Fixed Parameter Tractable Algorithms for Optimal Covering Tours with Turns

Nuo Yu

Computer Science

McGill University

Montreal,Quebec

August 2008

A thesis submitted to McGill University in partial fulfilment of the requirements for the degree of Master of Science

Your file Votre référence
ISBN: 978-0-494-66898-6
Our file Notre référence
ISBN: 978-0-494-66898-6

ABSTRACT

Many geometry problems can be solved by transformation to graph problems. Often, both the geometry version and graph version of the problem are NP-hard - and therefore not likely to be solved in polynomial time. One approach to solving these hard problems is to use fixed parameter tractable (FPT) algorithms. We present a framework for developing FPT algorithms for graph problems using dynamic programming, monadic second order logic of graphs, tree-width, and bidimensionality. We use this framework to obtain FPT results for covering tour problems on grid-graphs with turn costs. The results for these problems are not practical, but they demonstrate how the basic framework can be used to quickly obtain FPT results. We provide suggestions on further research to improve our FPT results and to apply our framework to obtain new FPT results.

ABRÉGÉ

De nombreux problmes de géométrie peuvent tre résolus par des transformations en problmes de graphes. Souvent, la version géométrique comme la version graphique du problme sont NP-dures - et il est donc peu probable qu'elles puissent tre résolues en temps polynomial. Une approche pour résoudre ces probl mes difficiles est d'utiliser des algorithmes Tractables avec Param tre Fixe (TPF). Nous présentons un paradigme pour développer des algorithmes TPF pour des probl mes de graphes, en utilisant la programmation dynamique, la logique monadique du second ordre sur les graphes, la largeur d'arbre, et la bidimensionalité. Nous utilisons ce paradigme pour obtenir des résultats TPF pour des probl mes de tournées couvrantes dans des graphes en grilles avec cots sur les virages. Les résultats sur ces probl mes sont impraticables, mais ils démontrent comment le paradigme de base peut tre utilisé pour obtenir rapidement des résultats TPF. Nous proposons des voies de recherches pour améliorer nos résultats TPF, et pour appliquer notre paradigme pour obtenir de nouveaux résultats TPF.

ACKNOWLEDGEMENTS

I would like to thank Sue Whitesides for inviting me to the Bellairs Workshop, where the seeds of this thesis were sown; for providing valuable guidance and feedback throughout this project. I would like to thank Mike Fellows, Fran Rosamond, and Christophe Paul for their illuminating discussions at the Bellairs Workshop; moreover, I would like to thank Christophe Paul for our meetings afterwards, where we developed the ideas presented in this thesis. For financial support, I would like to thank Sue Whitesides and Quebec AFE.

Lastly, I would like to thank my parents for their love and support.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1
Introduction

The purpose of this thesis is to study fixed parameter tractable (FPT) algorithms and techniques for solving problems on grid-graphs[1] , which can be used to solve or approximate geometry problems. We focus on covering tour problems on grid-graphs, but our techniques can be used for other problems on grid-graphs as well.

1.1 Motivation

A universal approach to solving a problem is to transform it a problem that we know how to solve. In doing so, we use existing algorithms for the latter problem and "export" them to solve the former. Graph problems are often targets of these problem transformations because graphs are suited to model many types of problems and graph theory is rich with algorithms and algorithmic techniques for solving these problems. We are interested in algorithms and techniques for solving problems on embedded graphs, or more specifically, grid-graphs. Such graphs can be used to model (sometimes only approximately) many geometry problems. In particular, we focus on covering tour problems on grid-graphs.

As an example illustrating the approach of solving a geometry problem by transformation to a graph problem, consider the SHORTEST PATHS PROBLEM in the plane

[1] See Chapter 2 for definitions of fixed parameter tractability and grid-graphs.

with polygonal obstacles [23]. This problem can be solved by computing the *visibility graph* for the geometric input, so that a shortest path in the geometric problem corresponds to a shortest path in the visibility graph, and then solving the shortest paths problem on the visibility graph using Dijkstra's algorithm. Of course, Dijkstra's algorithm is efficient and produces optimal solutions. Furthermore, the visibility graph corresponding to the geometric input can be efficiently computed. Thus, the SHORTEST PATHS PROBLEM can be efficiently solved by reducing to a graph problem.[2]

This approach can be generalized to solve the 3D SHORTEST PATHS PROBLEM. However, this problem is NP-Hard [12]. One difficulty in the 3D setting is that a shortest path need not travel via vertices of the polytope. This issue can be addressed by discretizing the problem using sample points and then solving the shortest paths problem on the visibility graph based on the discrete sample points [14]. However, the resulting solution is an approximation because the optimal solution to the original geometry problem may not be a solution in the visibility graph computed from the discrete sample points.

In the examples above, the underlying graph problem is easy; i.e., shortest paths in graphs can be computed by an efficient algorithm. In turn, this algorithm can be exported to compute geometric shortest paths. One should reduce to easy problems whenever possible. When this is not possible, an alternative is to reduce to NP-hard graph problems that can be solved using known approximation algorithms or

[2] We assume a real RAM model of computation for this example.

FPT algorithms. This way, approximation algorithms or FPT algorithms may be obtained for the original problem. Approximation algorithms are fast but need not find optimal solutions, while FPT algorithms find optimal solutions but need not have strictly polynomial running times.

In the next section, we will introduce geometric covering tour problems as well as the graph problems that model them. Both the geometry and graph versions are NP-Hard. Efficient constant-factor approximation algorithms are known for the graph problems, which are the ones we are interested in. We choose to use the FPT approach to solve this problem, to obtain new results and to offer an alternative with the approximation approach.

1.2 Covering Tour Problems

Abstractly, a geometric covering tour problem is to find a "good" path (tour) for an object (cutter) to cut a given region (pocket). Different applications dictate different rules and restrictions on the input. For example, the cutter may be a unit circle, a unit square, or some other shape. The pocket may be a discrete point-set, a simple polygon, or a multiply connected region. The tour may be restricted to sequences of straight line segments, or it may be an arbitrary curve. The cutter may or may not be allowed to leave the pocket. The cutter may or may not be allowed to cut parts of the pocket multiple times. The cost function may be tour length, number of turns, a combination of both, or something entirely different.

An example of a covering tour problem is the TRAVELING SALESMAN PROBLEM (TSP). In this case, the "cutter" is simply a point; the "pocket" is a discrete point set; the cutter is allowed to leave the region, but it is not allowed to cut any part of

the pocket multiple times. Note that this geometry problem is also a graph problem, where the vertices correspond to points from the discrete point set, and pairs of vertices are joined by edges with weight equal to the distance[3] .

We are interested in a type of covering tour problem called *milling problems*, which naturally arise in the context of numerically controlled (NC) machining. NC machines are used to shape metals by removing pockets from the initial shape to produce the desired shape. Given a pocket and a cutter, the objective is to find a milling tour. Upon completing a milling tour, the cutter must have removed exactly all material from the given pocket - no more and no less. Thus, milling problems are covering tour problems where the cutter is not allowed to cut outside the pocket, but it is allowed to cut parts of the pocket multiple times. Usually, the cutter shape is a unit square or circle. Traditionally, theoretical research has been focused on minimizing the tour length. We address the problem of minimizing the number of turns because it is a better cost measure for some applications and it can be parameterized more naturally.

A graph problem that reasonably models the geometric milling problem is to find a tour that visits all vertices at least once, while edges can be used any number of times. We can equip the edges with directions when the problem is to minimize the number turns. Unfortunately, this general graph problem is W[2]-hard (when parameterized by number of turns) and therefore not likely to be solved using an

[3] Of course, if the edge weights are constrained to be rational numbers, then the graphs may be only approximations

FPT algorithm[4] . We can address this problem by imposing restrictions on the input graphs. Specifically, we focus on grid-graphs. Such graphs arise naturally from transformation of the geometry problem (See Chapter 2). Looking ahead, this restriction allows us to prove *parameter-tree-width* bounds, and use algorithmic techniques to solve problems on *bounded tree-width graphs.*

1.3 Statement of originality

Many of the ideas in this work originated during discussions with Christophe Paul, Mike Fellows, Frances Rosamond, and my supervisor Sue Whitesides at the 2007 INRIA McGill Bellairs Workshop on Geometry. I would like to thank these people for their contributions. My original work is in chapters 4 and 5, which contains proofs of NP-Hardness, parameter-tree-width bound, fixed parameter tractability using Courcelle's machinery, and fixed parameter tractability via dynamic programming on tree decompositions. I contributed substantially to these results and authored the presentation of them in this thesis.

1.4 Organization of the rest of the thesis

In Chapter 2, we discuss related works on covering tour problems. These works generally provide approximation algorithms by transformation to graph problems. In Chapter 3, we present the background material necessary to obtain our FPT results. Most of Chapter 3 revolves around the concepts of tree-width and parameterized

[4] W[2]-hardness is the FPT analogue of NP-Hardness and polynomial-time algorithms. That is to say, W[2]-hard are unlikely to be solvable using FPT algorithms. The purpose of this thesis is not to explore W[2]-hardness. See [19] for a complete introduction to parameterized complexity theory, including W[2]-hardness.

complexity. In Chapter 4, we prove that the GRID-GRAPH MILLING PROBLEM is NP-Hard and discuss the hardness of its generalizations. In Chapter 5, we show that the GRID-GRAPH MILLING PROBLEM is FPT by using Courcelle's machinery; we also dynamic programming to obtain a more efficient FPT algorithm. In Chapter 7, we summarize our work and discuss possibilities for future research. In the Appendix, we provide a sketch of an improvement on the dynamic programming

CHAPTER 2
Related work in covering tour problems

In this chapter and the next, we review some results pertaining to various geometric covering tour problems and then discuss the theoretical background for our approach to solving these problems. Works presented in this chapter provide examples of various transformations from geometric covering tour problems to graph problems. In general, the geometry problems as well as the transformed graph problems are hard, and the results obtained are approximation algorithms.

The problem we choose to focus on is based on the ORTHOGONAL MILLING PROBLEM as described in [2]. The ORTHOGONAL MILLING PROBLEM is to find a minimum turn cost tour for a unit square cutter so that it exactly cuts a pocket composed of unit square pixels. The cutter must move in an axis parallel direction and the cutter is not allowed to cut outside the pocket. This geometry problem is equivalent to the GRID-GRAPH MILLING PROBLEM (See Figure 2–1). The authors of [2] also introduced a special case of the ORTHOGONAL MILLING PROBLEM, known as the THIN-CHANNEL MILLING PROBLEM, which has applications to snow removal and street cleaning (See Figure 2–2). In this case, it is more natural to consider edge covering tours. The authors showed that finding *minimum-turn orthogonal milling tours* is NP-Hard and presented constant factor approximation algorithms- a 3.75 factor for orthogonal milling and a 4/3 factor for thin-channel milling.

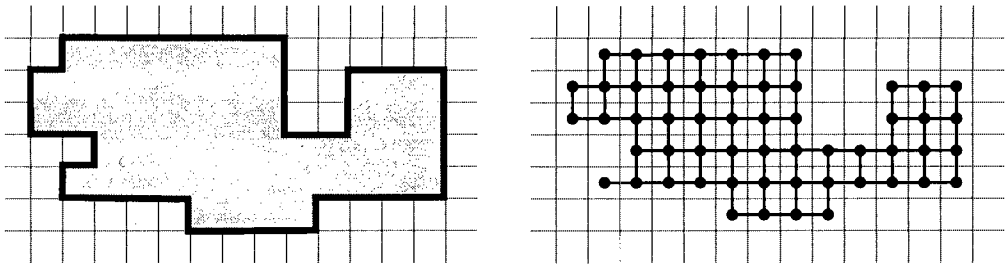

Figure 2–1: An instance of the ORTHOGONAL MILLING PROBLEM (left) and its corresponding grid-graph (right).

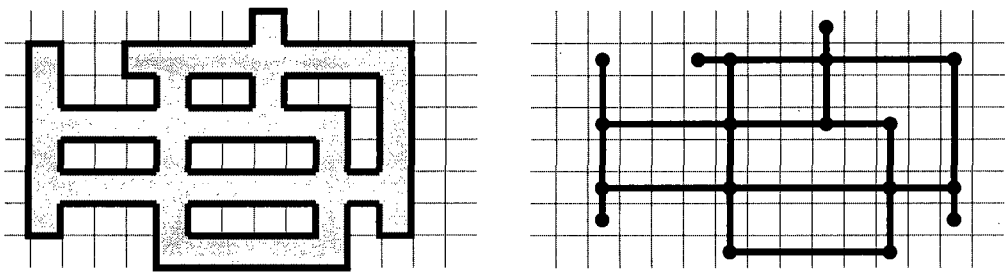

Figure 2–2: THIN-CHANNEL MILLING PROBLEMS: the pocket does not contain a region of 2×2 square pixels (left). This problem can be modeled as an edge covering problem on the corresponding graph (right).

The underlying technique used in these approximation algorithms is to merge minimum *strip covers* and minimum-turn *boundary cycle covers* (see Figure 2-2) into approximate minimum-turn covering tours. The authors showed that optimum strip covers and boundary cycle covers of grid-graphs can be found by solving matching problems on related graphs. Thus, the approximation algorithms to the orthogonal milling and thin channel milling problems were obtained by transformation to graph problems.

Another covering tour problem is the LAWN-MOWING PROBLEM, which is closely related to the milling problem. The difference between the two problems

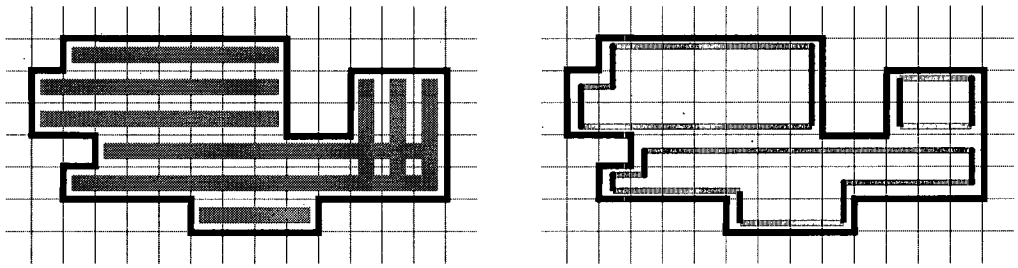

Figure 2–3: A strip cover covers square pixels using horizontal or vertical strips (left) and a boundary cycle cover covers boundary pixels with a set of cycles.

is that in the LAWN-MOWING PROBLEM, the cutter is permitted to cut outside the pocket[1] . In [3], Arkin et al. showed that finding minimum length lawn-mowing tours is NP-Hard. The authors presented approximation algorithms for three versions of the LAWN-MOWING PROBLEM: 1) square cutter with axis aligned motion, 2) square cutter with unrestricted motion, and 3) circular cutter with unrestricted motion. In all versions, the pocket P can consist of several components, each component being a polygon with holes.

These approximation algorithms were obtained by transforming the geometric instance into a grid-graph or grid-like graph (see Figure 2–4). It was shown that a good TSP tour in the grid-graph corresponds to a good lawn-mowing tour in the geometry problem. Specifically, a TSP tour within α of the optimal TSP tour produces a lawn-mowing tour within $3\alpha\beta$ of optimum. Here, β depends on the cutter shape and motion. Specifically, $\beta = 1$, $\beta \approx 1.08$, and $\beta \approx 1.15$ for the three cases

[1] When milling, we must take care not to gouge material outside the pocket. When lawn-mowing, we are happy as long as the entire lawn is mowed.

described in the previous paragraph. Two well-known approximation algorithms for finding TSP tours are Christophede's $\frac{3}{2}$-factor approximation algorithm and Arora's PTAS[2] . These two algorithms can then be used to obtain $\frac{9}{2}\beta$ and $3(1+\epsilon)\beta$ approximations, respectively, for the lawn-mowing problem.

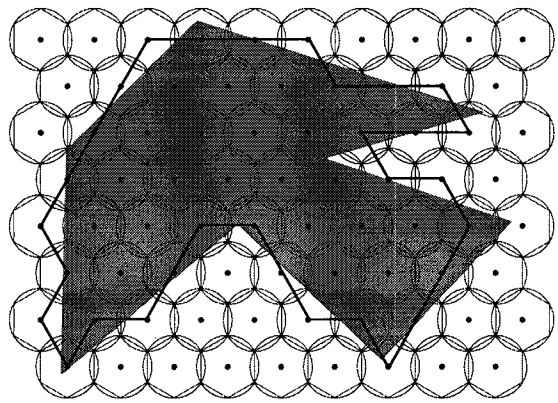

Figure 2–4: The shaded region is P, the pocket to be cut. The background is a hexagonal lattice, where the diameter of the hexagons is equal to the diameter of the circular cutter. The graph problem is to find a minimum length tour covering all vertices enclosed within the solid border.

Yet another covering tour problem is the TRAVELING CAMERAMAN PROBLEM (TCP), which is to cover a region (a set of connected components) with unit square "snapshots " and find a tour through the centers of those squares. An application for this problem is automated inspection of printed circuit boards. The goal is to minimize the length of the tour, or more generally, to minimize a combination of the number of snapshots and the length of the tour. The authors of [24] showed

[2] These algorithms are for the Euclidean TSP problem, which includes TSP on grid-graphs

this problem to be NP-Hard and studied the rectilinear version of this problem. The authors provided an approximation algorithm, which is to first find a minimal square covering and then solve the TSP instance for the center points of the squares in the covering. An optimal square covering along with an optimal TSP tour on the centerpoints does not in general produce an optimal TCP solution. Using known approximation algorithms for these two sub-problems, the authors showed that the rectilinear TCP can be approximated to within $(9 - \epsilon)$ in polynomial time.

In practice, research on pocket milling has focused on automatically generating feasible covering tours. See [22] for a survey on generating *contour-parallel* and *directional-parallel* covering tours. Contour-parallel tours follow the contour of the pocket, resembling the altitude map of a mountain. Directional-parallel tours simply go back and forth in a prescribed direction. In the context of directional milling, one important optimization problem is to minimize the number of tool retractions[3] . It was shown in [4] that minimizing tool retractions is NP-Hard for general pockets with holes. In the same work, the authors provided the first approximation algorithm (constant factor 5) and FPT result for minimizing tool retractions. Both algorithms are based on finding good machining paths in the *machining graph* (see Figure 2–5).

Due to their wide range of applications geometric covering tour problems have received much attention in the literature. Most of these problems are hard, but have

[3] Retracting the cutting tool and re-positioning it takes extra time and can negatively affect the quality of the pocket being cut.

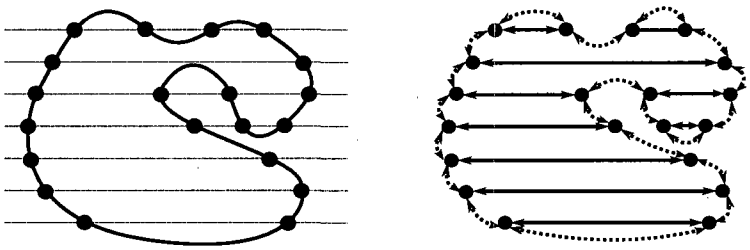

Figure 2–5: Boundary for a pocket (left) and its machining graph (right). Any pair of nodes not joined by solid or dotted edges is implicitly joined by a *retraction edge*. An optimal machining path is a path covering all solid edges exactly once while using each dotted edge at most once and minimizing the number of retraction edges used.

constant factor approximation algorithms. An important part of these approximation algorithms is to solve related combinatorial problems in graphs. Various versions of geometric covering tours can be approximated by the classic TSP problem. Approximation algorithms for minimizing other cost measures also make use of various known graph-theoretic algorithms.

CHAPTER 3
Theoretical Background

In this chapter, we present the theoretical background for our results. The main concepts covered are tree-width, parameter-tree-width bounds, dynamic programming, and monadic second order logic of graphs. In Section 3.1, we introduce the basic notation used in the rest of this thesis. In Section 3.2, we introduce parameterized complexity and width parameters, which are central ideas in this thesis. In Sections 3.3-3.6, we discuss two techniques used to solve problems on bounded tree-width graphs. In Section 3.7, we show how the previous two techniques are related to parameterized complexity theory. In Section 3.8, we present a detailed example incorporating the preceding material. Finally, in Sections 3.9-3.10, we discuss two FPT methods that are not tree-width based, but can be useful for our problem.

3.1 Notation and definitions

We use standard notation and definitions from graph theory. A graph $G = (V, E)$ is a pair: $V(G)$ is the set of vertices of G, and $E(G) \subseteq V(G) \times V(G)$ is the set of edges of G. We study simple[1] , undirected graphs. The *neighborhood* of v, denoted $N(v)$, is the set of vertices adjacent to v. The *subgraph induced by* $W \subseteq V$ is $G[W] = (W, \{(u, v) \in E \mid u, v \in W\})$.

[1] No parallel edges or loops

Two graphs H and G are *isomorphic* if there is a bijection $f : V(H) \to V(G)$ such that $(u, v) \in E(H) \Leftrightarrow (f(u), f(v)) \in E(G)$. *Contracting* an edge (u, v) means identifying its endpoints. Specifically, the vertices u and v are removed along with the edge (u, v) and a new vertex x is added along with edges between x and all vertices that were adjacent to u or v (except u and v). We say H is a *minor* of G if H is isomorphic to some graph resulting from a sequence of contractions, edge removals, or vertex removals in G.

A *plane graph* is a pair $G = (V, E)$ drawn on the plane satisfying the following conditions:

1. vertices are drawn as points;
2. edges are drawn as curves between vertices; and
3. the interiors of edges do not intersect vertices or edges.

Planar graphs are combinatorial graphs that can be drawn on the plane with the above conditions satisfied. In the context of plane graphs, "vertices" ("edges") can refer to either their drawings on the plane or the vertices (edges) in the underlying planar graph. A Σ-*plane graph* denotes a graph drawn on a surface Σ, and $\mathbb{S}_g$-*graphs* are graphs that can be drawn (without edge crossings) on a surface of genus g.

3.2 Parameterized complexity

When faced with an NP-Hard graph problem, one can hope to find algorithms to solve instances restricted to certain graph classes, such as trees, series-parallel graphs, planar graphs, etc. Many hard graph problems are easy on trees and series-parallel graphs. In general, the complexity of any computational problem depends

on its input. This was one motivation behind the development of parameterized complexity theory, defined below.

Definition 3.2.1 *A graph parameter is a function* $P : G \to \mathbb{N}$.

An example of a graph parameter is P_{VC}, which assigns to each G the minimum cardinality of a vertex cover of G.

Definition 3.2.2 *A parameterized problem is a language* $L \subseteq \Sigma^* \times \mathbb{N}$, *where* Σ^* *is the set of finite strings in the given alphabet* Σ, *and* k *is the parameter.* L *is fixed parameter tractable (FPT) if there is an algorithm that answers the question "*$(G, k) \in L$*?" with running time in* $O(f(k)n^{O(1)})$.

Instead of the formal definition, we present parametrized problems by specifying the input, the parameter, and the problem to be solved on the input and parameter. One way to view FPT problems is that they are tractable "by the slice." That is to say, FPT problems are tractable for instances where the parameter is bounded by a constant. Thus, we can study the complexity of a problem under different restrictions by varying our choice of the parameter. These ideas are illustrated by the next example.

The VERTEX COVER PROBLEM is as follows. Given a graph G and k, compute whether G has a vertex cover using at most k vertices. From the perspective of classical complexity theory, an algorithm solving this problem is considered efficient if its running time is bounded by a polynomial in the length of the input[2] . From

[2] We do not address the technical issue of how G and k are encoded, as it is not important in this discussion.

the perspective of parameterized complexity, an algorithm solving this problem is considered efficient if its running time is bounded by a polynomial in the length of the input and any function in the given parameter. Thus, each problem can be parametrized in many different ways, and the resulting parametrized problems can have different complexities. Using the VERTEX COVER PROBLEM as an example, we can use k, which is the desired size of the vertex cover, as the parameter. We will see that this parametrized problem is FPT, meaning that it can be efficiently solved for graphs having small vertex covers. We could have chosen many other parameters for the VERTEX COVER PROBLEM. For example, this problem is also tractable when parametrized by the tree-width of the input graph; i.e., it is easy to compute minimum vertex covers for graphs having low tree-width. This problem is probably not tractable when parametrized by the maximum degree of the graph, since vertex cover remains NP-Hard on graphs with maximum degree 3. Thus, we can see how different choices of the parameter correspond to the complexities of the VERTEX COVER PROBLEM under various restrictions to the input graph.

3.3 Tree-width

Now we consider tree-width, which has the desirable property that many NP-hard problems are easy for bounded tree-width graphs. As its name suggests, tree-width measures how closely a given graph resembles a tree. Graphs with small tree-width are more tree-like. It is often possible to extend algorithms on trees to algorithms on graphs with small tree-width.

Tree-width can be defined in many ways. Consider the following recursive definition of trees. A vertex is a tree. Adding a new vertex along with one edge joining

it to some vertex of an existing tree forms a new tree. This definition is the basis for k-trees, defined below [6].

Definition 3.3.1 *A k-clique is a k-tree. Let G be a k-tree, and let U be a k-clique of G. Then $G' = (V(G) \cup \{v\}, E(G) \cup \{(u, v) \mid u \in U\})$ is a k-tree.*

A graph is a *partial k-tree* if it is a subgraph of some k-tree. Finally, the tree-width of a graph G is defined to be the smallest k for which G is a partial k-tree. There are efficient algorithms that use k-trees to solve NP-Hard problems such as GRAPH ISOMORPHISM and EDGE COLORING [7, 37].

We focus on the concept of tree decompositions, which provides the basis for the most well-known definition of tree-width. The idea of tree decompositions is an important ingredient in Robertson and Seymour's work on graph minor theory [28].

Definition 3.3.2 *A tree decomposition of a graph $G = (V, E)$ is a pair $(T, \mathcal{X})$, where T is a tree and $\mathcal{X} : V(T) \rightarrow 2^{V(G)}$ is a function assigning a subset of $V(G)$ to each node of the tree, satisfying the following conditions:*

1. *for any vertex $x \in V$, there exists $v \in T$ such that $x \in \mathcal{X}(v)$;*
2. *for any edge $e = (x, y) \in E$, there exists $v \in T$ such that $\{x, y\} \in \mathcal{X}(v)$;*
3. *for any vertex $x \in V$, the set of nodes $\{v \mid x \in \mathcal{X}(v)\}$ induces a subtree of T.*

See Figure 3–1 for an example of a tree decomposition.

The subsets in the range of $\mathcal{X}$ are called *bags* of the tree decomposition. The width of a tree decomposition is $|B| - 1$, where B is a maximum size bag in $\mathcal{X}$. The tree-width of a graph is the minimum width among all its tree decompositions. Trees are graphs with tree-width 1, and series-parallel graphs are graphs with tree-width 2.

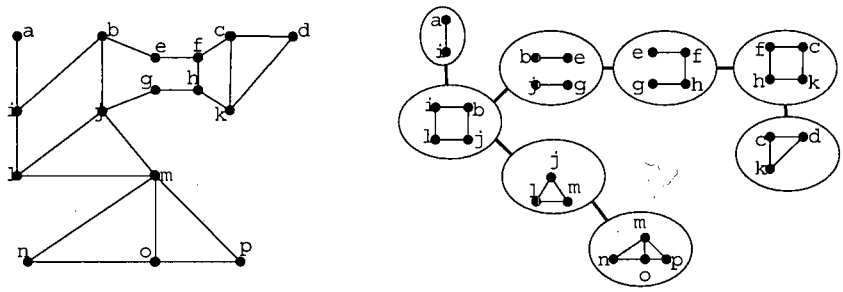

Figure 3–1: A graph (left) and a tree decomposition (right). The subsets in $\mathcal{X}$ are represented by ovals, whose interconnections form a tree. The width of this tree decomposition is 3.

3.4 Dynamic programming on bounded tree-width graphs

Dynamic programming can be used to solve problems that have *optimal substructure* - meaning that the optimal solution of subproblems can be used to compute the optimal solutions of the original problem.

The simple structure of trees naturally lends itself to dynamic programming. When dynamic programming is used on trees, a "subproblem" almost always refers to a generalized version of the original problem restricted to a subtree. To demonstrate this idea, let us use dynamic programming to solve the MINIMUM VERTEX COVER PROBLEM on trees. Let T be a tree rooted at r. For each node v, define T_v to be the subtree rooted at v. Each subtree T_v has two subproblems, whose optimal solutions are defined as:

- $A(v) = |S|$, where S is a minimum vertex cover of T_v such that $v \in S$;
- $B(v) = |S|$, where S is a minimum vertex cover of T_v.

If v is a leaf, $A(v) = 1$ and $B(v) = 0$. Otherwise, let $u_1, \ldots, u_k$ be the children of v. The following relations are used to compute $A(v), B(v)$.

$$\begin{aligned} A(v) &= 1 + B(u_1) + \cdots + B(u_k) \\ B(v) &= \min(A(v), \Sigma_{i=1}^{k} A(u_i)) \end{aligned}$$

The optimal solution to the original problem is $B(r)$. This dynamic programming formulation can be implemented in linear time. The reason $A(v), B(v)$ can be computed from such a simple relation is because 1) there are no edges between T_{u_i} and T_{u_j} for any $i \neq j$, and 2) there are no edges between T_{u_i} and $T - T_v$ for any i. That is to say, every internal node $v \in T$ is a separator, separating T into components $T_{u_1}, \ldots, T_{u_k}, T - T_v$.

These nice properties can be generalized to graphs of bounded tree-width. Let G be a graph and let $(T, \mathcal{X})$ be a tree decomposition of width k. For a given subtree $T' \subseteq T$, define $G[T']$ to be the subgraph induced by the bags of T'; i.e., $G[T'] = G[\bigcup_{v \in V(T')} \mathcal{X}(v)]$. Let v be any internal node in T, and let $u_1, \ldots, u_k$ be its children. It is easy to show from the definition of tree decomposition that there are no edges between any of the following induced subgraphs: $G[T_{u_1}], \ldots, G[T_{u_k}], G[T - T_v]$. That is the say, bags corresponding to internal nodes of a tree decomposition are separators of G.

Most dynamic programming algorithms for solving problems on trees can be generalized to solve problems on bounded tree-width graphs. The subproblems are now defined on the graphs induced by subtrees of the tree decomposition. On graphs

with tree-width at most k, the number and size of subproblems are bounded by some function of k.

As an example, we will generalize the dynamic programming formulation for MINIMUM VERTEX COVER on trees to solve the problem on graphs of bounded tree-width. Before doing so, we describe *nice* tree decompositions, on which dynamic programming is more straightforward.

Definition 3.4.1 *A nice tree decomposition is a rooted tree decomposition such that each node $u \in T$ is one of four types:*

- *leaf: u is a leaf, and $|\mathcal{X}(u)| = 1$;*
- *join: u has two children v_1, v_2, and $\mathcal{X}(u) = \mathcal{X}(v_1) = \mathcal{X}(v_2)$;*
- *introduce: u has one child v, and there is a vertex such that $\mathcal{X}(u) = \mathcal{X}(v) \cup x$;*
- *forget: u has one child v, and there is a vertex such that $\mathcal{X}(v) = \mathcal{X}(u) \cup x$.*

In linear time, a tree decomposition can be converted into a nice tree decomposition with the same width and $O(n)$ nodes [26]. Dynamic programming algorithms on nice tree decompositions tend to have the following structure.

1. Find a nice rooted tree decomposition $(T, \mathcal{X})$ of G with bounded tree-width.
2. In a bottom-up order, compute a table of *partial solutions* for each node of the nice tree decomposition. This is done by considering how to find the partial solutions for each of the four types of the node. The computation for a given node must only use use tables from its children.
3. Extract the solution from the table corresponding to the root.

Let us apply this technique to solve the MINIMUM VERTEX COVER PROBLEM on graphs with tree-width at most k. Assume a nice rooted tree decomposition $(T, \mathcal{X})$

with width at most k is given. The subproblems for a given node $v \in T$ are defined as follows: for each $S \subseteq \mathcal{X}(v)$, find the cardinality of a minimum vertex cover W in $G[T_v]$ such that $W \cap \mathcal{X}(v) = S$. The solution is denoted $A_v(S)$, with the convention that $A_v(S) = \infty$ if no solution exists.

Let us show how to compute $A_v(S)$ for each type of node. Suppose v is a

- leaf node: Then $A_v(\emptyset) = 0$ and $A_v(\mathcal{X}(v)) = 1$;
- join node: Then $A_v(S) = A_{u_1}(S) + A_{u_2}(S) - |S|$, where u_1, u_2 are the children of v;
- forget node: Let $\mathcal{X}(u) = \mathcal{X}(v) \cup \{x\}$, where u is the only child of v. Note that $G[T_v] = G[T_u]$. From this observation, it is easy to see that $A_v(S) = \min(A_u(S), A_u(S \cap \{x\}))$;
- forget node: Let $\mathcal{X}(v) = \mathcal{X}(u) \cup \{x\}$, where u is the only child of v. Observe that any neighbor of x in $G[T_v]$ must appear in the bag $\mathcal{X}(v)$ because x does not appear in any bag of T_v except $\mathcal{X}(v)$.[3] For any $S \subseteq \mathcal{X}(u)$, consider the following two cases:
  1. If S covers all edges incident to x, then $A_v(S) = A_u(S)$ and $A_v(S \cap \{x\}) = A_u(S) + 1$.
  2. Otherwise, $A_v(S) = \infty$ and $A_v(S \cap \{x\}) = A_u(S) + 1$.

The size of the minimum vertex cover is $\min_{S \subseteq \mathcal{X}(r)}(A_r(S))$, where r is the root of the tree decomposition. Each bag of the tree decomposition contains at most $k+1$ vertices, implying that the number of subsets for any bag is at most 2^{k+1}. Computing

[3] This follows from the third property in the definition of tree decomposition.

$A_v(S)$, for any choice of v and S, is constant time in the RAM model. Since there are $O(n)$ nodes in the nice tree decomposition, this dynamic programming formulation can be implemented in $O(2^k n)$ time.

3.5 Computing tree decompositions

Often, a tree decomposition is not given in the input. Instead, only a graph is provided. In this case, a necessary preprocessing step is to find a tree decomposition of low width. This preprocessing step is problem-independent, but it is important because most dynamic programming algorithms on a tree decomposition are exponential time in the width of the given decomposition.

Finding the tree-width of graphs is NP-Hard, even when restricted to graphs of bounded degree, bipartite graphs, and complements of bipartite graphs [9, 25]. The problem is open for planar graphs. The problem is easy for graphs with a polynomial number of minimal separators - for example, chordal graphs[4] [11].

Determining whether a graph has tree-width at most k, and constructing a decomposition when this is the case is FPT [10]. Although the running time is linear in n, it is exponential in the parameter. The reference [10] does not give an explicit running time, as it grows too fast with k to be considered practical (see [32] for experimental results). There are also practical efficient algorithms for specific small values of k. For example, we mentioned that graphs of tree-width 1 are trees, and graphs of tree-width 2 are series-parallel graphs; these classes of graphs are easy to

[4] A graph is *chordal* if every cycle of length > 3 has an edge between non-consecutive vertices

recognize [35]. The best known polynomial time approximation algorithm is a vector programming based algorithm that computes decompositions of width $O(k\sqrt{\log k})$ for graphs of tree-width k.

3.6 Branchwidth

Bounded tree-width graphs are suitable for efficient dynamic programming because they can be decomposed into components separated by small separators. There are other width parameters and corresponding decompositions that can likewise be used for dynamic programming. In this section, we present branchwidth and present some of its advantages over tree-width. Branchwidth is a notion introduced by Robertson and Seymour [29].

Definition 3.6.1 *A branch decomposition of a graph $G = (V, E)$ is a pair (T, μ) where*

1. *T is a ternary tree, and*
2. *$\mu : L \rightarrow E(G)$ is a bijection between the leaves of T and the edges of G.*

Each edge $e \in T$ is associated with a *midset* defined to be $\text{mid}(e) = V(G_1) \cap V(G_2)$, where G_1 and G_2 are the subgraphs induced by the edges associated with the leaves in the two subtrees of $T - e$. Thus, each edge of a branch decomposition partitions the edge set of G. The width of a branch decomposition is the size of the largest midset. The branchwidth of a graph, denoted $\text{bw}(G)$, is the minimum width among its branch decompositions. The branchwidth and tree-width of a graph are related by the following inequality.

Theorem 3.6.2 *For any connected graph with more than 1 edge,* $\text{bw}(G) \leqslant \text{tw}(G) + 1 \leqslant \lfloor 3/2\,\text{bw}(G) \rfloor$) *[30].*

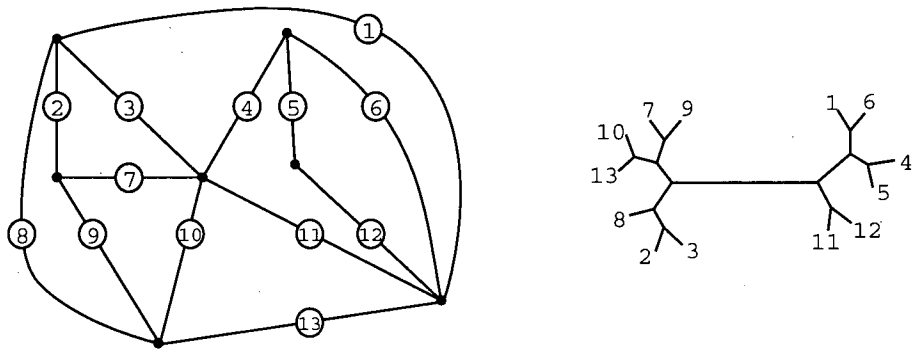

Figure 3–2: (left) A graph with labeled edges and (right) its branch decomposition. The width of this branch decomposition is 3.

Computing the branchwidth of a graph is NP-Hard [34]. When parameterized by branchwidth, the problem is FPT [8]. As is the case with tree-width, this FPT algorithm is not practical. However, unlike tree-width, the branchwidth of planar graphs can be computed by a practical and easily implemented algorithm with $O(n^3)$ running time [34]. Furthermore, there are "geometric branch decompositions," such as *sphere-cut branch decomposition*, which can be used to design more efficient and simpler dynamic programming algorithms for planar graphs [20].

3.7 Monadic second order logic of graphs

One way to show a graph problem is tractable on bounded tree-width graphs is to provide an efficient algorithm- for example, using a dynamic programming formulation. This is the usual algorithmic perspective. In this section, we present a different and theoretically interesting perspective. Instead of designing an algorithm to solve a problem, we describe the problem to a genie, who then provides us with a linear time algorithm to solve the problem. This genie speaks in the language

of *monadic second order logic*, and we must describe our problems in the genie's language in order to obtain linear time algorithms for them.

Before describing monadic second order logic, we introduce some more background on logic and descriptive complexity. Descriptive complexity theory provides a way to classify the complexity of problems. This type of theory classifies the complexity of a problem based on its *expressibility* in some logic.

We assume the reader is familiar with first order logic. Second order logic extends first order logic by allowing quantification over arbitrary finite-ary functions and relations, instead of quantification over just individual elements in the domain. The so called existential second order logic formulas, denoted $\exists SO$, are those of the form $\exists X_1, \ldots, X_k \phi$, where $X_1, \ldots, X_k$ are variables for finite-ary relations and ϕ is a formula of first order logic.

We say a graph G *models* a formula ϕ of some predefined logic L, denoted $G \models \phi$, if there is a satisfying assignment of vertices, subsets of vertices, edges, and subsets of edges of G to the variables of corresponding types in ϕ. A problem P is *expressible* in L if there exists a ϕ such that $G \models \phi$ if and only if G is a yes-instance of P. When there is no ambiguity, we use L to denote the class of all problems expressible in L. Now, we present a key result from descriptive complexity.

Theorem 3.7.1 *$NP \subseteq \exists SO$ [21]*

This result is, in a sense, a hardness result about the worst-case complexity of problems expressible in $\exists SO$. This is because NP-Complete problems, for which we do not expect to be efficiently solvable, are expressible in $\exists SO$. We are looking for a different type of result, one that provides efficient algorithms for all problems

expressible in some logic L. Since our focus is on problems tractable for bounded tree-width graphs, we are interested in finding a logic L such that all problems expressible in L can be efficiently solved on bounded tree-width graphs.

The logic $\exists SO$ is too expressive to be the one we seek because there are NP-Complete problems that are apparently difficult on trees. Since all NP-Complete problems are expressible in $\exists SO$, and trees are trivially bounded tree-width graphs, it is unlikely that all problems expressible in $\exists SO$ can be efficiently solved even when restricted to bounded tree-width graphs.

The logic we seek is a restriction of second order logic called *monadic second order logic*. Monadic second order logic allows quantification over only unary relations (or equivalently, subsets), instead of quantification over arbitrary finite-ary relations. We use MS_2 to denote monadic second order logic of graphs. In this case, the graph structure is given by $G = (V, E, I)$, where V and E are the vertex and edge sets, and I is the incidence relation between vertices and edges. MS_2 has the following predicates:

1. $x \in X$, where x is a vertex(edge) variable and X is a vertex(edge) subset variable[5] ;
2. Adj(u, v), where u, v are vertex variables;
3. Inc(v, e), where v is a vertex variable and e is an edge variable; and
4. equality testing for vertex/edge/vertex subset/edge subset variables.

[5] We use the convention that lower-case variable names are used for single elements and upper-case variable names are used for subsets.

Problems expressible in MS_2 are efficiently solvable for bounded tree-width graphs. This result, known as Courcelle's Theorem, is stated below.

Theorem 3.7.2 *Courcelle's Theorem [15]: Any graph property that is expressible in MS_2 can be recognized in linear time on bounded tree-width graphs.*

There are many NP-hard problems that are expressible in MS_2. The above theorem tells us that all such problems are solvable in linear time on graphs of bounded tree width. Furthermore, the proof of the theorem is constructive in the sense that one could, in principle, implement an algorithm that takes in an MS_2 formula and outputs a linear time algorithm solving the corresponding problem. As an example, we show that 3-COLORABILITY, an NP-hard problem, is expressible in MS_2 and thus efficiently solvable on bounded tree-width graphs. The following formula expresses 3-COLORABILITY.

$$\begin{aligned}\phi = \exists X_1, X_2, X_3 \quad & \forall \quad u(u \in X_1 \vee u \in X_2 \vee u \in X_3) \\ & \forall \quad u(\neg(u \in X_1 \wedge u \in X_2) \wedge \neg(u \in X_2 \wedge u \in X_3) \wedge \neg(u \in X_1 \wedge u \in X_3)) \\ & \forall \quad u, v[\neg \operatorname{Adj}(u, v) \vee \\ & \qquad (\neg(u \in X_1 \wedge v \in X_1) \wedge \neg(u \in X_2 \wedge v \in X_2) \wedge \neg(u \in X_3 \wedge v \in X_3))]\end{aligned}$$

Here, all variables are for vertices or vertex subsets. The subsets X_1, X_2, X_3 correspond to the vertex coloring. The first line checks that each vertex is given a color. The second line checks each vertex is given at most one color. The third and fourth lines check that adjacent vertices are not given same colors. Thus, any graph that models ϕ is 3-colorable.

There are many possible extensions to MS_2. An useful extension for solving optimization problems is *LinEMSOopt* (linear evaluation MSO optimization problem). First, augment the input graph with m functions (weights) $f_1^G, \ldots, f_m^G$ that evaluate vertices or edges to rational numbers. Extend these functions to subsets of vertices or edges as follows: for any subset A, $f_i^G(A) = \Sigma_{a \in A} f_i^G(a)$. Using thse functions, *linear terms* can be built using "+" and "−". *Linear evaluation relations* "=" and "<" can be used to compare linear terms with rational constants. A *LinEMSOopt* problem is defined as follows.

Definition 3.7.3 *Let ϕ be an MSO formula, and let ϕ_1 be a linear evaluation relation and let ϕ_2 be a linear evaluation term. A problem is a LinEMSOopt problem if it can be stated in the following form:*

$$\min\{\phi_2(A_1, \ldots, A_k) \mid G \models \phi(A_1, \ldots, A_k) \text{ and } \phi_2(A_1, \ldots, A_k) \text{ holds}\},$$

where $A_1, \ldots, A_k$ are vertex or edge subset variables.

As is the case with MS_2, problems expressible in LinEMSOopt are also solvable in linear time on bounded tree-width graphs.

Theorem 3.7.4 *Any graph property that is expressible in LinEMSOopt can be recognized in linear time on bounded tree-width graphs.[5]*

An example of a LinEMSOopt problem is MINIMUM WEIGHT VERTEX COVER. Let G be a graph, and let $f : V(G) \to \mathbb{Q}^+$ be the weights of vertices. The formula $\phi(A) = \forall v[v \in A \vee \exists u(u \in A \wedge \text{Adj}(u, v))]$ is true if and only if A is a vertex cover in G. For this problem, we do not need ϕ_1. Trivially, $\phi_2(A) = f(A)$, so that a vertex cover A minimizing $\phi_2(A)$ is a minimum weight vertex cover. Thus, MINIMUM

WEIGHT VERTEX COVER is expressible in LinEMSOopt using the ϕ, ϕ_1, ϕ_2 we have just defined.

Another useful extension arises by labeling of vertices (edges). That is to say, we apply labels, from a predetermined finite set, to vertices (edges) of the input graph and test for these labels within the MS_2 formulation. For example, we could color the vertices of the input graph with two colors - red and green. Then a vertex cover using only red vertices can be checked using the following formula: $\phi'(A) = \phi(A) \wedge \forall v(v \in A \Rightarrow Red(v))$, where ϕ is the formula given in the previous example.

Lastly, the expressive power of MS_2 can be enhanced by a technique called *semantic augmentation.* We say a problem P is expressible in MS_2 via semantic augmentation if there exists a ϕ and $\mathcal{M} : \mathbb{G} \to \mathbb{G}$, where $\mathbb{G}$ denotes the set of all graphs, such that G is a yes-instance of P if and only if $\mathcal{M}(G) \models \phi$. We call $\mathcal{M}(G)$ the *augmented graph* because it is often the original G augmented with new vertices and edges so that certain properties of G that are not expressible in G are expressible in $\mathcal{M}(G)$. In the context of FPT design, we require $\mathcal{M}$ to be polynomial time computable and to preserve tree-width bounds. By the latter we mean that there exists a function f such that $\text{tw}(\mathcal{M}(G)) < f(\text{tw}(G))$ for all G. Courcelle's Theorem imples that problems expressible in MS_2 via semantic augmentation can be efficiently solved for graphs of bounded tree-width. Our MS_2 formulation for the GRID-GRAPH MILLING PROBLEM will use semantic augmentation.

While MS_2 and its extensions are extremely powerful in their expressiveness, it is usually impractical to implement an algorithm for all but the most simplest

formulas[6] . This is because a formula with length k and q quantifier alternations[7] translates to an automaton with $2^{2^{\cdot^{\cdot^{\cdot^{k}}}}}$ states, where q is the height of the exponential tower. In fact, the exponential tower blow-up is unavoidable in this approach [27].

3.8 Parameter-tree-width bounds

In the previous sections, we presented two techniques used to solve problems on bounded tree-width graphs. These techniques produce algorithms with $f(w)n^{O(1)}$ running times, where w is the tree-width. While the right parameter for a problem is application dependent, many graph problems have natural, canonical parameters. In this section, we discuss the relationship between these parameters and tree-width, as well as its algorithmic consequences.

Each graph parameter P has a related parameterized problem "is $P(G) \leqslant k$" or "is $P(G) \geqslant k$"? We say this problem has a *parameter-tree-width bound* if there exists a function f such that if $\mathrm{tw}(G) > f(k)$ then the instance is easy to solve. Often, "easy to solve" means the solution is trivially "Yes" or "No."

Parameter-tree-width bounds can be used in conjunction with tree decomposition based algorithms to produce FPT algorithms. To see this, suppose we are given a problem with a parameter-tree-width bound as well as an algorithm A that solves the problem in $g(w)n^{O(1)}$ time for graphs with tree-width w. Then the following algorithm is an FPT algorithm that solves the problem for input (G, k):

---

[6] The basic idea is to translate MS_2 formulas into tree automatons. See [33].

[7] The number of quantifier alternations in a formula is the number of times an existential (universal) quantifier is followed by an universal (existential) quantifier.

1. Determine the $\mathrm{tw}(G)$ in FPT time (see Section 3.5);
2. If $\mathrm{tw}(G) > f(k)$, then output the trivial "Yes" or "No" solution;
3. Otherwise, apply algorithm A to solve the problem.

The running time of the last step (if neccessary) is at worst $g(f(k))n^{O(1)}$. Thus, this is an FPT algorithm parameterized by k. In the next section, we discuss techniques for establishing parameter-tree-width bounds.

3.8.1 Tree-width, grid minors, and bidimensionality

In order to prove parameter-tree-width bounds, one must use the structural properties of graphs with large tree-width. The property that is most often used is based on *grid minors.*

Definition 3.8.1 *The $k \times k$-grid, denoted Q_k, is the plane graph with vertex set $\{(i,j) \mid 0 \leqslant i,j \leqslant k-1\}$ and edges between vertices distance 1 apart.*

It is not difficult to see that the $k \times k$-grid has tree-width k, and thus any graph containing the $k \times k$-grid as a minor has tree-width at least k.[8] The converse is true "up to a constant" for planar graphs, and provides a useful tool for proving parameter-tree-width bounds.

Theorem 3.8.2 *Let $k \geqslant 1$ be an integer. Every planar graph with no $(k \times k)$-grid as a minor has branchwidth $\leqslant 4k-3$. and tree-width $\leqslant 6k-5$. [31]*

[8] Tree-width, branchwidth, as well as many other width parameters are *minor-cloesd*; i.e., if H is a minor of G, then $\mathrm{tw}(H) \leqslant \mathrm{tw}(G)$.

To show how Theorem 3.8.2 can be used to prove parameter-tree-width bounds, we prove the VERTEX COVER PROBLEM restricted to planar graphs has a parameter-tree-width bound (using the desired size of the vertex cover as the parameter). Let $c = \lceil\sqrt{k+2}\rceil$. Note that Q_c contains more than $4k$ edges. Since any vertex is incident to at most 4 edges, it is not possible for k vertices to cover all edges of Q_c.

Let G be a planar graph. By Theorem 3.8.2, if G has tree-width greater than $\lceil 6\sqrt{k+2}\rceil - 5$, then G contains a minor isomorphic to Q_c. Since it is impossible to cover Q_c using k vertices, it is also impossible to cover G using k vertices.

Note that this result applies only to planar graphs, and thus our parameter-tree-width bound only applies to the VERTEX COVER PROBLEM restricted to planar graphs. In the next section, we present a generalization of Theorem 3.8.2 to larger graph classes.

3.8.2 Bidimensionality

Bidimensionality is a concept defined by Demaine et al. [16] to study graph parameters of graphs containing large grids.

Definition 3.8.3 *A graph parameter P is $g(r)$-bidimensional if*

1. *$P(G) \geqslant g(r)$ for an $r \times r$ grid, and*
2. *for all G in a given family of H-minor-free graphs, $P(G)$ does not increase when taking minors.*[9]

[9] H-minor-free graphs are the family of graphs that do not contain a fixed H as a minor.

Examples of bidimensional parameters include: the number of vertices, the number of vertices in a minimum VERTEX COVER, DOMINATING SET, OR FEEDBACK VERTEX SET [16]. In fact, these parameters are $O(r^2)$-bidimensional. In a long series of papers, Demaine et al. developed the theory of bidimensionality and studied its algorithmic consequences. From the perspective of FPT algorithm design, the key result is that bidimensional parameters have parameter-tree-width bounds.

Theorem 3.8.4 *If a parameter P is $g(r)$-bidimensional, then for every graph G in the family associated with the parameter P,* $\mathrm{tw}(G) = O(g^{-1}(P(G)))$ *[16].*

Note that Theorem 3.8.2 implies that bidimensional parameters have parameter-tree-width bound for planar graphs. In order to prove Theorem 3.8.4, Demain et al. generalized 3.8.2 to hold for H-minor-free graphs.

Theorem 3.8.5 *For any fixed graph H, every H-minor-free graph of tree-width w has an $\Omega(w) \times \Omega(w)$ grid as a minor [16].*

Furthermore, this bound is tight up to a constant factor, which depends on H. This implies that the tree-width bound is also tight for bidimensional parameters. Although we omit the details here, it is worth mentioning that to prove Theorem 3.8.5, Demaine et al. extended known combinatorial results on planar graphs to results on bounded genus graphs, then "almost-embeddable graphs," and finally, clique sums of "almost-embeddable graphs." Thus, statements analogous to Theorem 3.8.5 are also true of these graph classes. It is also known that general graphs with tree-width greater than 20^{2r^5} contain $r \times r$-grid minors [31]. While this result can be used to prove parameter-tree-width bounds, the resulting bound does not lead to efficient FPT algorithms.

Prior to the introduction of bidimensionality theory, many parameter-tree-width bounds were obtained using Theorem 3.8.2. In retrospect, many of these previously established parameter-tree-width bounds can be obtained from the theory of bidimensionality. Furthermore, bidimensionality can be used to extend the tree-width bounds to larger graph classes. For example, we used Theorem 3.8.2 to prove VERTEX COVER on planar graphs has bounded tree-width. We can use the same argument to show that the size of a minimum vertex cover is a bidimensional graph parameter. It follows immediately from Theorem 3.8.4 that VERTEX COVER on H-minor-free graphs has bounded tree-width. This result, in conjunction with the tree decomposition-based dynamic programming for VERTEX COVER, implies VERTEX COVER on H-minor-free graphs is FPT. More generally, Demaine et al. stated the following:

Theorem 3.8.6 *Suppose a $g(r)$-bidimensional parameter P can be computed in $f(w)n^{O(1)}$ time, if a tree decomposition of G with width at most w is given. Then there is an FPT algorithm deciding $P(G) \leqslant k$ on any graph in P's corresponding graph class with running time $[f(O(g^{-1}(k))) + 2^{g^{-1}(k)}]n^{O(1)}$ [16].*

This result can be used to obtain FPT algorithms for many bidimensional parameters, such as VERTEX COVER, DOMINATING SET, FEEDBACK VERTEX SET, etc. for all H-minor-free graphs. As an aside, we mention that bidimensionality can also be used as a framework to design polynomial time approximation schemes and to prove many previously established separator theorems.

3.9 Example: finding Hamiltonian cycle on planar graphs

So far, we have seen techniques for solving problems on bounded tree-width graphs and for proving parameter-tree-width bounds. We have seen how these techniques can be used together to obtain FPT results for parameterized problems. In this section, we study how these techniques can be applied to the HAMILTONIAN CYCLE PROBLEM and the related k-CYCLE parameterized problem[10] on embedded graphs. As we have seen in Chapter 2, the HAMILTONIAN CYCLE PROBLEM is a type of covering tour problem.

The HAMILTONIAN CYCLE PROBLEM is, of course, NP-Complete on planar graphs. Hamiltonicity is a property expressible in MS_2. To see this, note that a graph contains a Hamiltonian cycle if every cut has at least two edges crossing it. This property is expressed by the following MS_2 formula ϕ:

$$\phi = \forall X_1, X_2 \quad [\quad \forall x \left((x \in X_1 \wedge x \notin X_2) \vee (x \notin X_1 \wedge x \in X_2)\right) \Rightarrow$$
$$\exists u_1 \in X_1, v_1 \in X_1, u_2 \in X_2, v_2 \in X_2(\text{Adj}(u_1, u_2) \wedge \text{Adj}(v_1, v_2))]$$

The variables X_1, X_2 correspond to vertex subset variables. The first line checks that X_1, X_2 form a bipartition of the vertex set (this is the same as a cut). The second line checks that there are two edges between X_1 and X_2 when they form a bipartition. Thus, by Courcelle's Theorem, we know that the HAMILTONIAN CYCLE PROBLEM can be solved in linear time on graphs of bounded tree-width. However, as previously stated, the hidden constant is too large to be considered practical.

[10] Given (G, k), the problem is to find whether G has a cycle of length k.

In [20], Dorn et al. used dynamic programming on sphere-cut branch decompositions to solve the Hamiltonian cycle problem (on planar graphs) in $O(2^{O(k)}kn+n^3)$ time, where k is the branchwidth. To obtain this result, the authors introduced the idea of *sphere-cut (sc) branch decompositions*, which relate combinatorial separators (vertices in a graph) to geometric separators (simple, closed curves on a surface). Sphere-cut branch decomposition is defined below.

A Σ-graph refers to a graph $G = (V, E)$ embedded without edge crossings on the unit sphere Σ. Define a *noose* to be a simple closed curve O in Σ that intersects G at vertices only and intersects every face at most once. A *sphere-cut branch decomposition* is a branch decomposition such that for every edge $e \in E(T)$, there is a noose O_e with $\text{mid}(e) = O_e \cap V(G)$ and $G_j \in \Delta_j \cup O_e$, $1 \leqslant j \leqslant 2$. In other words, the noose O_e partitions the embedded graph in the same way that its corresponding edge in the branch decomposition partitions the purely combinatorial graph. Recall that an optimal branch decomposition can be computed in $O(|V|^3)$ time for planar graphs. This is also true of sc-branch decompositions.

Theorem 3.9.1 *Let $G = (V, E)$ be any graph embedded on the unit sphere, with $\text{bw}(G) = l$ and no vertices of degree 1. Then there exists an sc-branch decomposition of G with width at most l and such a branch decomposition can be constructed in $O(|V|^3)$ time. [20]*

Dorn et al. used sc-branch decomposition to improve on the traditional dynamic programming formulation for the HAMILTONIAN CYCLE PROBLEM. The intuition behind the dynamic programming formulation is to view an optimal tour as a set of

path segments (defined by the endpoints) in subgraphs. The running time is dominated by the number of possible combinations of path segments. Clearly, these path segments cannot pair-wise cross each other. By applying the geometric interpretation of sc-branch decompositions, the authors reduced the number of such combinations to the number of non-crossing matchings[11] (of the vertices on a noose) to achieve the $O(2^{O(k)}kn + n^3)$ running time. Note that traditional branch decompositions do not contain any embedding information. Therefore, the concept of non-crossing matchings is not defined for the midsets. The best running time without using sc-branch decomposition is $O(2^{O(k \log k)}n^{O(1)})$.

A related parameterized problem is the k-cycle problem on planar graphs, which asks whether a given graph contains a cycle of length at least k. It is easy to see that the k-cycle problem on planar graphs has a parameter-tree-width bound. A planar graph with branchwidth at least $4\sqrt{k+1} - 3$ must contain a $\sqrt{k} \times \sqrt{k}$ grid minor, which trivially contains a cycle of length at least k. Thus, the k-cycle problem on planar graphs is FPT by using a slightly modified version of the dynamic programming algorithm for planar Hamiltonian cycle.

[11] Given a set of points lying on the boundary of the unit circle, a non-crossing matching is a pairing the given points such that the straight line segments drawn between them are not pairwise crossing.

To summarize, in [20], planarity was exploited three times: 1) to use sc-branch decompositions, 2) to reduce the number of path combinations in the dynamic program, and 3) to show that the k-CYCLE PROBLEM has a parameter-tree-width bound.

Extension to $\mathbb{S}_g$-graphs

The techniques seen in the previous section can be extended to solve the HAMILTONIAN CYCLE PROBLEM on graphs embedded on a surface of genus g. One technique often used to solve a problem on a bounded genus graph is to reduce the genus until the problem is on a planar graph. In [18], Dorn et al. showed how their approach on planar graphs can be used to solve the HAMILTONIAN CYCLE PROBLEM and the k-CYCLE PROBLEM on bounded genus graphs.

Recall that a noose is a simple closed curve which intersects G only at its vertices. A noose is *noncontractible* if it is cannot contract to a point. "Cutting" along a noncontractible noose on a surface of genus g results in a surface of genus $g-1$.

Let G be a graph embedded on a torus. Let N be a noncontractible noose and let $U = \{u_1, u_2, \ldots, u_k\}$ be the set of vertices intersecting N. We say N is *tight* if U is a connected subset. Define the *left (right) neighbors* of u_i to be the set of neighbors of u_i that appear on the left (right) side of N as we travel clockwise along N. Now consider the graph G' defined as follows.

- $V(G') = (V(G) - U) \cup U_l \cup U_r$, where U_l and U_r are "copies" of U
- $E(G') = E(G) \cup \{(u,v) \mid u \in U_l$ and v is a left neighbor of u$\} \cup \{(u,v) \mid u \in U_r$ and v is a right neighbor of u$\}$

It is clear that G' is planar. Furthermore, a Hamiltonian cycle in G induces a set of disjoint paths $\mathcal{P} = \{P_1, P_2, \ldots, P_l\}$, where each path is between some vertex in U_l and its copy in U_r. Such a set of disjoint paths can be found in $O(2^{O(\mathrm{bw}(G'))}|V(G')|^{O(1)})$ time using a modified version of the algorithm in [20].

To obtain the running time in terms of the size of the graph, it is necessary to bound 1) the number of vertices lying on the noose and 2) the branchwidth of G', as these bound the number of path segment combinations. Bounds for these values are established by the following lemmas.

Lemma 3.9.2 *Let G be a torus-embedded graph and let G' be the planar graph after cutting along a tight noncontractible noose. Then* $\mathrm{bw}(G') \leqslant \sqrt{4.5|V(G)|} + 2$ *[17].*

Lemma 3.9.3 *Any graph G embedded on a torus has a tight noncontractible noose with length at most $\sqrt{4.5|V(G)|} + 2$, and such a noose can be found in polynomial time [17].*

Using the above lemmas, the authors showed that the HAMILTONIAN CYCLE PROBLEM on graphs embedded on a torus can be solved in $O(2^{O(\sqrt{n})}|n|^{O(1)})$ time. This technique can be extended to solve the HAMILTONIAN CYCLE PROBLEM on graphs embedded on surfaces of genus g with running time $O(n^{O(g^2)}2^{O(g\sqrt{gn})})$. Finally, this technique was used in conjunction with bidimensionality theory to obtain an $O(2^{g^2 \log k + g\sqrt{gk}}n^{O(1)})$ algorithm for the k-CYCLE PROBLEM on graphs embedded on surfaces of genus g in the same paper.

3.10 Layer-wise separation

Although tree decompositions and branch decompositions are useful tools for devising FPT algorithms, there other decompositions that use planarity or other

geometric properties of graphs and may be used to give more efficient or simpler FPT algorithms. The reason efficient dynamic programming on bounded tree-width graphs is possible is because such graphs can be decomposed into components separated by small vertex separators. To obtain an FPT result for a given parameterized problem, it is necessary to also prove that it is parameter-tree-width bounded. In this section, we discuss another decomposition structure such that 1) efficient dynamic programming is possible for many problems, and 2) these problems have "parameter-width" bounds.

We refer to decompositions making use of outerplanarity. In [1], Alber et al. described the concept of layer-wise separation of planar graphs, which is defined as follows. Given an embedded planar graph G, let L_1 be the set of vertices on the outer face of G, and let L_i be the set of vertices on the outer face of G after removing L_j for $j < i$.

Definition 3.10.1 *A parameterized problem $\mathcal{L}$ for planar graphs is said to have the layer-wise separation property (LSP) of width w and size-factor d if for every $(G, k) \in \mathcal{L}$ and every embedding of G, there exists a sequence $(S_1, S_2, \ldots, S_r)$*

1. *$S_i \subseteq \bigcup_{j=i}^{i+w-1} L_j$,*
2. *S_i is an $L_{i-1} - L_{i+w}$ separator, and*
3. *$\Sigma_{j=1}^{r} |S_j| < dk$.*

Examples of problems exhibiting the LSP are PLANAR VERTEX COVER (width 2, size-factor 2) and PLANAR DOMINATING SET (width 3, size factor 51) [1]. To see that PLANAR VERTEX COVER has the LSP, consider a planar graph which has a vertex cover V' of size k. Choose $S_i = (L_i \cup L_{i+1}) \cap V'$ for the sequence of separator

sets; the size-factor follows from the observation that any $v \in V'$ appears in exactly 2 separator sets.

Alber et al. showed that problems exhibiting the LSP with width w and size dk have tree-width bounded by $f(k) = 2\sqrt{3dk} + (3w - 1)$ and that this tree decomposition can be computed in $O(k^{3/2}n)$ time. Thus, the LSP can be used as a tool for proving parameter-tree-width bounds, which sometimes leads to better bounds than obtained by previous methods. Furthermore, dynamic programming formulations using layer-wise separation often achieve better running times than dynamic programming formulations using tree decompositions.

CHAPTER 4
NP-Completeness of Grid Graph Milling

In this chapter, we formally define the GRID-GRAPH MILLING PROBLEM and prove that it is NP-Complete. The input to our problem is more general than the grid-graphs we have seen so far in that edges are not restricted to be of length 1. Also, we focus on the "walk" version of the problem, in which it is not necessary for the object to start and end at the same point. It is a simple exercise to extend our results to the "tour" version of this problem. Of course, as our FPT results apply to the more general version, they also apply to the restricted version.

4.1 Problem definition

Let $G = (V, E)$ be a plane graph such that the vertices are placed on grid points and the edges are horizontal or vertical straight lines. The embedding is specified by Γ, which assigns integer coordinates to each vertex. The problem is to find a k-milling walk in G, which is a path that covers all vertices using at most k turns or U-turns. Turns and U-turns are defined as follows:

The Grid-graph Milling Problem

Input: $G = (V, E)$ and $\Gamma : V \to N \times N$;

Parameter: k, the number of turns;

Question: Does G contain a k-milling walk?

Definition 4.1.1 *Let* $W = [v_0, e_1, v_1, \ldots e_i, v_i, e_{i+1}, \ldots e_l, v_l]$ *be a walk. A* turn *in* W *is a vertex* v_i *such that* e_i *is a horizontal edge and* e_{i+1} *is a vertical edge or vice-versa. A* U-turn *in* W *is a vertex* v_i *such that* $v_{i-1} = v_{i+1}$.

4.2 Proof of NP-Completeness

Theorem 4.2.1 *The* GRID-GRAPH MILLING PROBLEM *is NP-Complete*

Proof: We reduce from the HAMILTONIAN PATH PROBLEM in maximal planar graphs [13]. Let $G = (V, E)$ be a maximal planar graph. Since G is 2-connected, there is an polynomial-time computable embedding of horizontal bars on the plane such that each bar represents a vertex in V, and two bars can "see" each other if their corresponding vertices are adjacent [36]. Formally, a bar is a subset of the plane given by $\{(x, y) : y = y_0 \text{ and } x \in [x_0, x_1]\}$; we refer to this set as $B(x_0, x_1, y_0)$. Two bars $B(x_0, x_1, y_0)$ and $B(x'_0, x'_1, y'_0)$ can see each other if there is an $x^* \in (x_0, x_1) \cap (x'_0, x'_1)$ such that the line segment between (x^*, y_0) and (x^*, y_1) does not intersect any other bars. Given the bar embedding, we construct the GRID-GRAPH MILLING instance G' in two steps. In the first step, we start with a set of bars (horizontal line segments on the plane) representing vertices, and we add vertical line segments connecting bars representing to adjacent vertices. In the second step, we start with a set of horizontal and vertical line segments and we define the vertices and edges for the grid-graph. Details follow.

1. For each $(u, v) \in E$: let $B(x_0, x_1, y_0), B(x'_0, x'_1, y'_0)$ be the bars corresponding to u, v. Let x^* be as previously defined. Draw a (vertical) line segment between (x^*, y_0) and (x^*, y_1). When adding vertical line segments, ensure that we do not use the same x^* for different vertical line segments.

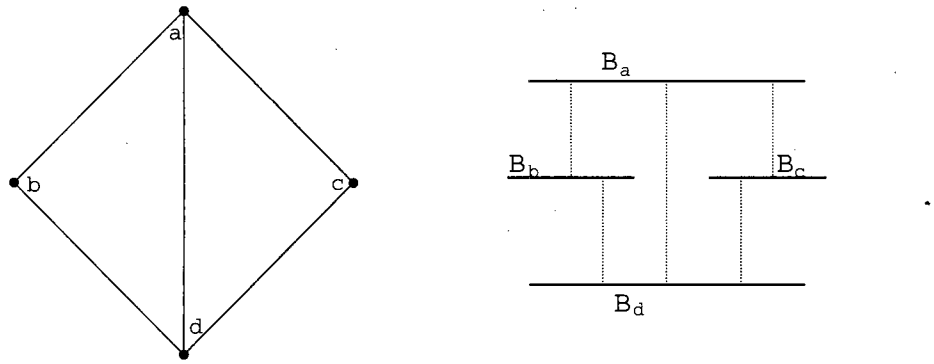

Figure 4–1: (left) A 2-connected plane graph G (right) Bar visibility representation of G, with dotted lines representing visibility

2. Define the vertices of the grid-graph to be the endpoints of both the horizontal and vertical line segments in the drawing. The edges and the embedding of the grid-graph are implicitly given by the drawing.

Lemma 4.2.2 *$G = (V, E)$ contains a Hamiltonian path iff $G' = (V', E')$ can be milled using $k = 4n - 4$ turns, where $n = |V|$.*

We first prove the forward direction. Let $v_1, \ldots, v_n$ be a Hamiltonian path in G. Let $B_1, \ldots, B_n$ be the corresponding bars in the bar visibility representation. Note that the $B_1, \ldots, B_n$ contains all the vertices in the grid-graph. Furthermore, for each B_i there is a vertex in $u_i \in B_i$ that is connected to a vertex in $v_{i+1} \in B_{i+1}$. Let e_i, e_i' be the endpoints (vertices) of bar B_i. The k-milling walk is $e_1, e_1', u_1, v_2, e_2, e_2', u_2, v_3, e_3, e_3', u_3, \ldots, v_n, e_n, e_n'$; here, all vertices on B_i are implicitly included between e_i and e_i'. Note that if we "enter" a bar at a non-endpoint, then exactly four turns are needed to visit every vertex and "exit" this bar. The first and last bars in the sequence requires only two turns, since we start and end on endpoints. Thus, the number of turns in the lawnmower walk is $4n - 4$.

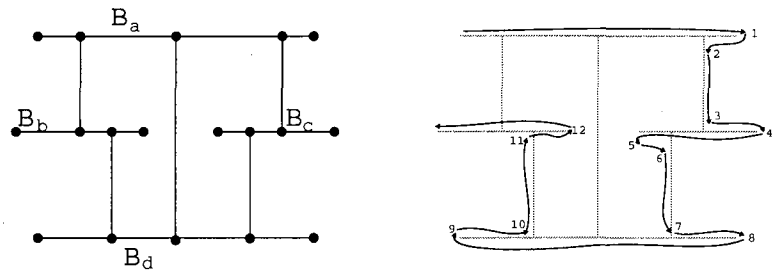

Figure 4–2: (left) The lawnmower instance corresponding to the bar visibility representation of G. (right) A lawnmower walk corresponding to the Hamiltonian path (a, c, b, d) in G. Note that there are exactly $12 = 4n - 4$ turns.

Now we prove the converse. Suppose W is a k-milling walk using at most $4n-4$ turns. We shall consider two cases.

Case 1: W starts and finishes at endpoints of bars. Note that there are $2n$ endpoints in the grid-graph and it costs 1 turn to visit each endpoint except the first and last vertices in the walk; this means at least $2n - 2$ turns are needed just to visit every endpoint. Let $\mathcal{B} = B_1, B_2, \ldots, B_j$ be the sequence of bars corresponding to the sequence of vertices in W. Clearly, every bar is in the sequence at least once. We will show that every bar is in the sequence at most once; this would imply that there is a Hamiltonian path in G.

Suppose to the contrary that $j = n + c$ for some integer $c > 0$. Note that for any i such that $j > i > 1$, it costs at least two turns to enter and leave B_i if $B_{i-1} \neq B_{i+1}$; this is because the $B_{i-1} - B_i$ 'line of sight' edge lies on a different x coordinate from the $B_i - B_{i+1}$ 'line of sight' edge. Otherwise, it costs at least one turn to enter and leave B_i if $B_{i+1} = B_{i-1}$; this case can occur at most c times, because otherwise the sequence could not contain all n bars. Leaving the first bar and entering the last bar also accounts for 1 turn each. Thus, the total number of turns used to enter and leave

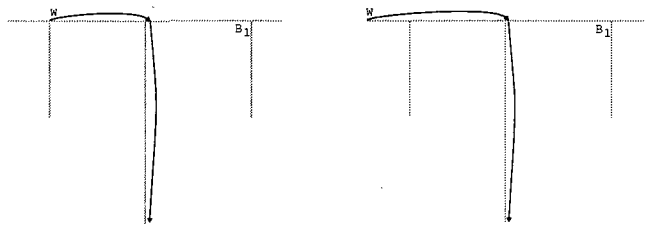

Figure 4–3: (left) If the first two vertices of W lie on B_1 then W can be modified (right) into a new walk which begins at an endpoint of B_1 without using more turns.

bars is at least $2+c+2(n-2) = 2n+c-2$. Note that the turns required for visiting endpoints are distinct from the turns required for entering and exiting bars because by construction bars cannot enter or leave at endpoints. Thus, the total number of turns in W is at least $4n-4+c$. Since $c > 0$, this contradicts the assumption that W uses at most $4n-4$ turns.

Case 2: W starts and finishes at internal vertices of bars. Again, let $\mathcal{B} = B_1, B_2, \ldots, B_j$ be the sequence of bars corresponding to the sequence of vertices in W. Note that if the first two vertices in W are both in B_1, then W can be modified so that it begins at an endpoint without using additional turns (see Figure 4-3), which brings us back to *Case 1.* Otherwise, B_1 must be in the sequence $\mathcal{B}$ at least twice, because in the first visit, the walk W immediately exited bar B_1 without visiting its endpoints. The same observation also applies to B_j: so we know that B_1 and B_j both appear at least twice in $\mathcal{B}$. Again, we count the number of turns needed to enter and exit bars in $\mathcal{B}$. Let $j = n+2+c$ for $c \geqslant 0$. We noted that every bar must appear at least once in $B_2, \ldots, B_{j-1}$. This implies at least $2n+c-2$ turns are needed to enter and exit bars. We also need at least $2n$ turns to visit all the endpoints, since the walk W does not begin or finish at endpoints. This means W

uses at least $4n-c+c$ turns, contracting the assumption that W uses at most $4n-4$ turns.

□

CHAPTER 5
FPT Algorithms for Grid Graph Milling

Chapter 3 established our basic framework for developing FPT algorithms based on the concept of tree-width. To show that a parameterized problem is FPT, we need to show that it has a parameter-tree-width bound and then provide an algorithm (or an MS_2 formula) to solve the problem. In Section 5.1, we show that the GRID-GRAPH MILLING PROBLEM has a parameter-tree-width bound when parametrized by the number of turns. In Sections 5.2 and 5.3, we provide an MS_2 formulation and a dynamic programming formulation, respectively, to solve our problem.

5.1 Parameter-tree-width Bound

Recall that a problem has a parameter-tree-width bound if there exists a function f_1 such that if the input graph G has tree-width greater than $f(k)$, then the problem is easy to solve. We will show that if $\text{tw}(G) > 6k + 13$, then G cannot be milled by a walk using at most k turns.

Theorem 5.1.1 *Let $G = (V, E)$ be a grid-graph. If $\text{tw}(G) > 6k + 13$ or $\text{bw}(G) > 4k + 9$, then G does not contain a k-milling walk.*

Proof: We will prove the tree-width bound; the same proof can be used for the branchwidth bound.

We say two vertices are *rook-independent* if they lie on different rows and columns. Note that if there exists a subset $U \subseteq V(G)$ such that elements of U are pair-wise rook-independent, then any walk that visits every vertex of $V(G)$ must

use at least $|U| - 1$ turns. We will prove that if $\text{tw}(G) > 6k + 13$, then it contains at a set of $k + 2$ pair-wise rook-independent vertices.

Let $k' = k + 3$. Assume $\text{tw}(G) > 6k' - 5$. By Theorem 3.8.2, G contains $Q_{k'}$ as a minor[1] . Notice that $Q_{k'}$ contains a sequence $C_1, C_2, \ldots, C_{\lfloor k'/2 \rfloor}$ of vertex-disjoint nested cycles[2] . We will show that there are 2 vertices per cycle such that these $2\lfloor k'/2 \rfloor$ vertices are pair-wise rook-independent.

First, it is trivial to see that there exists a pair of vertices v_1, v_1' on C_1 such that v_1 lies above and to the right of v_1'. We claim there are vertices v_2, v_2' on C_2 such that v_2 lies above and to the right of v_1, and v_2' lies below and to the left of v_1'. To see this, note that any closed curve on the plane that encloses the origin must traverse all four quadrants. In particular, if we let v_1 be the origin, then the cycle C_2 (which encloses v_1) must contain a vertex in the first quadrant. Similarly, if we let v_1' be the origin, then C_2 must contain a vertex v_2' in the third quadrant. This argument may be repeated inductively to produce a set of $2\lfloor k'/2 \rfloor$ pair-wise rook-independent vertices. □

Remarks. We mentioned in the introduction that finding minimum-turn milling tours is not likely to be FPT for general graphs, but it is FPT when restricted to grid-graphs. Now, we can see why. We need the extra structure of grid-graphs in three different parts of our paramter-tree-width bound proof: 1) to apply Theorem

[1] Recall that $Q_{k'}$ is $k' \times k'$ grid-graph

[2] By a sequence of nested cycles, we mean C_i "encloses" the previous cycles $C_1, \ldots, C_{i-1}$ in the sequence.

3.8.2 (for Grid-graph Minors), 2) to define the notion of rook-independence, and 3) to ensure that going between pair-wise rook-independent vertices costs at least one turn.

5.2 MS_2 formulation using semantic augmentation

The purpose this section is to present an MS_2 formula ϕ such that a graph models ϕ if and only if it can be milled using k turns. Note that the graph structure used in MS_2 is a purely combinatorial structure; i.e., the embedding information of a grid-graph is not used by the MS_2 formulation. Of course, the embedding is important to the actual problem because it is used to evaluate the cost function - the number of turns used. Thus, we need a combinatorial way to encode turns. Recall from Section 3.7 that two extensions to MS_2 are labeling of vertices (edges) and semantic augmentation. These extensions can be used to "store" the *maximal turn-free paths* of the grid-graph G. A *maximal turn-free path* is defined to be a path of maximal length in G whose edges are all vertical (or all horizontal) in the embedding of G.

For each maximal turn-free path P, we create a new vertex $s(P)$ and add edges between $s(P)$ to each vertex in P. Formally, we define the augmented graph $\mathcal{M}(G)$ as follows.

- $V(\mathcal{M}(G)) = V(G) \cup S$ with $S = \{s(P) \mid P \text{ is a maximal turn-free path of } G\}$
- $E(\mathcal{M}(G)) = E(G) \cup E(S)$ with $E(S) = \{(s(P), v) \mid s(P) \in S \text{ and } v \in P\}$

We will label the new vertices using $S(\cdot)$. That is to say, the predicate $S(v)$ evalulates to true if and only if v is a new vertex.

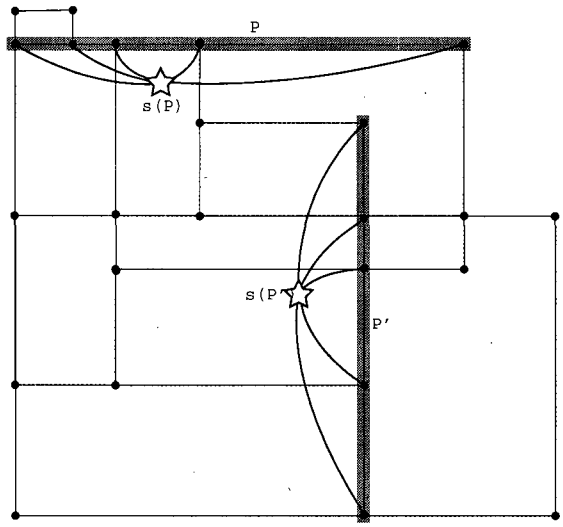

Figure 5–1: A partially augmented graph. The black vertices and dotted edges belong to the original graph. Two maximal turn-free paths, P, P' are highlighted. New vertices $s(P), s(P')$ (stars) are added in the augmented graph. New edges (solid lines) connect $s(P)$ to each vertex in P.

Note that a vertex can belong to at most 2 maximal turn-free paths, so that there are at most $2|V(G)|$ such paths. It is clearly easy to compute $\mathcal{M}(G)$ for any grid-graph G. The following lemma shows that the tree-width does not increase too much after augmenting G.

Lemma 5.2.1 *For any grid-graph G and its augmented graph $\mathcal{M}(G)$, their tree-widths are related by the following inequality:* $\mathrm{tw}(\mathcal{M}(G)) \leqslant 3 \cdot \mathrm{tw}(G) + 2$.

Proof: We will prove the lemma by showing how a tree decomposition of G can be modified into a tree-width decomposition of $\mathcal{M}(G)$ without increasing its width too much.

Let us consider an tree decomposition $(T, \mathcal{X})$ of G of width w. We can modify $(T, \mathcal{X})$ into a tree decomposition $(T, \mathcal{X}')$ of $\mathcal{M}(G)$ by inserting each vertex $s(P) \in S$, which corresponds to a maximal path P, into all bags containing any vertex of P.

To verify that $(T, \mathcal{X}')$ is a valid tree decomposition of $\mathcal{M}(G)$, we must verify that 1) all vertices and edges belong to some bag in $\mathcal{X}$ and that 2) the set of bags contain a given vertex $v \in V(\mathcal{M}(G))$ induces a subtree of T.

The first condition is trivially satisfied, as all new vertices S and edges $E(S)$ are added to some bag. We only need to verify the second condition; that is, we need to verify that the bags containing an arbitrary $s(P)$ vertex forms a subtree in the tree decomposition. Let $s(P)$ be an arbitrary vertex in S. A basic property of tree decompositions is that the set of bags containing any vertex of a connected component in G induces a subtree in T.[3] Thus the set of bags containing any vertex in any maximal path P induces a subtree in T. Since $s(P)$ is added to every bag in that subtree, it follows that the bags containing $s(P)$ indeed forms a subtree.

It remains to bound the width of the tree decomposition $(T, \mathcal{X}')$. Consider any bag $B \in \mathcal{X}$. Since each vertex belongs to at most two maximal turn-free paths, at most $2|B|$ vertices of S may be added to B. Thus, if the tree-width of G is w, then the tree-width of $\mathcal{M}(G)$ is at most $3w + 2$. □

Remarks. So far, we have shown that if a grid-graph G has a k-milling walk, then it has tree-width in $O(k)$. Furthermore, for any such grid-graph, we can compute in polynomial time the augmented graph $\mathcal{M}(G)$ whose tree-width is at most a factor

[3] This can be easily proven by induction on the size of the component U.

3 times the original grid-graph. In the remaining part of this section, we present a formula ϕ such that a grid-graph has a k-milling walk if and only if its augmented graph models ϕ. Then we can apply Courcelle's Theorem and conclude that the GRID-GRAPH MILLING PROBLEM is FPT with turn-cost as the parameter.

Lemma 5.2.2 *Let $G = (V, E)$ be a grid-graph. Having a k-milling walk in G is a property expressible in monadic second order logic on $\mathcal{M}(G)$.*

Proof: Suppose there is a k-milling walk in G. We may identify this walk by a sequence of turn-free path segments $P_0, \ldots, P_k$ such that 1) every vertex is in at least one path segment, and 2) the "endpoint" of path P_i is the "start-point" of path P_{i+1}. These conditions may be expressed in MS_2 on the augmented graph $\mathcal{M}(G)$ as described below.

Before we present ϕ, we need to develop some predicates or "sub-routines" that will be used in ϕ. We will be using only vertex or vertex subset variables. The former are denoted using lower-case letters; the latter are denoted using upper-case letters. First, we need a predicate $\text{TFPath}(P, s)$ that is true if and only if P induces a turn-free path segment in the grid-graph. To do this, we need to check that 1) the subgraph induced by P is connected and 2) every vertex in P is connected to s, which corresponds to some maximal turn-free path segment. Thus, $\text{TFPath}(P, s)$ can be constructed as follows.

$$\text{TFPath}(P, s) = \text{Connected}(P) \wedge S(s) \wedge \forall u(u \in P \Rightarrow \text{Adj}(u, s))$$

Here, we assumed we are given a predicate $\text{Connected}(P)$ that is true if and only if the subgraph induced by P is connected. Note that this is equivalent to the condition

that for every bipartition (X_1, X_2) of P, there is an edge between some vertex in X_1 and some vertex in X_2. We define Connected(P) and Bipartition(X_1, X_2, P) as follows.

$$\text{Connected}(P) = \forall X_1, X_2[\text{Bipartition}(X_1, X_2, P) \Rightarrow \exists u_1, u_2(\text{Adj}(u_1, u_2) \wedge u_1 \in X_1 \wedge u_2 \in X_2)]$$

$$\text{Bipartition}(X_1, X_2, P) = \forall u(u \in P \Rightarrow (u \in X_1 \wedge u \notin X_2) \vee (u \in X_2 \wedge u \notin X_1))$$

Lastly, we need a predicate Endpoint(u, P) that checks whether u is an "endpoint" of P. Here, we assume that P induces a turn-free path, so we only need to check whether u has degree one in the path induced by P.

$$\text{Endpoint}(u, P) = u \in P \wedge \neg\exists x, y(x \neq y \wedge x \in P \wedge y \in P \wedge \text{Adj}(x, u) \wedge \text{Adj}(y, u))$$

Equipped with these predicates TFPath(P, s) and Endpoint(u, P), we can finally present the MS_2 formula ϕ as follows.

$$\begin{aligned}
\phi \;=\; & \exists P_0, \ldots P_k, v_0, \ldots, v_{k+1}, s_0, \ldots s_k && (5.1)\\
& \bigwedge_{i=0}^{k}(\text{TFPath}(P_i, s_i)) && (5.2)\\
& \bigwedge_{i=0}^{k}(\text{Endpoint}(v_i, P_i) \wedge \text{Endpoint}(v_{i+1}, P_i) && (5.3)\\
& \bigwedge \forall u \left(\bigvee_{i=0}^{k} u \in P_i \right) && (5.4)
\end{aligned}$$

As previously stated, a k-milling walk can be identified by a sequence $P_0, \ldots, P_k$ of turn-free path segments. The second line verifies that $P_0, \ldots, P_k$ are indeed turn-free path segments. The third line verifies that the turn-free path segments share

appropriate endpoints (note that this formulation can be modified to express the existence of k-milling tour by setting the first and last endpoint to be the same vertex). The last line checks that every vertex is in some turn-free path segment. $\square$

5.3 Dynamic programming FPT algorithm

In this section, we present a dynamic programming formulation for the GRID-GRAPH MILLING PROBLEM. The basic idea is based on viewing the optimal milling tour as a sequence of path segments induced by geometric decompositions of the grid-graph and using dynamic programming to find these path segments.

In principle, our dynamic programming formulation is the same as the canonical formulations used for the HAMILTONIAN CIRCUIT PROBLEM. The main difference is that path segments in the GRID-GRAPH MILLING PROBLEM do not have to be disjoint. Thus, we cannot bound the number of combinations using non-crossing matchings, as was done for the HAMILTONIAN CIRCUIT PROBLEM. Another difference is that the GRID-GRAPH MILLING PROBLEM is a parametrized problem with a parameter-tree-width bound, whereas the HAMILTONIAN CIRCUIT PROBLEM is not a parametrized problem. Thus, we are able to provide an FPT algorithm for the former problem, but not the latter.

Instead of tree decomposition, we will use sphere-cut branch decomposition as the basis of our dynamic program. As mentioned in Chapter 3, sc-branch decomposition offers several advantages over tree-decomposition. Namely, an optimal sc-branch decomposition can be computed in polynomial time, and the components in the sc-branch decomposition can be viewed as geometric components in the embedded graphs.

5.3.1 Branch decompositions and sc-branch decompositions

We start with a brief review of sc-branch decomposition and the associated notation. A Σ-graph refers to a graph $G = (V, E)$ embedded without edge crossings

on the unit sphere Σ. Define a *noose* to be a simple closed curve O in Σ that intersects G at vertices only and intersects every face at most once. A sphere-cut (sc) branch decomposition is a branch decomposition such that for every edge $e \in E(T)$, there is a noose O_e such that $\text{mid}(e) = O_e \cap V(G)$ and $G_j \in \Delta_j \cup O_e$, $1 \leqslant j \leqslant 2$.

Lemma 5.1.1 showed that graphs having k-milling walks must have branchwidth upper-bounded by $4k+9$. By Theorem 3.9.1, we can efficiently compute a sphere-cut branch decomposition with width at most $4k + 9$ for an arbitrary k-millable graph.

Recall from Chapter 3 that the dynamic programming formulation examples used *rooted* tree decompositions. By rooting the trees, we can define a bottom-up sequence of sub-problems to be solved by dynamic programming. Our dynamic program uses *rooted* sc-branch decompositions.

Here, we define rooted sc-branch decomposition and explain the associated notation. First, root T (of the sc-branch decomposition) at an arbitrary internal node r. See Figure 5–2. Consider any edge $e \in E(T)$ of the rooted sc-branch decomposition. Removing e separates T into two sub-trees T_1, T_2. By convention, define T_1 to be the subtree not containing the root r. We use G_1 and G_2 to refer to the subgraphs induced by T_1 and T_2, respectively. Furthermore, let O_e be the noose corresponding to e. This noose partitions the sphere into two open discs Δ_1, Δ_2. The *closed interior region of* O, denoted $\Delta_1^c = \Delta_1 \cup O$, is the closed region on which G_1 is drawn. Vertices drawn on O are called *boundary* vertices, while vertices drawn in Δ_1 are called *interior* vertices of O.

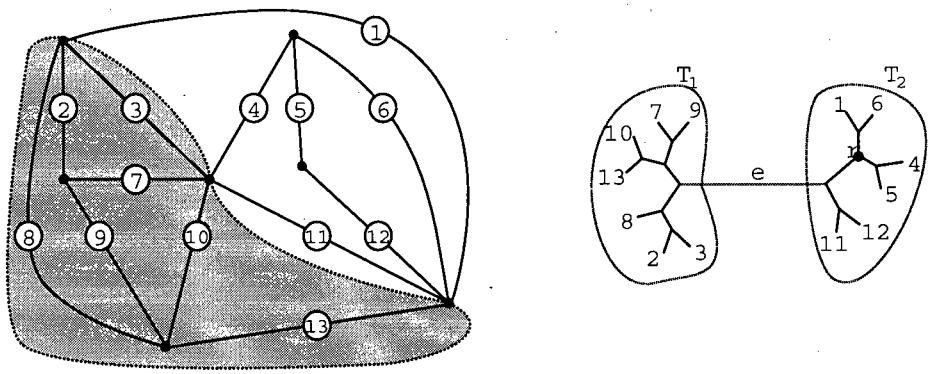

Figure 5–2: (right) A sc-branch decomposition (T, μ) rooted at r. T_1 is the component of $T - e$ that does not contain r. (left) The noose corresponding to e is represented by the dotted line. The shaded region is Δ_1 and the component drawn in Δ_1^c is G_1.

5.3.2 Solving subproblems corresponding to each node in T

Consider an optimum milling walk $W = \{v_1, e_1, v_2, e_2, \ldots, e_{s-1}, v_s\}$ for a grid-graph $G = (V, E)$. Let O be any noose. We say W *enters* Δ_1^c using edge e_j if $v_{j-1} \in \Delta_2$, $v_j \in O$, and $v_{j+1} \in \Delta_1^c$. We say W *leaves* O using edge e_{j-1} if $v_{j-1} \in \Delta_1^c$, $v_j \in O$, and $v_{j+1} \in \Delta_2$.

Let $e_1, \ldots, e_l$ and $e'_1, \ldots, e'_l$ be the sequences of edges the optimum walk W uses to enter and leave Δ_1^c, respectively. Within Δ_1^c, the optimum walk induces a path between between e_i and e'_i for $i = 1, \ldots, l$. The union of all such paths covers the interior vertices of Δ_1. However, this union does not necessarily cover all boundary vertices. Specifically, the endpoints of each path are boundary vertices (i.e., the endpoints are trivially covered), but the interiors of these paths may also cover other boundary vertices. The dynamic program looks for the optimum walk by enumerating possible combinations of paths for each noose.

We now define the subproblems corresponding to each nodes of the rooted sc-branch decomposition. For any $v \in V(T) - r$, let O_v be the noose corresponding to the edge between v and its parent. Consider an arbitrary subset $C \subseteq O_v \cap V(G)$ of boundary vertices and let $D = ((u,v) \mid (u,v) \in E(G_1)$ and u or v is inside $O_v)$ be the set of edges with at least one endpoint on O_v. Let $Q = \{(v_i, e_i, e'_i, v'_i)\}$ be an arbitrary multiset[4] of $C \times D \times C \times D$. We wish to compute $S_v(C,Q)$, which is a set $\mathcal{P} = \{P_1, \ldots, P_{|Q|}\}$ of paths satisfying the following conditions:

1. For $1 \leqslant i \leqslant |Q|$, P_i is a path with endpoints v_i and v'_i. The first (last) edge of P_i is e_i (e'_i).
2. For $1 \leqslant i \leqslant |Q|$, P_i is contained in G_1.
3. The union of all paths in $\mathcal{P}$ must cover all boundary vertices given by C along with the all interior vertices of O_v. The union of all paths may also cover boundary vertices not in C.
4. The total number of turns of all paths in $\mathcal{P}$ is minimized.

We say v is solved if $S_v(C,Q)$ has been computed for all possible combinations of C and Q.

The dynamic program proceeds in a bottom up manner, going from the leaves to the root r of T. Next, we show how a subproblem for an internal node can be solved using solutions of its two children (recall that internal nodes of a branch decomposition always have degree 3). the Let $v \in V(T) - r$ be any internal node and let x, y be the two children of v. Suppose x, y are already solved. See Figure 5–3 as a

[4] We will impose restrictions on Q in the next section.

guide. Note that the subgraph drawn in the closed interior region of O_v is the union of the subgraphs drawn in the closed interior regions of O_x and O_y. Consider $S_v(C, Q)$ for an arbitrary choice of C and Q. The paths for $(p_i, e_i, e_i', p_i') \in Q$ corresponds to a sequence of path segments in the closed interior regions of O_x and O_y. Inside O_x, this set of path segments corresponds to some $S_x(C_x, Q_x)$ which has already been computed. The same is true of O_y. Thus, to compute $S_v(C, Q)$, we enumerate combinations of $S_x(C_x, Q_x)$ and $S_y(C_y, Q_y)$ that form the desired paths defined in Q. See Figure 5–4 for an example. To solve v, we perform this procedure for every choice of C, Q.

The procedure described above allows us to solve the subproblems for every node except the root r. The root is a special case because it has three children $\{v_1, v_2, v_3\}$, which we assume are solved. Now, we may compute the milling walk by enumerating all combinations of $S_{v_i}(C_i, Q_i)$, $1 \leqslant i \leqslant 3$, which form a milling walk in G.

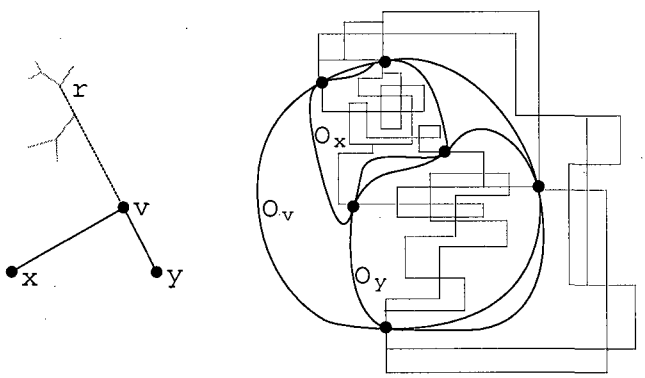

Figure 5–3: (left) edges a, b, c in the branch decomposition (right) nooses corresponding to edges a, b, c.

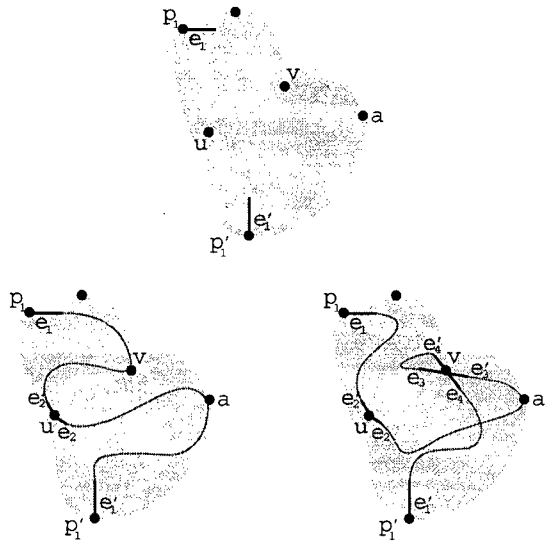

Figure 5–4: In this example, we consider two different possible solutions for $S_{O_v}(\{p_1, p'_1, a\}, \{(p_1, e_1, e'_1, p'_1)\})$. A possible solution is a $p_1 - p'_1$ path, with e_1, e'_1 as the first and last edges, coversing all interior vertices of $\Delta^c_{a_1}$ in addition to the boundary vertices $\{p_1, p'_1, a\}$. The first possible solution corresponds to two path segments: the segment $p_1 - u$ and the segment $u - p'_1$. This corresponds to the solutions $S_{O_x}(\{p_1, u, v\}, \{(p_1, e_1, e'_2, u)\})$ and $S_{O_y}(\{a, u, p'_1\}, \{(u, e_2, e'_1, p'_1)\})$. The second possible solution uses four total path segments. The reader should verify that this corresponds to the solutions $S_{O_x}(\{p_1, u, v\}, \{(p_1, e_1, e'_2, u), (v, e_3, e'_4, v)\})$ and $S_{O_y}(\{p'_1, u, v, a\}, \{(u, e_2, e'_3, v), (v, e_4, e'_1, p'_1)\})$

Bounding the number of path combinations

The dynamic programming formulation given in the previous section assumes no restrictions on the path combinations - i.e., different choices of Q. In order to obtain an FPT result, we must bound the number of such choices by $f(k)$ for some function on the parameter. We can modify the combinations so that only multisets with at most $k + 1$ elements need to be considered. Observe that if P_1, P_2 are two arbitrary turn-tree path segments inside a *convex* noose O, then any path between an endpoint of P_1 and an endpoint of P_2 must use at least one turn. Therefore, if a

milling walk P induces a sequence of paths $P_1, \ldots, P_{k+2}$ inside a convex noose, then P must use at least $k+1$ turns. In the Appendix, we show that it is possible to construct an sc-branch decomposition with convex nooses for grid-graphs with unit edge lengths.

For non-convex sc-branch decompositions, we can modify the dynamic program so that only multisets Q with at most $k+1$ elements need to be considered. Now, Q is re-defined to be arbitrary multisets of $D \times D \times \{1, 2, 3\}$. This is used to distinguish between three types of path segments:

type-1 : $(e_i, e'_i, 1, 1)$ - the path P_i is NOT straight and required to stay inside O_e;

type-2 : $(e_i, e'_i, 1, 2)$ - the path P_i is straight and required to stay inside O_e; and

type-3 : $(e_i, e'_i, 1, 3)$ - the path P_i is straight and required to leave O_e at least once.

Note that a type-3 path segment can be decomposed into a set of type-2 path segments inside O_e and a set of type-2 path segments in O_e. As defined thus far, the decomposition of an optimum walk W into path segments relative to O_e is not unique. To make this decomposition unique, we assume that if P_i, P_{i+1} are path segments of type-2 or type-3 (in O_e), then W uses at least one turn outside O_e because otherwise, P_i, P_{i+1} can be merged into a longer type-3 path segment. Furthermore, this implies that the we only need to consider multisets Q with at most $k+2$ elements.

We see that although the number of subproblems is bounded by a function of k, the actual bound is impractical for actual implementation. Nonetheless, this dynamic program establishes an FPT result for the GRID-GRAPH MILLING PROBLEM and provides a basis for future research (see summary).

CHAPTER 6
Summary

Our goal was to use FPT techniques and algorithms on graphs to produce FPT results for geometry problems. In our literature survey of covering tour problems, we focused this approach in the context of covering tour problems. We saw that the algorithms being exported are usually approximation algorithms. However, FPT algorithms could be used instead of approximation algorithms. Thus, we were interested in FPT algorithms for these covering tour problems.

A major portion of this thesis was to present the concepts and techniques that we used to obtain our FPT results. These concepts and techniques include tree-width, tree decompositions, dynamic programming on tree decompositions, monadic second order logic of graphs, parameter-tree-width bounds, and bidimensionality.

We used two classic problems - VERTEX COVER and HAMILTONIAN CYCLE - as examples to demonstrate how these concepts and techniques are unified into a framework for producing FPT results. Finally, we used this framework as a guide to devise FPT algorithms for the GRID-GRAPH MILLING PROBLEM and we conclude that these FPT algorithms can be used to obtain FPT results for the ORTHOGONAL MILLING PROBLEM.

6.1 Improvements and future direction

While our results can be used to obtain an FPT result for the ORTHOGONAL MILLING PROBLEM - a geometry problem, there are several practical issues that we

have not addressed. First, our FPT algorithms are not efficient and are therefore of only theoretical interest. Recall that while Courcelle's machinery can be used to obtain linear time algorithms, the hidden constant (depending on the tree-width bound and complexity of the formula used) is prohibitively large. Thus, we do not believe our FPT result can be improved to the point of being practical using Courcelle's machinery.

However, we believe that the dynamic programming formulation can be improved, probably significantly so. Recall that the TSP problem can be solved in $O(2^k \log^k n)$ compared to $O(2^{k^k} n)$ time for the GRID-GRAPH MILLING PROBLEM . The main difficulty in our problem is that each vertex can be used more than once and thus many more path segment combinations need to be considered. It is sensible to believe that the number of such combinations can be reduced by using a different formulation or a different decomposition paradigm. Furthermore, dynamic programming can be used in conjunction with other FPT techniques such as kernelization to obtain better results.

We have not fully established the applicability of the GRID-GRAPH MILLING PROBLEM in context of computational geometry. The GRID-GRAPH MILLING PROBLEM can be used to model the ORTHOGONAL MILLING PROBLEM . However, this geometric milling problem is severely restricted as compared to other geometric milling problems more often considered in computational geometry literature. It remains to be seen how grid-graphs or their generalizations can be used to model more general milling geometric milling problems.

Nonetheless, these graph milling problems can be used to approximate some of the more general geometric milling problems. However, using an FPT algorithm in this capacity has dubious value because the exported algorithm for the geometry problem (which is assumed to be hard) neither runs in fully polynomial time (the running time is exponential in the parameter) nor produces optimal solutions (because the grid-graph only approximates the geometry). An optimist point of view is that FPT algorithms produce better quality solutions than approximation algorithms; and in turn, FPT algorithms used in this capacity produce better quality solutions to the original geometry problem. The trade-off between solution quality and running time is ultimately application dependent. Since there are efficient constant factor approximation algorithms for the GRID-GRAPH MILLING PROBLEM and the current FPT algorithms developed in this thesis are not practical, we conclude that, at the moment, the former approach is better for this problem. It remains open whether there is a practical FPT algorithm for this problem.

Appendix: Convex sc-branch decompositions

Here, we sketch how to obtain *convex* sc-branch decompositions for grid-graphs with *unit* edge lengths (as opposed to integer edge lengths). We believe it is a straightforward technicality to extend the ideas in this sketch into a formal proof. If a grid-graph G has only unit length edges, then any "row" or "column" of G is a separator, and thus can be a midset of a branch decomposition of G. We will see that it is simple to construct convex nooses for these midsets, and that the resulting sc-branch decomposition has $O(k)$ if the grid-graph has a k-milling walk. We will implicitly describe this sc-branch decomposition as a set of cuts (midsets).

Let G be a grid-graph with unit edge lengths and assume G has a k-milling tour. We call a noose O *convex* if any turn-free path in G crosses O at most twice. The set $C_i = \{v \mid \Gamma_X(v) = i\}$ is defined to be the set of vertices drawn on column i. Similarly, $R_j = \{v \mid \Gamma_Y(v) = j\}$ is defined to be the set vertices drawn on row j. As mentioned above, C_i is a separator for any $x_{\text{MIN}} < i < x_{\text{MAX}}$, where x_{MIN} (x_{MAX}) is the smallest (largest) value of x such that there is a vertex v with $\Gamma_X(v) = i$. Since we assume the grid-graph is connected and has unit edge lengths, there must be at least one vertex in each column between the minimum and maximum. A column C_i is called *long* if $|C_i| > k + 1$; otherwise, C_i is called *short*. The next lemma shows that the number of long columns is bounded.

Lemma 6.1.1 *Let G be a grid-graph with unit edge lengths. If G has a k-milling walk, then G has at most $k + 1$ long columns.*

Proof: We will show that if there are more than $k+1$ long columns, then we can find $k+2$ pair-wise rook-independent vertices in G. This implies G does not have a k-milling walk.

Suppose $C_{i_1}, \ldots, C_{i_{k+2}}$ are long columns. Let v_{i_1} be any vertex of C_{i_1}. Since $|C_{i_2}| > k+1$ and at most most vertex in C_{i_2} lies on the same row as v_{i_1}, we conclude that there is a vertex $v_2 \in C_{i_2}$ that is rook-independent with v_1. This argument can be applied inductively to obtain a set of $k+2$ pair-wise rook-independent vertices as promised. □

We define a *cut along* C_i to be a partitioning of the edge set of the grid-graph into two parts:

- $E_{i_1} = \{(u,v) \mid \Gamma_X(u) \leqslant i, \Gamma_X(v) \leqslant i\}$, and
- $E_{i_2} = E(G) - E_{i_1}$.

It is easy to see that a convex noose can be drawn for any C_i cut. We will cut the grid-graph G along every short column. A consequence of the previous lemma is that if we cut G along every short column, then we obtain a set of components, with each component having "width" at most $k+1$.[1] . Furthermore, since we used only short columns for cuts and each component is involved in at most two cuts, it follows that the midset of each component has size at most $2k+2 \in O(k)$.

Now, let H be a component with width at most $k+1$. We can further cut H along each row of H. This produces a set of components with width at most $k+1$ and height 1. Since the width of H is bounded by $k+1$, we know that the midset

[1] Here, the "width" of a component is defined to be $x_{\text{MAX}} - x_{\text{MIN}}$

of the resulting components is at most $2k + 2$. It is easy to see that this sequence of cuts along columns and then rows can be used to obtain an sc-branch decomposition with bounded width.

REFERENCES

[1] J. Alber, H. Fernau, and R. Neidermier, *Parameterized complexity: Exponential speed-up for planar graph problems*, Proceedings of the 28th International Colloquium on Automata, Languages and Programming (2001), 261–272.

[2] E. Arkin, M. Bender E. Demaine, S. Fekete, J. Mitchell, and S. Sethia, *Optimal covering tours with turn costs*, SIAM J. Comput. **35(3)** (2005), 531–566.

[3] E. Arkin, S. Fekete, and J. Mitchell, *Approximation algorithms for lawn mowing and milling*, Comput. Geom. Theory Appl. **17** (2000), 25–50.

[4] E. Arkin, M. Held, and C. Smith, *Optimization problems related to zigzag pocket machining*, Algorithmica **26** (2000), 198–236.

[5] S. Arnborg, J. Lagergren, and D. Seese, *Easy problems for tree-decomposable graphs.*, J. Algorithms **12** (1991), 308–340.

[6] L. W. Beineke and R. E. Pippert, *The number of labeled k-dimensional trees*, Journal of Combinatorial Theory **6** (1969), 200–205.

[7] H. L. Bodlaender, *Polynomial algorithms for graph isomorphism and chromatic index on partial k-trees*, Journal of Algorithms **11** (1990), 631–643.

[8] H. L. Bodlaender and D. M. Thilikos, *Constructive linear time algorithms for branchwidth*, Lecture Notes in Computer Science **1256** (1997), 627–637.

[9] ———, *Treewidth for graphs with small chordality*, Discrete Applied Mathematics **79** (1997), 45–61.

[10] Hans L. Bodlaender, *A linear time algorithm for finding tree-decompositions of small treewidth*, SIAM J. Comput. **25** (1996), 1305–1317.

[11] V. Bouchitte and I. Todinca, *Treewidth and minimum fill-in: grouping the minimal separators*, SIAM J. Comput. **31** (2001), 212–232.

[12] J. Canny and J. H. Reif, *New lower bound techniques for robot motion planning problems*, Proc. 28th Annu. IEEE Sympos. Found. Comput. Sci. (1987), 49–60.

[13] V. Chvatal, *Hamiltonian cycles*, The Traveling Salesman Problem (E. Lawler, J. Lenstra, A. Rinnooy Kan, and D. Shmoys, eds.), John Wiley & Sons, 1985, pp. 403–429.

[14] K. Clarkson, *Approximation algorithms for shortest path motion planning*, Proc. 19th Annu. ACM Sympos. Theory Comput. (1987), 56–65.

[15] B. Courcelle, *The monadic second-order logic of graphs iii: tree-decompositions, minor and complexity issues*, RAIRO Inform. Theor. Appl. **26** (1992), 257–286.

[16] E. D. Demaine and M. T. Hajiaghayi, *Fast algorithms for hard graph problems: Bidimensionality, minors, and local treewidth*, Lecture Notes in Computer Science **3383** (2005), 517–533.

[17] Frederic Dorn, Fedor V. Fomin, and Dimitrios M. Thilikos, *Fast subexponential algorithm for non-local problems on graphs of bounded genus*, Manuscript (2006) http://www.ii.uib.no/publikasjoner/texrap/pdf/2006-320.pdf.

[18] ———, *Fast subexponential algorithm for non-local problems on graphs of bounded genus*, Proceedings of the 10th Scandinavian Workshop on Algorithm Theory, 2006, pp. 172–183.

[19] R. G. Downey and M. R. Fellows, *Parameterized complexity*, Springer-Verlag, 1999.

[20] H. L. Bodlaender F. Dorn, E. Penninkx and F. V. Fomin, *Efficient exact algorithms on planar graphs: Exploiting sphere cut decompositions*, Tech. report, Department of Information and Comput. Sciences, Utrecht University, 2005.

[21] R. Fagin, *Generalized first-order spectra and polynomial-time recognizable sets*, Proc. SIAM-AMS Sympos. Appl. Math. **7** (1974), 43–73.

[22] M. Held, *On the computational geometry of pocket machining*, Lecture Notes in Computer Science **500** (1991), 17–53.

[23] J. Hershberger and S. Suri, *An optimal algorithm for euclidean shortest paths in the plane*, SIAM J. Comput. **28(6)** (1999), 2215–2256.

[24] K. Iwano, P. Raghavan, and H. Tamaki, *The traveling cameraman problem, with applications to automatic optical inspection*, Proceedings of the 5th Annual International Symposium on Algorithms and Computation (1994), 29–37.

[25] T. Kloks, *Treewidth of circle graphs*, Algorithms and Computation (1993), 108–117.

[26] ———, *Treewidth. computations and approximations*, Lecture Notes in Computer Science **842** (1994), 173 – 184.

[27] K. Reinhardt, *The complexity of translating logic to finite automata*, Lecture Notes in Computer Science **2500** (2002), 249–256.

[28] N. Robertson and P. D. Seymour, *Graph minors, i. excluding a forest*, Journal of Combinatorial Theory Series B **35** (1983), 39–61.

[29] ———, *Graph minors, x. obstructions to tree-decomposition*, Journal of Combinatorial Theory Series B **52** (1991), 153–190.

[30] ———, *Graph minors, xx. wagner's conjecture*, Journal of Combinatorial Theory Series B (1992), 325–357.

[31] N. Robertson, P. D. Seymour, and R. Thomas, *Quickly excluding a planar graph*, Journal of Combinatorial Theory Series B (1994), 323–348.

[32] H. Rohrig, *Tree decomposition: a feasibility study. master's thesis*, Max-Planck-Institut fur Informatik, Saabrucken, Germany (1998).

[33] D. Seese, *Interpretability and tree automata: a simple way to solve algorithmic problems on graphs closely related to trees.*, Stud. Comput. Sci. Artificial Intelligence. **10** (1992), 83–114.

[34] P. D. Seymour and R. Thomas, *Call routing and the ratcatcher*, Combinatorica **14** (1994), 217–241.

[35] J. Vlades, R. E. Tarjan, and E. L. Lawler, *The recognition of series-parallel graphs*, SIAM J. Comput. **11** (1982), 298–313.

[36] S. Wismath, *Characterizing bar line-of-sight graphs*, Proceedings of the Symposium on Computational Geometry (1985), 147–152.

[37] X. Zhou, S. Nakano, and T. Nishizeki, *Edge coloring partial k-trees*, Journal of Algorithms **21** (1996), 598–617.