/journals/us/us17/us17_007.pdf.

[4] Sanjeev Arora. Polynomial time approximation schemes for euclidean traveling salesman and other geometric problems. *J. ACM*, 45(5):753–782, sep 1998. ISSN 0004-5411. doi: 10.1145/290179.290180. URL https://doi.org/10.1145/290179.290180.

[5] Tibor Gallai. Über extreme punkt- und kantenmengen. *Ann. Univ. Sci. Budapest Eötvös Sect. Math.*, 2:133–138, 1959.

[6] Nicholas Rhodes, Peter Willett, Alain Calvet, James B. Dunbar, and Christine Humblet. Clip: similarity searching of 3d databases using clique detection. *Journal of Chemical Information and Computer Sciences*, 43(2):443–448, Mar 2003. ISSN 0095-2338. doi: 10.1021/ci025605o. URL https://doi.org/10.1021/ci025605o.

[7] Ram Samudrala and John Moult. A graph-theoretic algorithm for comparative modeling of protein structure. *Journal of Molecular Biology*, 279(1):287–302, 1998. ISSN 0022-2836. doi: https://doi.org/10.1006/jmbi.1998.1689. URL https://www.sciencedirect.com/science/article/pii/S0022283698916898.

[8] Johan Håstad. Clique is hard to approximate within $n^{1-\epsilon}$. *Acta Mathematica*, 182(1):105–142, Mar 1999. ISSN 1871-2509. doi: 10.1007/BF02392825. URL https://doi.org/10.1007/BF02392825.

[9] David Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. In *Proceedings of the Thirty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '06, page 681–690, New York, NY, USA, 2006. Association for Computing Machinery. ISBN 1595931341. doi: 10.1145/1132516.1132612. URL https://doi.org/10.1145/1132516.1132612.

[10] Richard J. Lipton and Robert Endre Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36(2):177–189, 1979. doi: 10.1137/0136016. URL https://doi.org/10.1137/0136016.

[11] Richard J. Lipton and Robert Endre Tarjan. Applications of a planar separator theorem. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, SFCS '77, page 162–170, USA, 1977. IEEE Computer Society. doi: 10.1109/SFCS.1977.6. URL https://doi.org/10.1109/SFCS.1977.6.

[12] Brenda S. Baker. Approximation algorithms for np-complete problems on planar graphs. *J. ACM*, 41(1):153–180, jan 1994. ISSN 0004-5411. doi: 10.1145/174644.174650. URL https://doi.org/10.1145/174644.174650.

[13] Neil Robertson and Paul D. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *Journal of Algorithms*, 7(3):309–322, 1986. ISSN 0196-6774. doi: https://doi.org/10.1016/0196-6774(86)90023-4. URL https://www.sciencedirect.com/science/article/pii/0196677486900234.

[14] Umberto Bertelè and Francesco Brioschi. *Nonserial Dynamic Programming*. Academic Press, Inc., USA, 1972. ISBN 0120934507.

[15] Hans L. Bodlaender. Dynamic programming on graphs with bounded treewidth. In Timo Lepistö and Arto Salomaa, editors, *Automata, Languages and Programming*, pages 105–118, Berlin, Heidelberg, 1988. Springer Berlin Heidelberg. ISBN 978-3-540-39291-0.

[16] Greg N. Frederickson. Fast algorithms for shortest paths in planar graphs, with applications. *SIAM J. Comput.*, 16(6):1004–1022, dec 1987. ISSN 0097-5397. doi: 10.1137/0216064. URL https://doi.org/10.1137/0216064.

[17] Michael T. Goodrich, Siddharth Gupta, Hadi Khodabandeh, and Pedro Matias. How to catch marathon cheaters: New approximation algorithms for tracking paths. In Anna Lubiw, Mohammad Salavatipour, and Meng He, editors, *Algorithms and Data Structures*, pages 442–456, Cham, 2021. Springer International Publishing. ISBN 978-3-030-83508-8.

[18] Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization: Theory of Parameterized Preprocessing*, chapter 6, pages 13, 93–98. Cambridge University Press, 2019. doi: 10.1017/9781107415157.

[19] Marco Cesati and Luca Trevisan. On the efficiency of polynomial time approximation schemes. *Information Processing Letters*, 64(4):165–171, 1997. ISSN 0020-0190. doi: https://doi.org/10.1016/S0020-0190(97)00164-6. URL https://www.sciencedirect.com/science/article/pii/S0020019097001646.

[20] Norishige Chiba, Takao Nishizeki, Shigenobu Abe, and Takao Ozawa. A linear algorithm for embedding planar graphs using pq-trees. *Journal of Computer and System Sciences*, 30 (1):54–76, 1985. ISSN 0022-0000. doi: https://doi.org/10.1016/0022-0000(85)90004-2. URL https://www.sciencedirect.com/science/article/pii/0022000085900042.

[21] Pravin M. Vaidya. Speeding-up linear programming using fast matrix multiplication. In *30th Annual Symposium on Foundations of Computer Science*, pages 332–337, 1989. doi: 10.1109/SFCS.1989.63499.

[22] Ravindran Kannan and Clyde L. Monma. On the computational complexity of integer programming problems. In Rudolf Henn, Bernhard Korte, and Werner Oettli, editors, *Optimization and Operations Research*, pages 161–172, Berlin, Heidelberg, 1978. Springer Berlin Heidelberg. ISBN 978-3-642-95322-4.

[23] Wikipedia contributors. L-reduction — Wikipedia, the free encyclopedia, 2023. URL https://en.wikipedia.org/w/index.php?title=L-reduction&oldid=1136313322. [Online; accessed 26-June-2023].

[24] Frank Kammer and Torsten Tholey. Approximate tree decompositions of planar graphs in linear time. *Theoretical Computer Science*, 645:60–90, 2016. ISSN 0304-3975. doi: https://doi.org/10.1016/j.tcs.2016.06.040. URL https://www.sciencedirect.com/science/article/pii/S0304397516302961.

[25] Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Daniel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer Publishing Company, Incorporated, 1st edition, 2015. ISBN 3319212745.

[26] David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms*, pages 269–277. Cambridge University Press, 2010.

[27] Hans L. Bodlaender. Planar graphs with bounded treewidth. *Technical Report RUU-CS-88-14*, mar 1988.

[28] Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *J. Comput. Syst. Sci.*, 62(2):367–375, mar 2001. ISSN 0022-0000. doi: 10.1006/jcss.2000.1727. URL https://doi.org/10.1006/jcss.2000.1727.

[29] Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, dec 2001. ISSN 0022-0000. doi: 10.1006/jcss.2001.1774. URL https://doi.org/10.1006/jcss.2001.1774.

[30] Daniel Marx. On the optimality of planar and geometric approximation schemes. In *Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '07, page 338–348, USA, 2007. IEEE Computer Society. ISBN 0769530109. doi: 10.1109/FOCS.2007.50. URL https://doi.org/10.1109/FOCS.2007.50.

[31] Irit Dinur. Mildly exponential reduction from gap-3sat to polynomial-gap label-cover. *Electronic colloquium on computational complexity ECCC*, TR16-128:15, 2016. ISSN 1433-8092. URL https://eccc.weizmann.ac.il/report/2016/128.

[32] Pasin Manurangsi and Prasad Raghavendra. A birthday repetition theorem and complexity of approximating dense CSPs, 2016. URL https://arxiv.org/abs/1607.02986.

[33] Michael R. Garey, David S. Johnson, and Larry Stockmeyer. Some simplified np-complete graph problems. *Theoretical Computer Science*, 1(3):237–267, 1976. ISSN 0304-3975. doi: https://doi.org/10.1016/0304-3975(76)90059-1. URL https://www.sciencedirect.com/science/article/pii/0304397576900591.

[34] Marjan Marzban and Qian-Ping Gu. Computational study on a ptas for planar dominating set problem. *Algorithms*, 6(1):43–59, 2013. ISSN 1999-4893. doi: 10.3390/a6010043. URL https://www.mdpi.com/1999-4893/6/1/43.

[35] Mahdi Belbasi and Martin Fürer. Saving space by dynamic algebraization based on tree decomposition: Minimum dominating set. *CoRR*, abs/1711.10088, 2017. URL http://arxiv.org/abs/1711.10088.

[36] Jochen Alber, Michael R. Fellows, and Rolf Niedermeier. Polynomial-time data reduction for dominating set. *J. ACM*, 51(3):363–384, may 2004. ISSN 0004-5411. doi: 10.1145/990308.990309. URL https://doi.org/10.1145/990308.990309.

[37] George L. Nemhauser and Leslie E. Trotter. Vertex packings: Structural properties and algorithms. *Mathematical Programming*, 8(1):232–248, Dec 1975. ISSN 1436-4646. doi: 10.1007/BF01580444. URL https://doi.org/10.1007/BF01580444.

[38] Samir Khuller. Algorithms column: The vertex cover problem. *SIGACT News*, 33(2):31–33, jun 2002. ISSN 0163-5700. doi: 10.1145/564585.564598. URL https://doi.org/10.1145/564585.564598.

[39] Jianer Chen, Iyad A. Kanj, and Weijia Jia. Vertex cover: Further observations and further improvements. *Journal of Algorithms*, 41(2):280–301, 2001. ISSN 0196-6774. doi: https://doi.org/10.1006/jagm.2001.1186. URL https://www.sciencedirect.com/science/article/pii/S0196677401911861.