Minor-Embedding Planar Graphs in Grid Graphs

by

Ehsan Tavakoli

B.Sc., Computer Engineering - Software
Ferdowsi University of Mashhad, Iran, 2013

Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

in the
School of Computing Science
Faculty of Applied Sciences

Approval

Name:	**Ehsan Tavakoli**
Degree:	**Master of Science (Computing Science)**
Title:	***Minor-Embedding Planar Graphs in Grid Graphs***
Examining Committee:	**Chair:** Eugenia Ternovska Associate Professor

David Mitchell
Senior Supervisor
Associate Professor

Pavol Hell
Supervisor
Professor

Ramesh Krishnamurti
External Examiner
Professor

Date Defended: May 17, 2016

Abstract

A recent development in solving challenging combinatorial optimization problems is the introduction of hardware that performs optimization by quantum annealing. Exploiting this hardware to solve a problem requires first solving a graph minor-embedding problem, which is also apparently intractable. Motivated by this application, we consider the special case of finding a minor-embedding of a planar graph in a grid graph. We introduce two algorithmic approaches to the problem, based on planarity-testing techniques. For one approach, we provide an implementation and an experimental evaluation demonstrating its potential.

Keywords: Planar embedding, Minor-embedding, Adiabatic quantum annealing, Planar graphs, Grid graphs

Dedication

This thesis is dedicated to my ever faithful parents to whom I owe everything and to my amazingly supportive wife without whom this achievement would not have been possible.

Acknowledgements

My deepest gratitude goes to my senior supervisor, Dr. David Mitchell, who was the most resourceful, supportive, and patient supervisor I could ever ask for. I want to thank my supervisor, Dr. Pavol Hell, and my thesis examiner, Dr. Ramesh Krishnamurti, for their helpful comments and remarks on my thesis. I would also like to thank Dr. Eugenia Ternovska for serving as my thesis examining committee chair.
I want to express my sincere appreciation to my family and friends who have always been there for me. My special thanks go to my brother Ahmad Tavakoli, who has always been a source of inspiration for me. I also want to thank my uncle, Hamid Akbari, and his lovely family, for their emotional support and encouragement throughout my Master's studies.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Computer scientists have been trying for decades to find efficient and practical approaches for dealing with NP-hard problems. Over the past few years, D-Wave Systems, Inc. has been designing hardware, which employ quantum computing techniques to solve intractable problems. *D-Wave Two* is one of such device which works based on a mechanism referred to as *adiabatic quantum annealing.* Due to hardware design limitations, the device can only be used for solving problems in a special format. Performance of D-Wave Two was compared with several respectable optimization systems, including CPLEX, on a variety of benchmarks. The solutions found by D-Wave Two were at least as good as those found by its competitors, and it was significantly faster than other solvers [48]. However, there are challenges with regard to solving an NP-hard problem, even in the required format, using D-Wave Two. The prerequisite of having to find a solution to another NP-hard problem is one of the greatest of those challenges.

A graph describes necessary architectural attributes of the quantum device, and another graph describes the structure of problem. Before the quantum annealing process can be applied to solve the instance, a proper solution must be found for the NP-hard *graph minor-embedding* problem, which accepts the two graphs as inputs. Graph minor-embedding was virtually not studied, specifically, before the D-Wave application was found for it. However, the concepts of *minors, planar embedding*, and *graph drawing* are very well studied, in the graph theory literature, and can be argued to be related to special cases of the minor-embedding problem.

This chapter presents the motivation of this research, with a brief introduction to the concept of quantum computing in the D-Wave Two device and also the definition of graph minors and the minor-embedding problem, which are crucial to the application of the device. Next, related work on the topics of minor-embedding and graph drawing are reviewed. At the end of this chapter, contributions of this research to the problem are presented, followed by the organization of this thesis, in the next chapters.

1.1 Motivation

1.1.1 Graph Minors

Throughout years, graph minors have attracted much attention, due to the vast range of applications implied by their role in fixed-parameter tractability [5]. Throughout this thesis, the term "graph" refers to a simple, undirected, and finite graph, unless stated otherwise.

Definition 1.1. *Given two graphs G and H, we say H is a minor of G, if by deleting vertices, contracting edges or removing edges from G, we can create a graph isomorphic to H [5].*

The operation of removing an edge from the graph is trivial. Removing a vertex consists of removing the vertex and all edges incident to that vertex, from the sets of vertices and edges of the graph, respectively. Edge contraction takes place by removing the two vertices, at the ends of an edge and adding a new vertex to the graph, which is adjacent to the union of sets of neighbours of the two deleted vertices. Let $H = (V, E)$ and $H' = (V', E')$ be two graphs, which are to be tested for being isomorphic. It is said H is isomorphic to H', or $H \simeq H'$, when there exists a 1-1 function $f : V \rightarrow V'$, such that for each $u, v \in V, \quad (u, v) \in E$, if and only if $(f(u), f(v)) \in E'$ [25].

1.1.2 Application

The D-Wave Two device employs a model, known as Adiabatic Quantum Computing (AQC), which works by evolving a quantum system "slowly enough" from an initial state to the final state [16]. This device is used to solve a specific type of quadratic optimization problem, a form of *Ising model*, in which variables can only obtain values $\{-1, 1\}$. This optimization problem is NP-hard and can be formulated as follows [12]:

$$\text{Minimize} \quad f(z) = w^T z + z^T J z$$

where z is a size n vector of variables, w is a size n vector of weights, and J is a $n \times n$ matrix. Each variable can be mapped to a *qubit*, in the device hardware, and each non-zero element in J can represent a *coupling* between the cubits. The minimized $f(z)$ can be obtained from a specific energy function of the quantum system, called a *Hamiltonian*, in the final stage of the aforementioned system evolution process, or quantum annealing. More details about the quantum mechanics aspects of the process of quantum annealing can be found in [32, 52].

The device hardware can be presented by a graph $G = (V_G, E_G)$, where V_G and E_G are the sets of vertices and edges of G, respectively. Each $v \in V_G$ represents a *physical qubit* in the device, and for each $u, v \in V_G$, if $(u, v) \in E_G$, the Ising model of the problem is allowed to have a non-zero value for J_{uv}. The Ising model of the problem can also be represented by a graph $H = (V_H, E_H)$, where V_H represents the *logical qubits*, and corresponds to z, and $(u, v) \in E_H$ if $J_{uv} \neq 0$. Due to hardware design limitations, the hardware graph G cannot be a complete graph, which limits the problem instances that can be solved by the hardware. However, using a technique, this limitation can become less restrictive. The technique makes it possible to represent a logical qubit with more than one physical qubit, as long as the vertices corresponding to the set of selected physical qubits form a connected subgraph of G. For $x \in V_H$, let $\phi(x)$ be a subset of V_G, which maps to the set of physical cubits assigned to represent the logical qubit of x, in the Ising model of the problem. For each two neighbouring vertices $u, v \in \phi(x)$, if $J_{uv} = -\infty$ in the Ising model of the hardware, the values of u and v are forced to be the same in the variable vector z which minimizes the value of $f(z)$. Also, for all $u, v \in V_H$ for which $J_{uv} \neq 0$ in the problem Ising model, it is possible to represent these interactions in the hardware Ising model, between $\phi(u)$ and $\phi(v)$.

The explained limitations and possibilities of the D-Wave Two device leads to the conclusion that it is possible to map the defined problem Ising model onto the device, if and only if H is a minor of G. We need to find a set of subgraphs of G, which represent the vertices of H, as outlined above. In short, finding such subgraphs of G is called minor-embedding H in G.

In the D-Wave's application, H is part of the input and varies for different problems. The fastest exact algorithm, given arbitrary graphs G, H, for finding a minor-embedding of H in G runs in exponential time in the branch-width of G. It is also known about the D-Wave quantum annealer that the design policies of the hardware graph encourage it to have a large tree-width, and also the graphs, corresponding to the problem Ising models, are not very small graphs, in practice [12]. It is easy to conclude, from the presented facts, that exact algorithms cannot be used for the D-Wave's minor-embedding problem, in practice.

Figure 1.1 shows the hardware graph of the D-Wave Two device, called the *Chimera* graph. Each node represents a qubit, and those nodes with darker colour correspond to inactive qubits.

Definition 1.2. *A k by k Chimera graph, with no inactive qubits, has $8k^2$ vertices in the format of a $k \times k$ grid structure, in which all vertices of the grid are replaced by $K_{4,4}$ bipartite graphs. Generally, a complete bipartite graph $K_{4,4}$ has two disjoint sets U and V of 4 vertices, and vertices of each set are only connected to all vertices of the other one. For each $K_{4,4}$ of the Chimera graph,*

- *all vertices of V are also connected to the respective vertices in the V sets of the left and the right neighbouring $K_{4,4}$ graphs (if they exist) in the grid structure, and*
- *vertices of U are adjacent to the respective vertices in the U sets of the neighbouring $K_{4,4}$ graphs, above and below, if they exist in the grid [21].*

A closer look at the structure of the Chimera graph will reveal that most of the vertices have degree 6, and if no vertices correspond to inactive quibits, it contains 4 layers of grid graphs, with edges between respective vertices of neighbouring layers, as a minor. Inspired by this structure, the specific problem of minor-embedding planar graphs in limited size grid graphs is the main focus of this work, and we may refer to it simply as the minor-embedding problem here on after. Our perspective for future extension of this work is to try to break H (the *input graph*) into a limited number of planar graphs, minor-embed them in different layers of the Chimera and eventually embed the connecting edges between separated components of H. Strategies based on planarization techniques may also be worthy of investigation.
Although, G (the *target graph*) is set to a grid graph in this version of the problem, we still consider the possibility of inactive qubits and possibility of slight changes to the structure of the Chimera. For that purpose, the heuristic solutions should not be very dependent on precise structural properties of a grid graph. Since input graphs have been limited to the family of planar graphs, concepts which are specific to this graph family and can be related to the minor-embedding problem should be explored and possibly employed by the solution.

1.2 Related Work

The problem of minor-embedding a graph H in a graph G, for the D-Wave application, was first introduced by Choi [16]. They continued working on minor-embedding in adiabatic quantum computation, by investigating a possible hardware architecture, which admits efficient algorithms for finding a minor-embedding of any graph, with a bounded maximum degree. This problem remained open [17].
To test H for being a minor of G, when H and G are arbitrary and both given as input, is an NP-complete problem. Robertson and Seymour, as a part of their pioneering work on graph minors, proposed an algorithm that can verify if a fixed graph H is a minor of an arbitrary graph G in polynomial time [54]. However, in case of the D-Wave's application, H is part of the input and varies for different problems. Given b is the branch-width of G, the fastest exact algorithm for the problem of finding a minor of G that is isomorphic to H runs in $O(2^{(2b+1)\log b}|H|^{2b}2^{2|H|^2}|H|)$ time [1]. For these reasons, application of exact algorithms to the D-Wave's minor-embedding problem is practically impossible.
There has not been much interest in heuristic algorithms for the minor-embedding problem in the literature, most probably because of the limited applicability before the D-Wave's application. To our knowledge, the only existing practical solution to this problem is a heuristic, which mainly uses pathfinding strategies to find a minor-embedding of H in the hardware graph [12]. Most choices in this heuristic are made either randomly or using a greedy approach, and the structure of H is not a parameter. More recently, a determinis-

tic algorithm has been proposed for the more restricted problem of minor-embedding the cartesian product of two complete graphs in a Chimera graph without inactive quibits [64].
A planar graph can be described in different ways. A geometric definition of planarity states that a graph is planar if it can be drawn on a Euclidean plane with no two edges crossing each other [45]. The combinatorial definition, more helpful with testing for planarity, shows that a graph is planar if and only if $K_{3,3}$ and K_5 are not contained in it as minors [5]. Planarity testing methods usually either output a *planar embedding*, that is, a clockwise ordering of incident edges to all vertices of the input graph, or one of its *forbidden minor*($K_{3,3}$ or K_5) subgraphs. The first linear time planarity testing algorithm was introduced by Hopcraft and Tarjan [42], and was later known as the *path addition* method. Another algorithm, sometimes called the *vertex addition* method, was introduced with quadratic time [46], and was improved later to a linear time algorithm [31, 7]. Both of these methods are considered to be fairly complex. Later attempts introduced linear time algorithms using simpler approaches, such as traversing a DFS tree in reverse order and embedding back-edges while preserving planarity. This approach was first introduced in [35] and does not use complex structures, such as the *PQ-trees* employed in [7]. Boyer and Myrvold based their linear time algorithm on this approach and presented it in details [10]. Since the algorithm works by adding back-edges, working its way up from the bottom of the DFS tree, it can be referred to as *edge addition* method.
Graphs admit visual representations in the plane, which can be generated by *graph drawing algorithms*. The generated drawings have multiple attributes, and algorithms prioritize them for optimization based on the application. While number of bends, embedding area, uniform vertex and edge degree distribution, drawing symmetry, and so forth are included in the desired attributes, minimizing the edge crossings is arguably the most important one [51]. The reason is the vertex adjacency confusion caused by edge crossings. For that reason, drawing methods usually start with planarity testing of the input. Drawing algorithms exist for planar and non-planar graphs.
Since a crossing-free drawing does not exist for non-planar graphs, different approaches exist in the literature for drawing these graphs. *Planarization* of a non-planar graph is one appraoch, which involves replacing crossing points with virtual vertices, or removing a number of edges from the graph. To find the smallest set of edges which, upon their removal, gives a planar graph, is NP-hard [53]. Both exact methods, such as a branch-and-cut algorithm [43], and heuristic solutions [57, 38] have been introduced for this problem. Following approaches mostly work on sparse graphs. Minimizing the number of crossings, although an NP-hard problem [22], has been pursued for sparse and small graphs [4, 30]. Finding a *k-planar* drawing, in which no edges are crossed more than k times, is the goal of some algorithms [11, 23]. Another approach, surveyed in [24], tries to make the crossings easier to comprehend by widening the angle of crossing edges.
Another interesting approach is to focus on finding a large planar subgraph of the input

and preserving its planarity in the drawing [2]. *Confluent drawings* are another means to conveying the adjacency information, while avoiding crossing of edges [22]. These drawings bundle up crossing edges in *tracks* and replace crossings with parallel paths.

Methods for drawing planar graphs usually depend on an ordering of vertices, or edges, or both. Several algorithms rely on a *canonical ordering* of vertices and a certain degree of connectivity of the graph [45]. This concept was first introduced, by De Fraysseix *et al.* [34], improved by Kant [44], and was demonstrated to be sufficient for generating a *straight-line drawing*, a graph drawing in which all edges are represented by straight line segments. Earlier, Fáry [33] had discovered that a straight-line drawing exists for each planar graph, and algorithms had been introduced to compute such drawing, using arithmetic techniques [61, 15, 14]. The proposed canonical ordering allowed iterating over vertices and adding them to the drawing in that order. In each iteration, a subset of edges are embedded to generate a planar drawing of the graph, induced by the set of vertices which have been iterated over. Chrobak and Payne introduced a simpler straight-line drawing based on the canonical ordering, which improved the original $O(n \log n)$ time algorithm to a linear time method with the same required grid area of $(2n-4) \times (n-2)$ [18]. Schnyder used the same ordering to introduce a straight-line drawing in a $(n-2) \times (n-2)$ grid, which took advantage of a barycentric representation of the graph [55]. Authors of [18], however, claimed their algorithm remained popular, despite worse embedding area requirements, due to its simpler implementation and more aesthetically pleasing output. Although the canonical ordering is mostly employed by straight-line drawing algorithms, it is also the basis of some *orthogonal drawing* methods, an instance of which can be found in [44].

Planar orthogonal drawings are another family of graph drawings for planar graphs, in which only horizontal and vertical line segments are permitted to represent the edges. The angles between all line segments are $\frac{n\pi}{2}, n \in \mathbb{Z}$, which are aesthetic and also suitable for VLSI design applications. An orthogonal drawing is also a grid drawing, if we map the coordinates of its vertices and bend points to nodes of an arbitrary 2D-grid. A graph with maximum degree of more than four cannot admit an orthogonal planar drawing. However, other polyline algorithms with fewer restrictions on the edge angles exist in the literature [39, 40, 13, 29], for graphs with maximum degrees of more than four. These algorithms are mostly based on Kant's work [44] and use the same concept of adding vertices, one-by-one, based on the canonical ordering. Older orthogonal drawing algorithms focused on minimizing the number of bends at the expense of embedding area and run time. However, taking advantage of a *visibility representation* in the 2D plane, in which both vertices and edges of the graph are represented with orthogonal line segments, newer algorithms achieve the same goal in linear time and smaller embedding area [28]. Algorithms using network flows can minimize the number of bends in a bounded area, although they run in near-quadratic time [58, 36, 20].

A visibility representation is defined generally as the result of representing vertices with hyper-rectangles in $\mathbb{R}^n$ Euclidean space, in a way that the edges are represented orthogonal to the hyper-rectangles [49]. In a 2D-plane, vertices and edges are represented by horizontal and vertical lines, respectively. The concept has been well-studied, due to its many applications in VLSI design [27, 60, 63, 47]. It was proved in [59, 63] that a graph admits a visibility representation in the $\mathbb{R}^2$ plane if and only if it is planar, in which case it can be generated in linear time. Tamassia and Tollis classified these representations into three classes, based on their restrictions and combinatorial attributes [59]. Cobos *et al.* generalized the work in [59, 63] to bound the dimensions of the Euclidean plane which can contain the visibility representation for several graph families [19]. Boyer also presented an extention to his work [10], in [9], introducing an efficient method of computing visibility representations, without using complicated structures.

1.3 Contribution

In this thesis, we introduce two algorithms for the problem of minor-embedding a planar graph $H = (V_H, E_H)$ in a grid graph $G = (V_G, E_G)$ of limited dimensions. To facilitate future modification of the algorithms to compensate for inactive qubits and possible changes of the target graph structure, our solutions employ pathfinding algorithms to solve the problem, using only vertices of the target graph which represent the active and available qubits of the hardware. Both algorithms take advantage of the planarity of the input graph H, to minor-embed it in a grid graph G.

The first presented method uses a specific planar representation of H to minor-embed it in G. The main component of the algorithm is a novel technique introduced to help reduce the minor-embedding size, which can be used in some other minor-embedding algorithms as well. Being provided with the required planar representation of H, the minor-embedding algorithm can minor-embed H in G, and the required dimensions of G depend only on $|V_H|$. These attributes make this algorithm suitable for minor-embedding dense graphs in G. However, the algorithm requires as input the aforementioned planar representation of H to minor-embed it in G. We propose two modifications for the first algorithm so that it can minor-embed an arbitrary planar graph in a bounded grid graph.

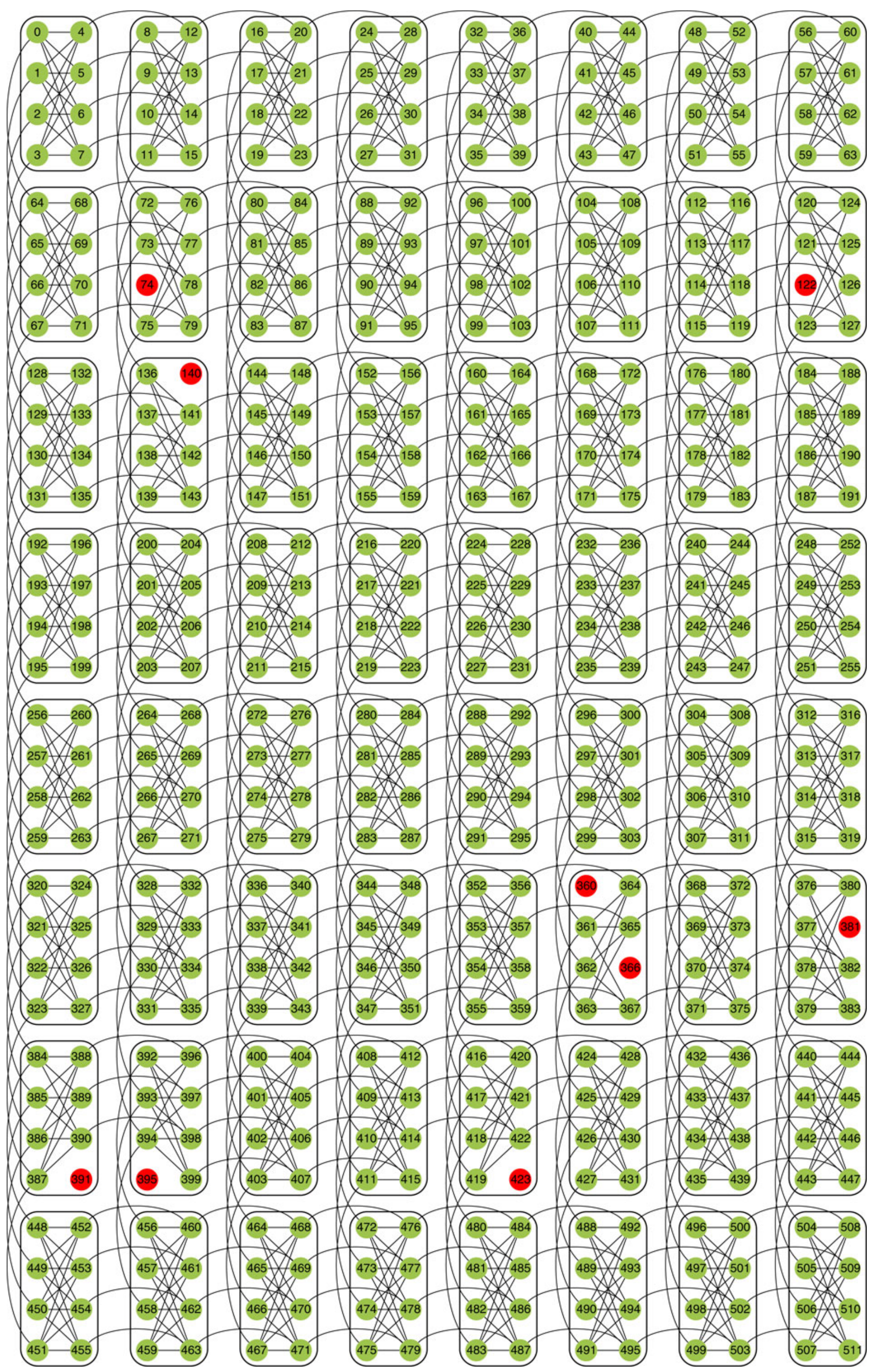
Our second algorithm generates a solution based on an obtained visibility representation of H. This method takes advantage of the technique, used in the previous algorithm, and another novel optimization technique which demonstrates good performance in improving desired qualities of the minor-embedding. The algorithm is shown to be complete and works in two phases. In the first phase, a minor-embedding of H is guaranteed to be found in G, with bounded dimensions, and the second phase tries to optimize the found solution.

We give an experimental evaluation of the second algorithm, including a comparison with the algorithm in [12]. The results of the experiments have been analyzed and presented, followed by suggestions for future work. Our results demonstrate the consistent success of our algorithm in minor-embedding all instances of different sizes from various graph families, whereas the competing algorithm [12] is only able to embed the smaller ones, or none at all, per each family.

1.4 Organization of the Thesis

Chapter 2 presents necessary preliminaries and background concepts. In Chapter 3, a specific planar representation of the input graph is suggested as the basis of our first algorithm, and a rather generic, novel technique is introduced to improve the results. This technique is also employed in the proposed algorithm, in Chapter 4, which takes advantage of a visibility representation of the graph, to guarantee its minor-embedding, in a grid graph. Experiments are conducted on this algorithm and another minor-embedding heuristic, the results of which are presented in Chapter 5. Chapter 6 concludes the thesis and proposes future courses of action to extend this work.

Figure 1.1: The 8x8 Chimera graph, with 512 vertices, and maximum degree 6 [50]



Chapter 2

Background and Preliminaries

Background and preliminaries, essential to better comprehend the next chapters, are presented in the following sections.

2.1 Planarity

Definition 2.1. *Let $G = (V_G, E_G)$ be a graph and $D : V_G \cup E_G \to 2^{\mathbb{R}^2}$ be a mapping from vertices and edges of G to points and simple curves in the plane, respectively. D is a planar representation of G, if:*

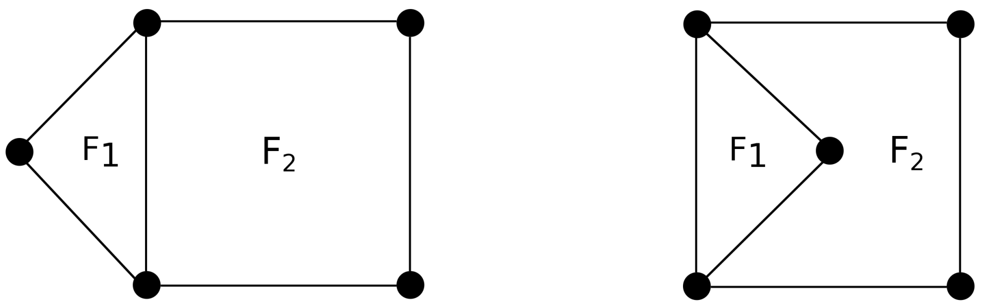
- *For each $v \in V_G$, $D(v) \in \mathbb{R}^2$ is a point, and for all pairs $u, v \in V_G$, $D(v) \neq D(u)$.*
- *For each $(u, v) \in E_G$, $D(u, v)$ is a simple curve satisfying the following conditions:*
  - *$D(u), D(v) \in D(u, v)$, and the end points of $D(u, v)$ are $D(u)$ and $D(v)$.*
  - *For all $u, v, w \in V_G$, if $D(w) \in D(u, v)$ then $w = u$ or $w = v$.*
  - *For all $e, (u, v) \in E_G$, if $e \neq (u, v)$ then $D(u, v) \cap D(e) \subset \{D(u), D(v)\}$.*

Let $D_G = \bigcup_{x \in (V_G \cup E_G)} D(x)$ be the set of all points in the plane representing vertices and edges of G. Each point in the plane either belongs to D_G, belongs to an area bounded by D_G, or neither. The areas in $\mathbb{R}^2$ bounded by D_G are called the *faces* (or *inner faces*) of D, and the set of points in $\mathbb{R}^2 - D_G$ not on the inner faces of D, constitute the *outer face* of D.

Definition 2.2. *A graph is called a planar graph if and only if it has a planar representation.*

Since $\mathbb{R}^2$ defines the set of points in a plane, a graph can have zero or an infinite number of planar representations [45]. It is possible to partition planar representations of G into a finite number of equivalence classes. Each equivalence class is referred to as a *planar embedding* of G, and shortly, we will explain the equivalence relation between two planar

Figure 2.1: Two planar representations of one graph, which do not realize the same planar embedding.



representations realizing the same planar embedding.
First, for each $v \in V_G$, we define I_v to be the set of all edges of G incident to v.

Definition 2.3. *Let D and D' be two planar representations of a graph $G = (V_G, E_G)$. D and D' realize the same planar embedding if for all $v \in V_G$ with $I_v = \{e_1, e_2, \ldots, e_m\}$, the same clockwise ordering of appearance in $\mathbb{R}^2$ applies to both sets $\{D(e_1), D(e_2), \ldots, D(e_m)\}$ and $\{D'(e_1), D'(e_2), \ldots, D'(e_m)\}$ [3].*

Figure 2.1 illustrates planar representations of two different embedding classes for a graph.

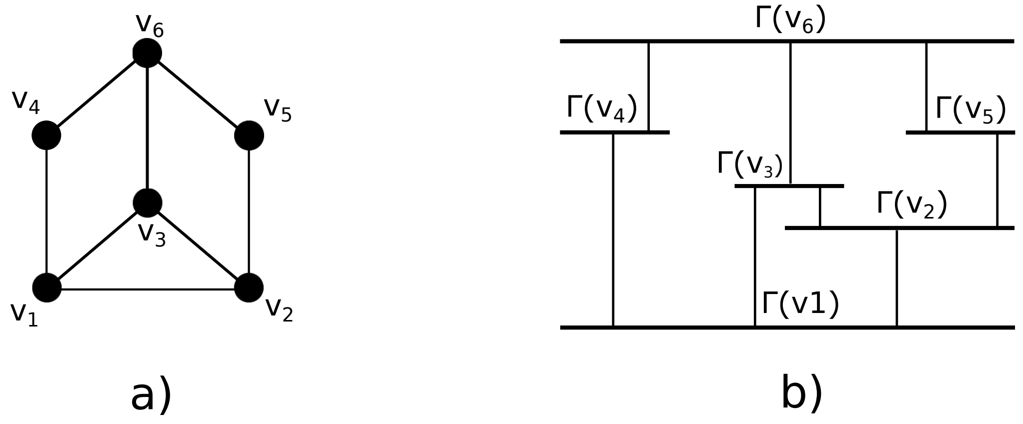
2.2 Visibility Representation

Definition 2.4. *Let $H = (V_H, E_H)$ be a planar graph and $\Gamma_H : V_H \cup E_H \to 2^{\mathbb{R}^2}$ be a function, mapping vertices and edges of H to non-overlapping line segments in the plane. Γ_H is a visibility representation of H if:*

- *For each $v \in V_H$, $\Gamma_H(v)$ is a horizontal line segment.*
- *For each $e \in E_H$, $\Gamma_H(e)$ is a vertical line segment.*
- *If $\Gamma_H((u, v)) \cap \Gamma_H(w) \neq \emptyset$, then $w = u$ or $w = v$.*

A visibility representation of a graph provides an ordering for its vertices and edges, which is often used in planar graph drawing algorithms. In Figure 2.2, a graph and a visibility representation of it are depicted.

Figure 2.2: Pictures a and b depict a planar graph G and a visibility representation of G, respectively.



2.3 2D Grid Graphs

A grid graph is a planar graph with specific structural attributes. In order to define grid graphs, we need to present the operation, used for generating them. The *Cartesian graph product* of two graphs $H_1 = (V_1, E_1)$ and $H_2 = (V_2, E_2)$, denoted by $H_1 \square H_2$, is the graph $G = (V_G, E_G)$ with $V_G = \{(u, v) : u \in V_1, v \in V_2\}$. For each pair $(x_1, y_1), (x_2, y_2) \in V_G$, $((x_1, y_1), (x_2, y_2)) \in E_G$ if and only if $(x_1 = x_2$ AND $(y_1, y_2) \in E_2)$ OR $((x_1, x_2) \in E_1$ AND $y_1 = y_2)$ [6].

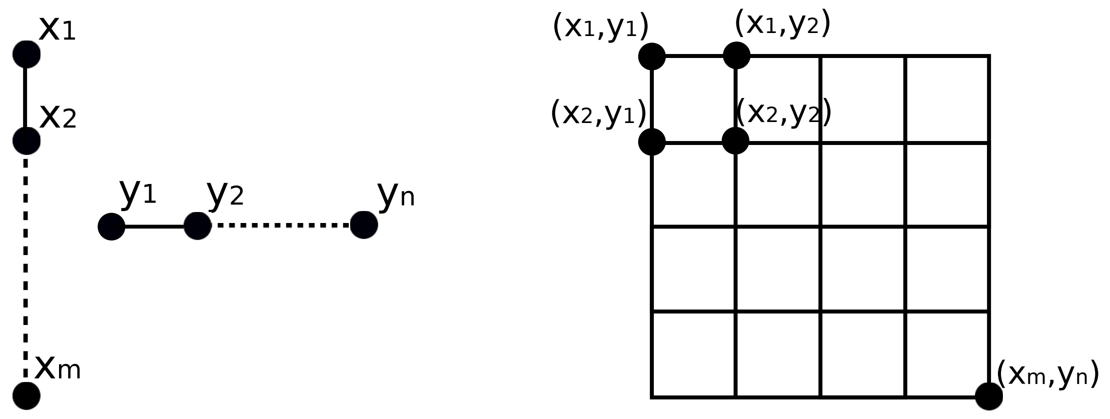
Definition 2.5. *A grid graph, or simply a grid, is the Cartesian graph product of two path graphs. The grid graph $G_{m,n} = P_m \square P_n$ is called an $m \times n$ grid, and has mn vertices and $(m-1)n + (n-1)m$ edges.*

An $m \times n$ grid has a unique planar representation with m horizontal (rows) and n vertical (columns) simple straight line segments, orthogonal to each other. The planar representation is called a *square grid*, if it tiles a rectangular area of the plane with identical squares [62].
Figure 2.3 shows two path graphs of sizes m and n and the grid, resulting from the Cartesian graph product of those paths.

2.4 A* Algorithm

Before presenting the A* algorithm, it is easier to remind the reader of the Dijkstra algorithm for finding a shortest path from a vertex (source) to another vertex (destination) of a graph. The algorithm starts by selecting the source as the *current vertex*, setting its *distance upper*

Figure 2.3: Grid graph $G_{m,n}$ is the Cartesian graph product of path graphs P_m and P_n



bound to itself to zero and its distance upper bound to other vertices to infinity. At each iteration, the distance upper bound from the source to each *unvisited* neighbour of the current node is updated, if its path to the source, going through the current vertex, is shorter than its currently known path. The current vertex is then marked as *visited*, and unless the current vertex is also the destination, the next unvisited vertex, with the smallest distance upper bound to the source, is selected as the current vertex, and the next iteration begins (the graph is assumed to be connected).

A* extends the Dijkstra algorithm by adding a new component to it, which helps explore the more "promising" branches of the search tree earlier than the rest. The prioritization is carried out using a *heuristic* h, which estimates the distance between each vertex and the destination. At each iteration of the A* algorithm, similar to the Dijkstra algorithm, one vertex must be chosen to be the current vertex of the next iteration. The unvisited vertex v minimizing $g(v) + h(v)$ will be selected, where $g(v)$ indicates the distance upper bound between the source and v.

The asymptotic time complexity of A* is exponential in the length l of the shortest path, in the worst case. The average number of possible choices at each node of a decision tree is called the branching factor (b) of that tree. The worst-case time complexity of A* is $O(b^l)$, which corresponds to a scenario in which, all possible choices are explored by the algorithm, in a way that the last possible choice leads to finding the shortest path between the two points. The heuristic is supposed to prune as many of those possible branches as possible. If the branching factor is finite, and edge costs are positive, A* is complete. Furthermore, if the heuristic does not overestimate the distances of nodes to the destination (is admissible), A* will always find the optimal path [56].

2.5 The D-Wave Algorithm for Minor-Embedding Problem

Cai *et. al.* [12] introduced a heuristic minor-embedding algorithm, designed for instances in which the target graph is a Chimera graph. This heuristic will be referred to as the D-Wave algorithm, hereinafter. Given graphs G and H as input, it employs a heuristic based on pathfinding algorithms to try minor-embedding H in G. One way of looking at minor-embedding is in terms of *vertex-models*. Given the input graph $H = (V_H, E_H)$ and the target graph $G = (V_G, E_G)$, for each vertex $x \in V_H$, $\phi(x) \subset V_G$ denotes the vertex-model of x. The set of vertex-models of V_H in V_G is defined as follows:

Definition 2.6. *Vertex-models of V_H are disjoint subsets of V_G, which satisfy the following conditions:*

- *For each $x \in V_H$, there must exist at least one connected subtree of G, containing exactly $\phi(x)$.*
- *For each edge $(x, y) \in E_H$, there must exist an edge $(u, v) \in E_G$, such that $u \in \phi(x)$ and $v \in \phi(y)$.*

There is a minor-embedding of H in G, if and only if there exists a set of vertex-models of V_H in V_G [12].

The algorithm finds vertex-models iteratively. Finding a minor-embedding of H in G is the first priority of the heuristic, however it is desirable if each vertex-model is as small as possible, so that more vertices are left in the target graph for embedding the remaining vertices of H. The algorithm runs in two phases. In the first phase, V_H is randomly ordered and iterated over. For each $v \in V_H$, if no neighbours of v have been embedded yet, a random vertex in G is assigned to be $\phi(v)$. Otherwise, the vertex in G, "closest" to the vertex-models of all embedded neighbours of v, is selected to be the *root* r_v of $\phi(v)$. The root is found using pathfinding algorithms, and the vertices in the shortest paths are also added to either $\phi(v)$ or the neighbouring vertex-models. The random ordering of vertices and random assignment of vertex-models to vertices without embedded neighbours determine the chance of finding vertex-models for all $v \in V_H$, in a way that the set of vertex-models constitute a minor-embedding of H in G. In particular, two or more vertex-models are *infeasible* and cannot be in the set of vertex-models if they are not disjoint. In the first phase, infeasible vertex-models are allowed, but penalized by added costs.

In the second phase, the algorithm iterates over the vertices multiple times. At each iteration, the vertex-model is cleared, and the vertex is embedded again in hopes of reducing the number of non-disjoint vertex-models and improving the quality of the solution. In this phase, regardless of the order of vertices, neighbours of each vertex are already embedded. Therefore, at every iteration, vertices which had been embedded randomly get a chance to be assigned to more "relevant" vertex-models, and the algorithm tries to decrease the

number of infeasible vertex-models.
Following are a few points crucial to understanding each phase of the algorithm:

Phase One - For each $x \in V_H$, the root r_x of $\phi(x)$ is found by minimizing a function f_{dist}. This function accepts a vertex $v_g \in V_G$ and the set of all embedded vertex-models, adjacent to x, as inputs and represents a notion of distance between v_g and all the given vertex-models. Let N_x be the set of vertex-models passed as input to f_{dist}. The vertex in the target graph, minimizing f_{dist}, will be the r_x. f_{dist} is defined as follows:

$$f_{dist}(v_g, N_x) = \max_{vm \in N_x} dist(v_g, vm)$$

where $dist(u, A)$ is the shortest path from $u \in V_G$ to any vertex in $A \subset V_G$.
One of the requirements of all pathfinding algorithms is a function to calculate the cost of each edge of the search graph, or a matrix of predefined costs for all edges. The edge costs for the minor-embedding problem are expected to help find the shortest path between the source and the destination and also heavily penalize *crossings*, which occur when two or more vertex-models have a non-empty intersection. Therefore, for each $v \in V_G$, a *vertex-weight* $W(v) = D^{n_v}$ is defined, where D is the diameter of the target graph, and n_v is the number of vertex-models containing v. Now, the edge cost for each $(u, v) \in E_G$ is equal to $W(v)$, which might be different from $W(u)$ and cost of $(v, u) \in E_G$.
The shortest path between r_x and each vertex-model $vm \in N_x$ may contain vertices of G, other than r_x or vertices in vm. For each $vm \in N_x$, a set containing these vertices is maintained, and the algorithm adds each vertex v_g in the union of these sets

- to $vm \in N_x$, if v_g appears only in the shortest path, between r_x and vm, or
- to $\phi(x)$, otherwise.

A naive way of finding the vertex $v \in V_G$ minimizing $f_{dist}(v, N_x)$ is to compute this value for all vertices in G, and find the r_x. Dijkstra's algorithm and A* are among the algorithms which can be utilized for this purpose. The authors of [12] also suggested an alternative to this approach, which expands shortest path trees (using Dijkstra or A* algorithms) from each $vm \in N_x$, and for each vertex in V_G, keeps records of being visited by each of the shortest path trees. Once a vertex is visited by all trees, the expansion of trees stops, and that vertex is selected as r_x. The visited paths, leading back from r_x to each $vm \in N_x$, will be the shortest paths from r_x to the corresponding vertex-model. If A* is used to expand the shortest path trees, this method is referred to as *multisource A** [12].

Phase Two - This phase of the algorithm consists of an uncertain number of rounds. In each round, all vertices of the input graph are embedded again in the same random order as the first phase. It is safe to assume that the input graph does not contain any isolated vertices, because in that case, we can simply separate those vertices from the graph, minor-embed the separated subgraph, and finally find random vertices of G with vertex-weights of 1 to represent the isolated vertices of the input graph. Based on that assumption, in all iterations of all rounds in the second phase, $N_x \neq \emptyset$, and so vertices which have been embedded randomly earlier will be assigned vertex-models in relevance to the vertex-models of all their neighbours. Therefore, not only the number of crossings are expected to decrease, but also the sum of cardinalities of all vertex-models will be reduced to some point, as a result of multiple rounds of re-embedding all vertices.

At the end of each round, the algorithm determines if any improvements have been made to the solution by measuring the following parameters of the embedding and comparing them with those of the best known solution. Checking parameters for reduction in values is conducted in the following order, and as soon as one measurement indicates improvement, the remaining comparisons are aborted:

1. The number of vertices in V_G that appear in two or more vertex-models (number of crossings)
2. The sum of the cardinalities of all vertex-models (also referred to as the minor-embedding size or the solution size)
3. The size of the vertex-model with the maximum cardinality

The algorithm will return a minor-embedding if the best found solution has zero crossings, and will return failure if a consecutive number of rounds end without any improvements.

As was stated earlier, despite the fact that the heuristic is not limited to a fixed target graph, such as a Chimera graph, the effectiveness of the heuristic depends on several attributes of both graphs. As the target graph gets more dense, the costs of pathfinding algorithms grow [56], and the sparseness of the Chimera graph makes it suitable for this heuristic. Furthermore, since the Chimera can be looked at as a number of connected isomorphic blocks of $K_{4,4}$ graphs, the random choices on the order and representation of the vertices of H have relatively small chances at affecting the success probability of the heuristic negatively. These attributes of a Chimera graph, as the target graph, boost the effectiveness of the algorithm, however the heuristic is reported to be successful only when the input graph is much smaller than the target graph [12].

Chapter 3

Minor-Embedding Based on a Planar Embedding

In this chapter, a new minor-embedding algorithm is introduced, which utilizes information extracted from a planar representation of the input graph, contingent on specific conditions being satisfied by the planar representation. A linear-time algorithm for planarity testing of graphs inspired us to define a restricted planar representation of a planar graph H, which will be used for minor-embedding it in a grid graph G. Contingent on being provided with a qualified planar representation, the algorithm can minor-embed all planar graphs in a bounded grid graph. A new technique to manage construction of vertex-models, called *dynamic arm assignment*, is introduced to reduce the solution size. For each minor-embedding, the area of the smallest axis-aligned rectangle in the target grid confining all the vertex-models is referred to as the embedding area of the solution. Using the dynamic arm assignment technique, the algorithm achieves an arguably desirable upper bound on the embedding area, which does not depend on the number of edges of the input graph.

3.1 Planar Embedding Algorithm

Basic definitions of planarity and planar embedding have been provided earlier. Boyer and Myrvold introduced a linear-time planar embedding method [10], which will be referred to as the BM algorithm henceforth. A planar embedding can be viewed as a set of specified clockwise orders for edges incident to each vertex. Given a simple, undirected graph G as input, the BM algorithm's output is either a planar embedding (the edge ordering for each vertex), or an isolated Kuratowski subgraph of G. What made this algorithm intriguing for us was the process of generating a planar embedding.
The BM algorithm iterates over the vertices of the input graph in a reverse DFS visit order. We assume the reader is familiar with the depth first search (DFS) algorithm, however a short reminder of the concept is presented here. The depth first search algorithm is a

linear-time method of traversing a graph, in which the first vertex to visit is called the *root* and is added to a stack S. Vertices in S are popped out and visited, one by one, and before marking each vertex as visited, its unvisited neighbours are pushed onto to S. The algorithm terminates, when all vertices have been visited.

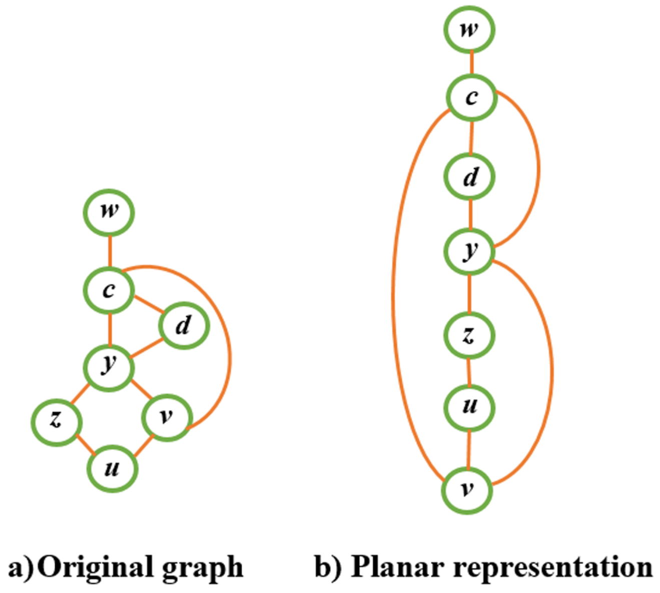
The BM algorithm creates a data structure for the embedding, which contains a set of biconnected components of the embedding and is constructed and maintained, as the edges of the input graph are added to it. Addition of edges to the embedding occurs, while iterating over the vertices of the input graph. During each iteration, a number of edges might be added to the embedding, and the embedded components are then manipulated in a way that either the planarity is preserved, or a forbidden subgraph is isolated. This goal is achieved by recognizing the *externally active* vertices, and keeping them on the outer face of the embedding. As the algorithm iterates over vertices, planar biconnected components are created and merged together, while realizing edges with endpoints in two different components. A component might be flipped to prevent an edge from altering the outer face while it includes externally active vertices. As pointed out in Chapter 1, the technique employed in the BM algorithm is referred to as the edge addition technique. For a planar input graph, the final output of the BM algorithm is the specific order of edges, which are generated through iterating over vertices in the reverse DFS order and finding specific vertical and horizontal orderings for embedding them in the plane, while connecting neighbouring vertices with non-crossing simple arcs.

Inspired by the edge addition technique, we define a restricted planar graph representation to be utilized by the minor-embedding algorithm. Such a planar representation may not exist for all planar graphs. However, any planar graph $H = (V_H, E_H)$ which has a planar representation D, satisfying the following conditions, can be minor-embedded in a bounded grid graph, using the minor-embedding algorithm to be introduced.

- The X coordinate of $D(v)$ in the plane is the same for all $v \in V_H$, and vertices are represented on a virtual vertical line ℓ in the plane.
- For each edge $(u, v) \in E_H$,
  - if $D(u)$ appears exactly before or after $D(v)$, in the ordering of vertices on ℓ, $D(u, v)$ is a vertical line segment, connecting the two;
  - otherwise, $D(u, v)$ is a simple arc, located entirely either on the left or the right side of ℓ.

A planar representation D of graph H that satisfies the above conditions is referred to as a *columnar planar representation* of H, which also realizes a *2-page book embedding* of H. In that context, ℓ is called the *spine* of the book embedding, which is shared by the two half planes (*pages*) on its sides, and the representation of each edge must be entirely in one page. A planar graph $H = (V_H, E_H)$ has a 2-page book embedding if and only if it

Figure 3.1: Figures a and b depict a graph and a columnar planar representation of it, satisfying the required conditions for being an input for the minor-embedding algorithm.



is a *sub-Hamiltonian* graph, which means it is either Hamiltonian or can be turned into a Hamiltonian graph by adding edges to E_H, without making H non-planar[41].

Figure 3.1 illustrates a planar representation of a graph that satisfies the above conditions. A columnar planar representation of the input graph is the foundation of the minor-embedding algorithm presented in this chapter. Given a columnar planar representation of a planar graph H with n vertices, the algorithm finds a minor-embedding of H in a grid graph G with maximum area of $n \times (n-1)$. In fact, it will be explained how the grid will have n rows, and it usually will have fewer than $n-1$ columns.

The algorithm utilizes the vertical order of vertices in the planar representation, along with the *orientation* of each edge, indicating the side (right or left) of the stack of vertices on which the simple arc representing the edge is located. Orientation is not defined for the edges which are represented by vertical line segments.

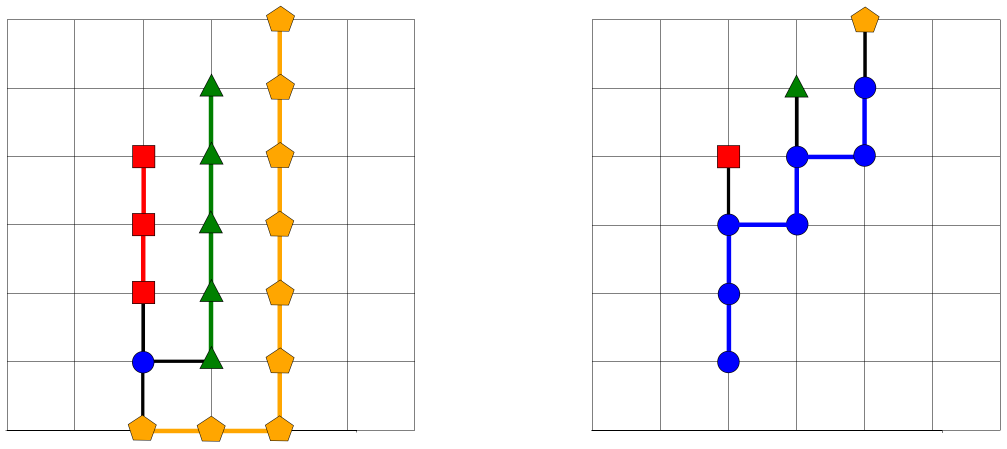
Verifying the existence and generating a columnar planar representation of H may not be trivial. In general, the problem of determining if a given planar graph H has a 2-page book embedding is NP-complete[41]. There might be several approaches for obtaining the required information to generate a columnar planar representations of the input graph, depending on the case. For instance, suppose the vertical order of vertices of the required planar representation D of the input graph H is known, and D realizes the output of running the BM algorithm with H as the input. If a vertex $v \in V_H$ is adjacent to the two vertices, appearing before and after it in the vertical order of vertices in D, it is possible to compute the orientation of edges, incident to v, which connect it to its other neighbours. That is because D realizes the output of the BM algorithm, which provides a clockwise

ordering of edges incident to v. We know two edges, incident to v, will be represented by two vertical line segments, connecting $D(v)$ to the representations of its neighbours above and below it. Applying the known clockwise ordering to the edges, incident to v, will determine the orientation of edges which will be represented by simple arcs. If for each edge $(u, v) \in E_H$, either of u or v are adjacent to the two vertices, represented above and below its representation, the orientation of $e = (u, v)$ can be determined. In this instance, if the adjacency matrix of H makes it possible to determine the orientations of all edges in E_H, D satisfies the required conditions for being used in the minor-embedding algorithm.
In the next section, a method for managing vertex-models will be introduced, which can be employed in minor-embedding algorithms based on finding shortest paths between vertex-models. Afterwards, the minor-embedding algorithm which uses this technique and a columnar planar representation of the input graph will be explained in detail.

3.2 Constructing Vertex-Models Using Dynamic Arm Assignment

This method can be used in any arbitrary minor-embedding algorithm that embeds the roots of vertex-models (root-vertices) in the target graph and then adds the vertices, forming the shortest paths between root-vertices, to appropriate vertex-models. The sum of sizes of vertex-models is influenced by the strategy of adding vertices of the target graph to the vertex-models. Consider the example illustrated in Figure 3.2, in which the difference between the various strategies and their impact can be observed. In the D-Wave algorithm, vertices are added permanently to vertex-models right after finding the shortest paths between a root-vertex and its adjacent root-vertices. It will be demonstrated that in many cases a finalized addition can be delayed, and deferring the definite addition of a vertex $v_g \in V_G$ to a vertex-model can contribute to better solutions. In the dynamic arm assignment technique, we make partial decisions for each vertex $v_g \in V_G$ first, limiting the possibilities for v_g only as much as it is required, and then, at the final stage, the vertex is added to a vertex-model.
Before describing this technique, we should introduce some terminology. For a vertex $v_g \in V_G$, we can *make v_g unavailable* or inaccessible by assigning infinite weights to all or some of the edges incident to v_g. In addition, we can *make an edge unavailable* and prohibit it from appearing in a shortest path by simply setting its weight to infinity. In order to make v_g, *available* again, we need to reverse the changes made to make the vertex unavailable in the first place. Depending on the context, some or all of the edges incident to v_g must be modified.
The dynamic arm assignment technique is implemented using two different data structures. The first of these data structures consists of, for each vertex $v \in V_H$, a set A_v of vertices of G, called the *available points* or the *arm* of v. A_v is initialized to contain only the root r_v of

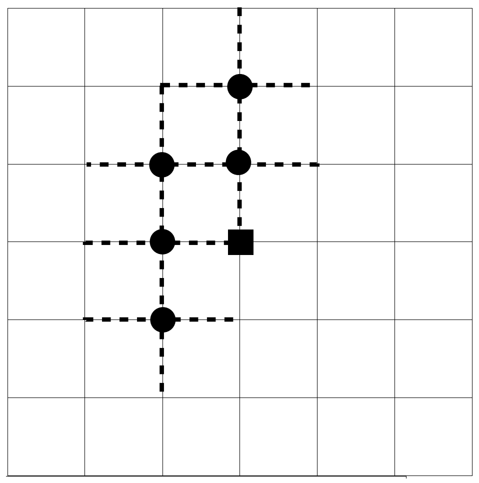
Figure 3.2: Blue(circle) vertex is adjacent to other vertices, and a minor-embedding algorithm embeds the edges adjacent to the red (square), green (triangle), and orange (pentagon) vertices, in that order. Figures depict the results of two possible strategies for adding vertices to the vertex-models.



$\phi(v)$ and then extended over time to be a potential vertex-model of v. Each vertex $v_g \in A_v$ can be contained only in shortest paths between r_v and other root-vertices, unless v_g also belongs to sets of available points of vertices other than v. The arms of vertices of H are used to restrict the set of edges of G that must be available before finding a shortest path between any two root-vertices. After a shortest path is found between two root-vertices, some modifications might be made to the available points of a subset of vertices of H. Initially, all vertices in all arms are unavailable, which means all edges incident to those vertices have infinite weights. Given $v, u \in V_H$, before searching for a shortest path between r_u and r_v, roots of $\phi(u)$ and $\phi(v)$, A_u and A_v are made available, without making vertices of G in arms of other vertices of H available. Before finding the shortest path between r_u and r_v, only the *proper set of edges* incident to each vertex in A_u and A_v can be made available. Given r_u and r_v are the source and the destination of the shortest path to be found, the set of edges of G incident to a vertex g in A_v or A_u, and which are not incident to vertices in arms of vertices other than u and v, are referred to as the proper set of edges incident to g. Figure 3.3 depicts an example of how making the arm of one vertex available can accidentally make an unwanted vertex available.

The second data structure consists of, for each vertex $v_g \in V_G$, a set of vertices of H that have v_g in their available points and is referred to as the *point-permit* of v_g. Point-permits of vertices of G are crucial to maintaining arms of vertices of H because they are used to determine the permitted modifications to available points of vertices. The point-permit of each vertex can only have three states, and each of these states provide the following

Figure 3.3: Vertices illustrated by circles and a square, belong to available points of two different vertices. If all edges incident to the circles are made available in the grid, an unwanted path can contain the vertex marked by the square.

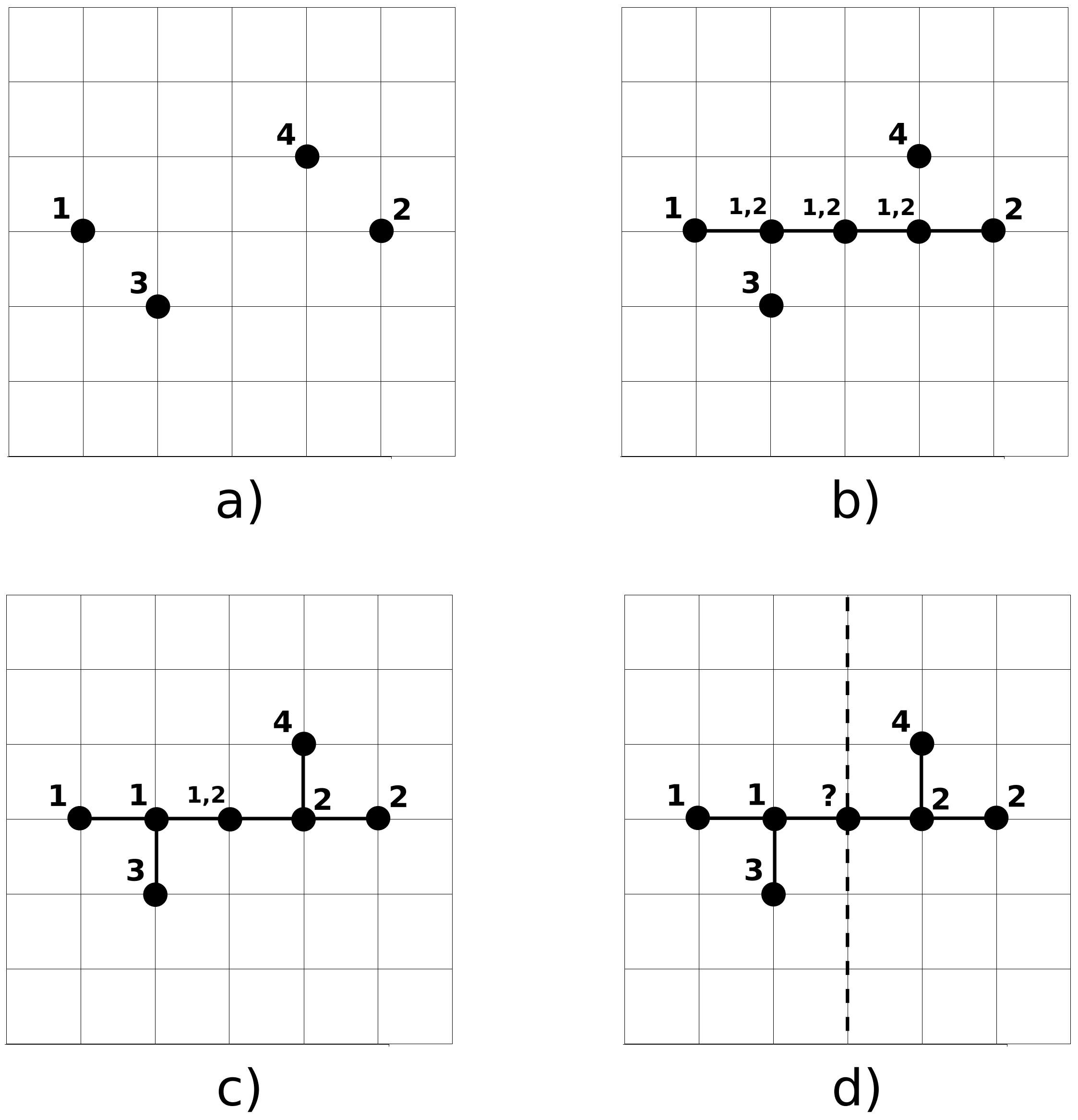


information. If the point-permit, P_{v_g}, of a vertex v_g is empty, v_g is permitted to be a part of a shortest path between any two root-vertices. That is because no other paths have contained v_g so far. On the other hand, if the point-permit of a vertex v_g contains only the index of one vertex v, it must be that $v_g \in \phi(v)$. The other possible state for a point-permit corresponds to making partial decisions. In this state the point-permit contains two vertices of H and v_g is not immediately added to the vertex-model of either one of them.
The status of the point-permits and arms of vertices will change by a certain set of transitions. After embedding only the roots of vertex-models, point-permits of all vertices of G are empty, except for the point-permit of each root-vertex r_v which contains only the index of v. Every time a new shortest path is chosen between two root-vertices r_u and r_v, the point-permit of each $g \in V_G$ in the chosen path will be updated as follows:

- if $P_g = \emptyset$, then both u and v are added to P_g. Also, g is added to both A_u and A_v.
- if $P_g = \{u\}$ or $P_g = \{v\}$ no changes are made to either P_g or arms of the vertices. At this point, it is certain that g belongs to the vertex-model of the only vertex in P_g.
- if P_g has two elements, including only one of u and v and another vertex $w \in V_H$, the decision about the vertex-model which will contain g must get finalized. This is done by removing w from P_g and also removing g from the available points of w.

Figure 3.4 demonstrates a possible scenario under which all these transitions take place, and how this method can lead to smaller sizes for vertex-models, compared to methods which make the assignments (addition of vertices of G to vertex-models) right after finding

Figure 3.4: Populating vertex-models 1-4, using the dynamic arm assignment technique. Edges (v_1, v_2), (v_1, v_3), and (v_2, v_4) are embedded, respectively. Each number indicates the vertices in the point-permit of the grid vertex, next to it.



each shortest path. Suppose dynamic arm assignment is to be used for minor-embedding a planar graph H in a grid graph G. The scenario involves embedding some of the edges, incident to 4 vertices of H, which are referred to as vertices 1 to 4.

In Figure 3.4, picture a illustrates embedding of roots of vertex-models of vertices 1 to 4. The numbers close to each vertex g of the grid indicate the indices of vertices in the point-permit of g. No numbers are presented for vertices with empty point-permits. In picture

b, the edge between vertices 1 and 2 is embedded, and all grid vertices in the shortest path between 1 and 2 have been labelled by both 1 and 2 (vertices 1 and 2 are added to the point-permits of those grid vertices). Vertex 3 is only adjacent to 1, and vertex 4 is only adjacent to 2. The two edges connecting vertices 1 and 3 to their neighbours are embedded in G in two different iterations, using the available points of the two end points of each edge, at each iteration. The resulting paths and the updated point-permits can be observed in picture c. There is still one vertex in the grid with both vertices 1 and 2 in its point-permit. As illustrated in picture d, it can be employed later for connecting the root-vertex of either 1 or 2 to one of their neighbouring root-vertices.
A reasonable probability exists for any algorithm, using dynamic arm assignment method, to embed all edges of the input graph and end up with a number of labelled, but not finalized, vertices in the target graph. These vertices with point-permits of size 2 can be assigned to any of the two possible vertex-models at the end, as long as the feasibility of the solution is preserved. This fact introduces an opportunity to apply application-specific preferences and strategies, such as minimizing the maximum size among all vertex-models in the D-Wave Two quantum computer application. In case no preferences are enforced by the application, the fates of dually labelled vertices can be determined arbitrarily, as long as the solution remains feasible.

3.3 Minor-Embedding Algorithm

Before presenting the minor-embedding algorithm, one point needs to be explained about the target graph, an $m \times n$ grid graph, where $m, n \in \mathbb{Z}$. Let $G_{m,n} = (V_G, E_G)$ be the target graph. One method of referring to a vertex $v_g \in V_G$ is to provide the coordinates of $D'(v_g)$, where D' is the planar representation of G, which is a square grid with unit length edges in the Cartesian coordinates system, and its leftmost column and lowest row contain the origin of the quarter-plane. In this context, a column (row) of G refers to the set of vertices of G that are being represented on the respective column (row) of $D'(v_g)$. Since the algorithm is going to generate a minor-embedding based on a planar representation, referring to vertices of the target graph will be much easier using this convention.
The algorithm starts with labelling vertices of H in the sequence $S = \{v_1, v_2, \dots v_n\}$, in the ascending vertical order of their representations in the provided columnar planar representation D. Root-vertices are embedded on adjacent vertices of a single column of G based on the order in S. For each $v \in V_H$, the root of $\phi(v)$ is denoted by r_v. A_v is initialized to contain only r_v, and the point-permit of r_v, P_{r_v} is set to $\{v\}$. The set of edges between root-vertices in E_G are called the *spine* of the minor-embedding, and if two vertices $v_i, v_{i+1} \in S$ are adjacent to each other, the edge between them will be represented by an edge on the spine, which connects r_{v_i} to $r_{v_{i+1}}$. Then, given the orientation of edges, the algorithm embeds the remaining edges of H around the spine. Further details of these operations will be

presented shortly.
The Y coordinate of each root-vertex can be directly derived from its position in S. For each vertex $v_i \in S$, we embed r_i on a grid vertex with coordinates (x, i). The value of x is the same for all vertices, and we will discuss how it is determined, shortly. When finding shortest paths to represent edges, it is necessary to enforce the edge orientations to prevent the paths from *trapping* other vertices. A vertex is said to be trapped, when due to the arrangement of other found paths in the target graph, it is not possible to find a path between that vertex and at least one of its neighbours, through the available vertices in the target graph. The algorithm iterates over vertices in S in the ascending order. For each vertex $v_i \in S$, the set of *low neighbours* of v_i, denoted NB_i, is defined as the set of all vertices $v_j \in S$, where $j < i$ and $(v_i, v_j) \in E_H$. In iteration i, for $1 \leq i \leq |S|$, a shortest path is found between r_i and the root-vertex of each vertex in NB_i, using the dynamic arm assignment technique and the A* algorithm. The edges are embedded on the predetermined sides of the spine in the following order. NB_i is sorted in the reverse order of appearance of vertices in S, and based on that, the algorithm embeds incident edges to v_i. This way, r_i is connected to the closer neighbouring root-vertices first. After embedding each edge, arms of all vertices are made unavailable. It will shortly be discussed why embedding edges in this order will not trap other vertices and results in finding a minor-embedding of H in G. Each $m \times n$ grid has n columns of $m - 1$ vertical edges, and m rows of $n - 1$ horizontal edges. If there exists a column (row) for which non of its vertical (horizontal) edges have been a part of at least one of the shortest paths found between two root-vertices, that column (row) can be eliminated from the grid without affecting the feasibility of the final solution. On the other hand, a *used column* is one containing an edge of a shortest path, and hence, cannot be removed. Removing a column that crosses the point $(i, 0)$ of the plane is equivalent to replacing each vertex $v \in V_G$, with X coordinate $x_v > i$ in any of the vertex-models, by a vertex with coordinates $(x_v - 1, y_v)$ and removing vertices with $x_v = i$ from all vertex-models. A similar statement holds for removing a row of the grid. Each removed row or column decreases the consumed area of the embedding, and also reduces the total size of vertex-models by the number of shortest paths through it.
Using the dynamic arm assignment technique, the algorithm adds at most one used column on each side of the spine at each iteration i. This can be shown, using the ordering of S and the optimality of A*. First we need to show that the A* algorithm used for finding shortest paths in the minor-embedding algorithm is optimal. We use *Manhattan distances* of vertices of G as the A* heuristic. The definition of the Manhattan distance of two vertices $P_1, P_2 \in V_G$, with respective coordinates of (x_1, y_1) and (x_2, y_2) follows:

$$d_M(P_1, P_2) = |x_2 - x_1| + |y_2 - y_1|$$

This value is exactly equal to the cost of the shortest path between P_1 and P_2, if all edges in E_G are available and have a unit weight. Further, considering the search graph is a grid graph, the Manhattan distance is an admissible heuristic which guarantees A* optimality. At each iteration i, due to the order of iterations over V_H and optimality of A*, all grid vertices on the same row as r_i have empty point-permits, except for r_i itself. Moreover, incident edges to each vertex are embedded exactly in the order in which their corresponding simple arcs appear in the columnar planar representation. Since no two simple arcs cross each other, anytime the algorithm attempts to find a shortest path between r_i and the root-vertex of an element in NB_i, the grid vertex adjacent to the root of that neighbour either has an empty point-permit or is in the available points of the neighbour. Each path of length more than one starts and ends with at least one horizontal edge in the grid with a sequence of vertical and horizontal edges in between. Suppose a shortest path is to be found between r_i and r_j, where $j < i$, on a specific side of the spine. Provided by the vertical ordering of vertices and the clockwise ordering of arcs in the columnar planar representation of the input graph, for any $j < k < i$, no paths can exits between r_k and another root r_l, such that $l > i, l < j$, on the same side of the spine. r_k is referred to as a *covered* vertex on that specific side of the spine. All paths on each side of the spine, originating from r_i, are subgraphs of a sub-tree of G for which r_i is the tree root, and root-vertices corresponding to vertices in NB_i are its leaves. All such subgraphs have an even number of horizontal edges. In the *ith* iteration, the only possible destinations for a shortest path, on each side of the spine, are the non-covered vertices on that side of the spine. Non-covered vertices either do not belong to trees on the same side, corresponding to paths originating from other root-vertices, or are the highest or the lowest grid vertices of such trees, in G. The minor-embedding has the maximum number of used columns if the generated tree, at each iteration, covers only the vertices in the most recently embedded tree on the same side of the spine by finding a path around that tree, from one vertex above its highest point to one vertex below the lowest point of the covered tree. Each such tree needs to use only 1 extra column of the grid, as a result of applying the dynamic arm assignment technique. If the formations of trees on both sides of the spine follow this scenario, the embedding will require $2(n/2) - 1$ or $n - 1$ columns. Figure 3.5 demonstrates examples of the occurrence of the extreme cases for graphs with 6(even) and 7(odd) vertices. It should be noted that there exist other configurations of paths that result in the same embedding area.

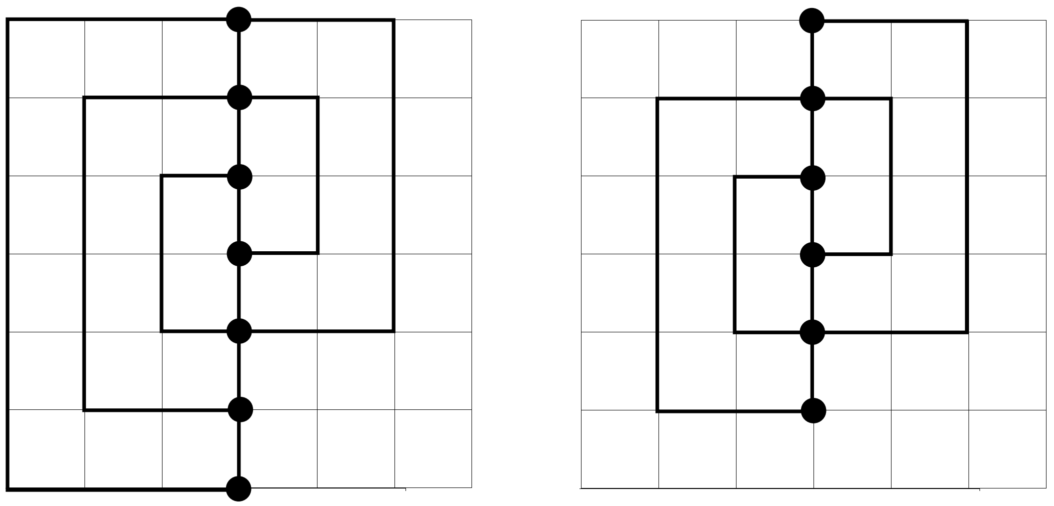
Now, it is clear how the X coordinate of the spine of the minor-embedding is determined. At the beginning, n columns are made available to the embedding, and the spine is embedded on the $\lceil \frac{n}{2} \rceil^{th}$ column. In the most extreme case, one column, and usually more columns remain unused which are simply removed from the grid.

The only edge of H for which knowing the orientation is not necessary, is the one connecting the highest vertex on the spine to the lowest one, if it exists. That edge is the last edge to

Algorithm 1 Algorithm A for minor-embedding a planar graph based on a columnar planar representation of it

```
Input: Planar input graph $H = (V_H, E_H)$ and a columnar planar representation of $H$
Output: Minor-embedding of $H$ in a grid graph $G$ with maximum dimensions of $n \times (n-1)$,
    where $n = |V_H|$
 1: for $i = 0$ to length$(S) - 1$ do
 2:     Set $r_{v_i} = v_{g_i}$, where $v_{g_i} \in V_G$ has the coordinates $(\lceil \frac{|V_H|}{2} \rceil, i)$.
 3:     Set $A_{v_i} = \{r_{v_i}\}$ and make $r_{v_i}$ unavailable.
 4:     Set point-permit of $r_{v_i}$ to contain only $v_i$.
 5: end for
 6: for all $v \in V_H$, in the ascending order of $S$ do
 7:     for all $u \in \mathrm{NB}_v$, in the descending order of $Y$ coordinates of their root-vertices do
 8:         Make vertices in $A_v$ and $A_u$ available, by setting the weights of the proper set of
            edges incident to each vertex to the unit weight.
 9:         Use A* with Manhattan distance heuristic to find the shortest path between $r_v$ and
            $r_u$, considering the orientation of $(u, v) \in E_H$.
10:         Update the point-permit of all vertices in the chosen path, and also arms of all
            affected vertices in $V_H$, using the defined transitions (Dynamic Arm Assignment).
11:         Re-set the weights of edges incident to vertices in $A_v$ and $A_u$ to $\infty$.
12:     end for
13: end for
```

Figure 3.5: Minor-embedding two graphs with $n = 6$ and $n = 7$ vertices in a grid graph. Each embedding requires n rows, and at most $n - 1$ columns



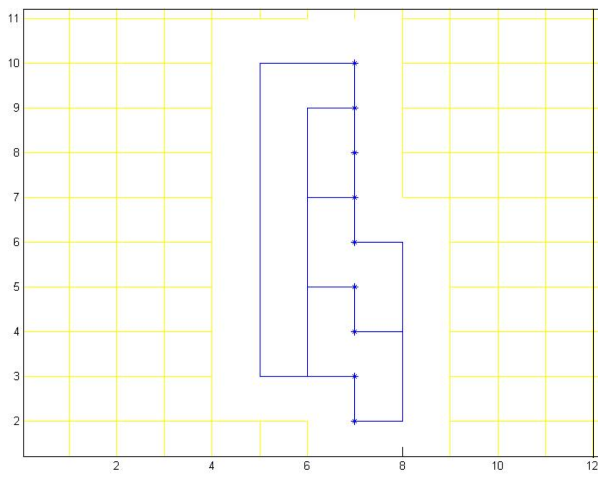
be embedded and cannot trap any vertices. Algorithm 1 summarizes the minor-embedding algorithm, which will be referred to as Algorithm A henceforth.

3.4 Experiments and Analysis

Algorithm A can successfully minor-embed a planar graph H with n vetices in an $n \times (n-1)$ grid graph G, being provided with a columnar planar representation of H. The time complexity of the algorithm can be analyzed, assuming that the columnar planar representation of H is provided in the input. The running time of the algorithm is bottlenecked by running A* for each edge $e \in E_H$. G has n^2 vertices, so running A* requires a priority queue with the distance estimates for at most n^2 vertices. Hence, removing the vertex with the minimum distance estimate from the priority queue, or updating the distance estimate of each vertex in the priority queue is done in $O(\log n^2) = O(\log n)$ time. Each single A* search requires $O(n^2)$ stages, because of the number of vertices in G. At each stage, the vertex with the minimum distance estimate is removed from the priority queue, and the distance estimates of its (at most) 3 unvisited neighbours are updated with a cost of $O(\log n)$ time. Therefore, the time complexity of each A* search in Algorithm A is $O(n^2 \log n)$. Moreover, we know that H is a planar graph, and if $n \geq 3$, $m \leq 3n - 6$, where $m = |E_H|$. Based on that the run time of all A* searches in Algorithm A is $O(n^3 \log n)$, for $n \geq 3$.
In practice, the algorithm's performance is expected to be much better than would be implied by the proven upper bound, for the following reason. As discussed earlier in Section 2.4, the worst-case time complexity of A* is $O(b^l)$, where b is the branching factor of the search problem and l is the length of the shortest path between source and destination. Furthermore, we know that a heuristic is optimal, if it indicates the exact distance between a vertex and the destination, and in that case, the effective branching factor is equal to 1 [56]. If the search space is a grid graph, Manhattan distance is an optimal heuristic, if there are no unavailable edges in the graph, and even otherwise, it is still fairly close to an optimal heuristic for the application of A* in Algorithm A. Based on that, in practice, Algorithm A is expected to run in close-to-linear time in $|E_H|$.
The main limitation of Algorithm A is its dependence on the columnar planar representation of the input graph. The two key pieces of information to be extracted from the planar representation are the ordered sequence of vertices, and the orientations of edges located on the sides of vertices. Figure 3.6 depicts the result of minor-embedding the grid graph $G_{3,3}$, as an example of a relatively small graph for which we can generate a columnar planar representation and minor-embed it in a grid graph. For this purpose, an implementation of the BM algorithm [8] was used to output a planar embedding of the $G_{3,3}$ alongside the DFS visit order, employed by the BM algorithm, which was used to hand generate a columnar planar representation of $G_{3,3}$. The vertical ordering of vertices and the edge orientations of the planar representation were passed to an implementation of Algorithm A, leading to the illustrated minor-embedding in Figure 3.6.

A strategy is required to make Algorithm A applicable to all planar graphs. We present two approaches to this problem, shortly. A sketch of the first approach is provided for

Figure 3.6: Minor-embedding of $G_{3,3}$,with 9 vertices, in a $G_{9,4}$. Shortest paths found by Algorithm A are depicted in the figure.



further investigation in future, and the second approach is pursued in the next chapter.

The first approach uses the BM algorithm to generate a columnar planar representations for subgraphs of the input graph, which can be minor-embedded separately in the target graph. After that, finding paths between certain vertices, in different embedded subgraphs of the input, can generate a solution for the main problem.

Recalling the outlined employed technique by the BM algorithm, biconnected components of the planar embedding are merged together, when an edge is to be added to the embedding which connects the two components. For each two biconnected components which are supposed to be merged together, a vertex is represented in both components, as the root of one and an element on the outer face of the other one. Merging the two components occurs by possibly flipping the one which has the vertex representation as its root, merging the two representations of the same vertex, and embedding the connecting edge between the two components. The root of each component has a flag, which indicates if the component was flipped in the merge process. The resulting component is a biconnected component, as well, with all its externally active vertices on its outer face. If Algorithm A is used to minor-embed two components, which are required to be merged together, instead of merging root-vertices which correspond to the representations of vertex $v \in V_H$, we can find a path between them in the target graph and add the vertices in the path to $\phi(v)$. Moreover, both components keep their externally active vertices on their outer faces, which makes it possible to find a path between available points of the two end points of the edge, connecting the two components.

The key to making this strategy work is to generate the described set of planar represen-

tations of subgraphs of the input graph. Trivially it is preferred to keep the number of separated planar representations as small as possible. An algorithm for generating the planar representation(s), based on the BM algorithm, will probably have a certain set of steps per each step of the BM algorithm. The vertical order of each planar representation can be determined, using the BM reverse DFS visit order, and the collection of flip flags of roots of biconnected components is expected to determine the edge orientations of planar representations. The important attribute of any such algorithm should be to avoid unnecessary increase in number of planar representations. While iterating over vertices of the input graph, each vertex should be represented in an existing planar representation, if the result will remain a columnar planar representation of the subgraph. If doing so is not possible (mainly caused by backtracks in the DFS leading to tree nodes with degrees greater than 2), adding a new planar representation to the set might be involved in the algorithm steps, taken upon the corresponding merge step in the BM algorithm.
The second strategy involves an extension of the BM algorithm, introduced by Boyer to calculate a visibility representation of the input graph, besides finding a planar embedding for it [9]. In [9], vertices are assigned with labels *above* or *between*, compared to a subset of their DFS ancestors when the BM algorithm is merging the biconnected components. The vertices are post-processed by being traversed in pre-order DFS, and then using the assigned labels, a finalized vertical order of vertices is generated. To compute the correct arrangement of vertical lines of the visibility representation, the algorithm connects the DFS tree root to its neighbours, and keeps track of the first edge connected to each vertex as its *generator edge*. Since the BM algorithm generates the edge ordering for all vertices, it can draw the vertical lines corresponding to edges in the correct order just with a 360 degree sweep around the generator edge of each vertex.
Applications of visibility representation in generating graph drawings is well studied in the literature. The second approach is to use a specific visibility representation of the input graph to minor-embed it in a bounded grid graph. A detailed presentation of a minor-embedding algorithm based on this approach is presented in the following chapter.

Chapter 4

Minor-Embedding Based on Visibility Representation

In this chapter, we introduce an efficient algorithm for minor-embedding a graph H in a grid graph, using a visibility representation of H. As stated earlier, a graph H has a visibility representation in the $\mathbb{R}^2$ plane, if and only if H is planar. The following algorithm will minor-embed any planar graph in a bounded grid graph.

4.1 Application of Visibility Representations to Minor-Embedding Problem

Visibility representations are widely employed in graph drawing algorithms. In this section, it is shown how a visibility representation of a graph can be used in minor-embedding it in a grid graph. The orthogonal drawing algorithms usually start by computing a visibility representation of the planar graph in linear time. Most of these algorithms are limited to graphs of maximum degree 4. In a visibility representation, each vertical line segment has two end points, which belong to two horizontal line segments. If ℓ is one of the two horizontal line segments, the other one is either above or below ℓ in the plane. In a visibility representation of a graph with maximum degree of 4, there exist only $\binom{4}{2} = 6$ different cases of connecting a horizontal line segment to (at most) 4 others, based on their relative positions in the plane (counting symmetrical cases as one). Since most drawing algorithms represent each vertex with exactly one point, six schemes of gateway assignment are designed by embedding 4 edges incident to a vertex, based on the 6 possible positionings of the representation of the vertex relative to the line segments, representing its 4 neighbours, in the visibility representation [28].

The problem of minor-embedding a planar graph H in a grid graph G has a different definition than planar embedding H in G. In particular, to find a minor-embedding of H in G a vertex-model needs to be found for each vertex, and a vertex-model can contain a subset of vertices of G, as opposed to only one vertex. This difference allows minor-embedding planar graphs with maximum degrees of more than 4, in a grid graph and is the basis of our minor-embedding algorithm to be introduced.

A visibility representation Γ_H of H can be used to generate a minor-embedding of H in G. If the lengths of all line segments of Γ_H are made divisible by a real number ι, then Γ_H can overlap a square grid with edge lengths of ι, representing G, such that the end points of all line segments of Γ_H are exactly placed on points, representing vertices of G. It is easy to map each horizontal (vertical) line segment of Γ_H to the path in G, which is represented by the sequence of horizontal (vertical) edges of the square grid, overlapping the line segment, and also map that specific path in G to the line segment which is mapped to the path. For a vertex $v \in V_H$, vertices of G on the path which is mapped to the horizontal line segment $\Gamma(v)$ are added to $\phi(v)$. Given $(u, v) \in E_H$, vertices of G on the path which is mapped to the vertical line segment $\Gamma(u, v)$ can be added to either $\phi(v)$ or $\phi(u)$. The set of generated vertex-models satisfies all conditions to be a minor-embedding of H in G. In this chapter, we will refer to this method of minor-embedding H in G, using Γ_H, as the *simple method.* Implemented in [8], the extension of the BM algorithm can generate a visibility representation Γ_H of H and output it in the format of an ASCII art representation of Γ_H. In the generated visibility representation, vertices of the input graph are represented by ordered horizontal line segments with unique Y coordinates in the Cartesian plane and specific lengths. Edges of the input graph are represented by ordered vertical line segments with unique X coordinates in the plane, terminating at the two horizontal line segments which represent the end points of each edge. Furthermore, the distances between consecutive horizontal (vertical) line segments of the visibility representation are set to a unit value [9, 8]. The generated Γ_H, for a planar graph $H = (V_H, E_H)$, can be drawn in a grid area of dimensions $|V_H|$ by $|E_H|$. Figure 4.1 depicts the ASCII art of Γ_H for graph H with $|V_H| = 14$ and $|E_H| = 36$.

Since each planar graph H has a visibility representation, and the BM algorithm extension can be used to find a visibility representation Γ_H of H as presented, the simple method for minor-embedding H in the grid graph $G_{m,n}$, where $m = |E_H|$ and $n = |V_H|$ is complete. However, we believe that almost always using the simple method will not lead to the optimal solution in terms of the solution size, because of the following reason. In a visibility representation, edges of H are represented by non-crossing vertical line segments, and each of these segments contribute to disjoint sets of vertices of G being added to the vertex-models. Several paths in G which represent vertical line segments of Γ_H could be replaced with paths which have shared subtrees, without affecting the feasibility of the minor-embedding. Figure 4.1 illustrates several parallel vertical lines, originating from one horizontal line to connect it to other horizontal lines. Mapping each of these parallel line segments to disjoint sequences of vertices, which will be added to the vertex-models, is not necessary to generate a minor-embedding.

Figure 4.1: ASCII art of Γ_H. $|V_H| = 14$, $|E_H| = 36$.

```
-----------------0------------------
| |  | |                           |
| |  | -------------4--------------|
| |  |  ||| |  | | |    |   |  |  ||
| |  -12-|| |  | | |    |   |  |  ||
| |   |  || |  | | |    |   |  |  ||
| |   |  || |  | | |    |   -13|  ||
| |   |  || |  | | |    |    |||  ||
| |   |  || |  | | |    |    |-10-||
| |   |  || |  | | |    |    |  ||||
| |   |  || |  | | |    ----5----|||
| |   |  || |  | | |     |||     |||
| |   |  || |  | | ---6---||     |||
| |   |  || |  | |  ||||  ||     |||
| |   |  || |  | -11-|||  ||     |||
| |   |  || |  |  |  |||  ||     |||
| |   |  || |  ---9---||  ||     |||
| |   |  || |   |     ||  ||     |||
| |   |  || -----7-----|  ||     |||
| |   |  ||  ||        |  ||     |||
| |   |  |-8--|        |  ||     |||
| |   |  | |  |        |  ||     |||
| ------------3------------|     |||
|  ||                      |     |||
-2--|                      |     |||
 |  |                      |     |||
 -----------------1-----------------
```

In the next sections, we introduce the two phases of an algorithm to find and optimize a minor-embedding of H in G, using a valid visibility representation of H. The minor-embedding algorithm is called the VRM algorithm and runs in two phases. The goal of the first phase, also referred to as the embedding phase, is to find a minor-embedding of H in G. The presented method in Section 4.2 for achieving this goal will be referred to as the *advanced method*, in the remaining of this chapter. In the second phase, known as the optimization phase, the obtained embedding will be optimized to decrease the solution size and the embedding area, while preserving the feasibility of the solution. The optimization phase of the algorithm will be presented in Section 4.3.

4.2 The Minor-Embedding Algorithm (First Phase)

The first step of the VRM algorithm is to generate the visibility representation Γ_H, using the extension of the BM algorithm [9]. The distances between each two lines in both sequences of vertical and horizontal lines of Γ_H, are set to 1 unit which is equal to the edge length of a square grid representing the target graph. Having that, we use the coordinates of each line segment to find the roots of vertex-models. For each vertex $v \in V_H$, we denote by r_v the root of $\phi(v)$ and by $\Gamma(v)$ the horizontal line segment of length ℓ_v representing v in Γ_H. The coordinates of r_v in the target graph are denoted by (x_v, y_v). For all v, y_v is equal to the distance of $\Gamma(v)$ from the $x-$axis of the plane, and x_v is set to the floor of the X coordinate of the point in the middle of $\Gamma(v)$. The algorithm continues by setting $A_v = \{r_v\}$, $P_{r_v} = \{v\}$, and making r_v unavailable.
After finding the roots of vertex-models, the algorithm iterates over vertices in the ascending vertical order of their root-vertices. For each vertex v, let the set of its neighbours with their root-vertices below r_v be denoted by NB_v. Iterating over v involves embedding all edges between v and vertices in NB_v in the reverse vertical order of neighbours' root-vertices. Dynamic arm assignment is employed here as well to manage the sets of available points of vertices, during the process of embedding each edge. A*, employing the Manhattan distance heuristic, is used by the algorithm to find the shortest path between two root-vertices. In the first phase of the VRM algorithm, a unit weight is assigned to all available edges of the target graph. The proper sets of edges incident to vertices in the arms of a pair of vertices of H are made available, before finding the shortest path between their root-vertices. After finding each shortest path, the steps of the dynamic arm assignment technique are followed by the algorithm. Before the process of embedding an edge is complete, all edges of the target graph made available for finding the shortest path between the two root-vertices are made unavailable again.

4.3 Optimizing a Minor-Embedding (Second Phase)

In the optimization phase, two main techniques are used to reduce the embedding area and the sizes of vertex-models. In this phase, the algorithm iterates over all vertices in a random order, and in each iteration, the set of available points of one vertex is eliminated and replaced with a new one. This process can be referred to as re-embedding the vertex. Once the algorithm re-embeds all vertices of H, in $|V_H|$ iterations, a *round* is completed. The stopping criteria for the optimization phase is termination after a fixed number of rounds.
Re-embedding a vertex $v \in V_H$ has three key steps. The first step is to find a new root-vertex for v which will be the root of the new $\phi(v)$. The first of the two main techniques is employed in this step to reduce the embedding area and the solution size. The second

key step is to safely remove the existing available points of v and initializing the new A_v. Available points of different vertices of H and point-permits of vertices of the target graph are related to each other, and this is taken into account in the second step of re-embedding a vertex. The third step is to embed the edges incident to v, using the A* algorithm and the dynamic arm assignment technique. The second main technique of the optimization phase is used in this step and contributes to reducing the solution size.

The first key step of re-embedding each vertex in the optimization phase is choosing a vertex in G as its new root-vertex. This choice must be made carefully because it is probable that changing the relative vertical or horizontal ordering of a few root-vertices will lead to failure in embedding a number of edges. During an iteration in which a vertex is re-embedded, if the algorithm fails at embedding an edge or if the result of that iteration is not better than the best found solution to that point, the iteration is referred to as an unsuccessful iteration. Since the optimization phase begins with a minor-embedding, the algorithm only accepts the successful modifications and reverts the unsuccessful iterations by not replacing the already existing arm with the new one. However, the more successful re-embedding of vertices occur during each round, the more the quality of the solution tends to improve. Hence, the method used for choosing new root-vertices needs to contribute to making that iteration successful, as much as possible. That is why the first main technique of the optimization phase is employed in this step to try to reduce the embedding area, while trying to preserve the feasibility of the solution. In order to re-embed a vertex $v \in V_H$, the new r_v is selected from the existing set A_v. For each vertex $u \in A_v$, which has only v in its point-permit (is permanently labelled), a *proximity value*, $\psi(u)$, is computed to measure a notion of closeness between each vertex in the available points and the arms of all neighbours of v.

$$\psi(u) = \max_{w \in N_v}\left(\min_{x \in A_w} d_E(u, x)\right)$$

In the presented function ψ, N_v denotes the set of all neighbours of v in the input graph, and d_E represents the Euclidean distance between any two vertices in G. Then, we set $r_v = \operatorname{argmin}_{u \in A_v, |P_u|=1} \psi(u)$. Each permanently labelled vertex in A_v is already connected to the roots of all the neighbouring vertex-models, meaning that finding a shortest path between the newly selected root-vertex r_v and each of the root-vertices of neighbours of v is guaranteed to be possible. However, the probability still exists that the shortest paths found between the new r_v and the root-vertices of a subset of N_v trap the roots of some other vertices in N_v. In that case the re-embedding is unsuccessful, and the result of that iteration is not applied to the solution.

In the second key step, after finding a new root-vertex, the existing set of available points of the vertex needs to be removed. Each vertex of H might share a subset of its available points with available points of other vertices, and also affect the point-permits of a subset of V_G; hence, re-embedding a vertex must be accompanied by necessary modifications to

other structures which depend on it. Upon finding a new root-vertex for $v \in V_H$, the algorithm starts with the process of safely removing vertices of G from A_v, following the required steps to be explained shortly. Vertices in A_v can be divided into two groups: those permanently assigned to $\phi(v)$ with their point-permits containing only v, and those shared between v and another vertex u. If $P_g = \{u, v\}$ for a vertex $g \in V_G$, it means g is only contained in the path between r_v and r_u. Therefore, when elements of A_v are being cleared, g and other vertices in A_v that also appear in the available points of another vertex, can be removed from both sets of available points, and the point-permit of the shared vertex must be emptied. Trivially, point-permits of permanently labelled vertices of G that are only in A_v, must also be emptied, after being removed from A_v. As a vertex $g \in G$ is being removed from A_v, the weights of any edge $(x, g) \in E_G$ must be set to the unit weight, if $P_x = \emptyset$. After following the explained procedure to remove all vertices from A_v, the algorithm sets $A_v = \{r_v\}$, makes r_v unavailable, and sets $P_{r_v} = \{v\}$.
In the third step, after selecting a new r_v and clearing the previous available points of v, the optimization phase proceeds to embed the edges incident to v. The Euclidean distance of the new r_v to the neighbouring root-vertices determines the order of embedding the edges, by prioritizing the closest neighbours over the farthest ones.
The second main technique of the optimization phase tries to reduce the solution size by assigning different weights to edges which are to be made available. In the embedding phase, all shortest paths between a pair of root-vertices are equally acceptable to be found by A*, whereas in the optimization phase, the second technique aims to encourage A* to find a shortest path which uses as many available points of the two vertices as possible. Assuming the algorithm is re-embedding v and needs to find a shortest path between r_v and r_u, for $(u, v) \in E_H$, the arms of u and v must be made available first. Using the second technique, this is done by setting the weights of proper sets of edges, incident to vertices in available points of u and v, to a small real number $\epsilon > 0$. After finding the r_v-r_u path and applying the dynamic arm assignment steps, the modified weights are set to ∞, and the arms are made unavailable again.
With this modification, for a vertex $w \in V_H$, connecting a vertex $z \notin A_w$ to any of the vertices in A_w costs approximately the same. In other words, this tends to make the cost of the selected shortest path between r_v and r_u very close to the cost of the shortest path between the two closest vertices in available points of v and u. The result is significant reduction in the solution size as will be demonstrated in our experiments.
Algorithms 2 and 3 summarize the two phases of the VRM algorithm.

Algorithm 2 The embedding phase of the VRM algorithm

```
Input: Planar input graph $H = (V_H, E_H)$
Output: Minor-embedding of $H$ in a grid graph $G$, and meta-data: the root-vertices and
    available points of vertices of $H$, point-permits of vertices of $G$, and found shortest
    paths
 1: Obtain the visibility representation $\Gamma_H$ of $H$
 2: for all $v \in V_H$ do
 3:     Find $r_v$ in $G$ using coordinates of $\Gamma(v)$.
 4:     Set $A_v = \{r_v\}$ and make $r_v$ unavailable.
 5:     Set $P_{r_v} = \{v\}$
 6: end for
 7: for all $v \in V_H$ in the ascending vertical order of root-vertices do
 8:     for all $u \in \mathrm{NB}_v$ in the descending vertical order of root-vertices do
 9:         Make vertices in $A_v$ and $A_u$ available by setting the weights of the proper set of
            edges incident to each vertex to the unit weight.
10:         Use A* with Manhattan distance heuristic to find the shortest path between $r_v$ and
            $r_u$.
11:         Apply the dynamic arm assignment steps to the vertices in the chosen path.
12:         Set the weights of edges, made available, to $\infty$.
13:     end for
14: end for
```

4.4 Completeness and Time Complexity

The VRM algorithm is complete, meaning that it always succeeds at minor-embedding a planar graph $H = (V_H, E_H)$, in a $|V_H| \times |E_H|$ grid graph G. The proof relies on the fact that the algorithm operates in two phases. In the first phase, the objective is to find a minor-embedding of H in G. First, the advanced method is used to find a feasible solution. If the advanced method fails at doing so, the simple method, explained and proved to be complete at the beginning of this chapter, is used to find a minor-embedding of H in G. It will be shown in Chapter 5 that the advanced method was successful at finding a feasible solution for all instances used in our experiments. The second phase of the algorithm accepts a feasible solution as the input and tries to improve it, in multiple rounds. Each round consists of $|V_H|$ iterations, and re-embedding one vertex at each iteration may lead to a change in the minor-embedding. The solution is modified at the end of each iteration, only if the newly found solution is both feasible and reduces the total size of the vertex-models. Hence, it is not possible for the second phase of the algorithm to make the solution infeasible, and the algorithm is complete.

The time complexity of the algorithm can also be analyzed, separately for the two phases. The visibility representation, required for the first phase, is computed in linear time [9]. The analysis of the remaining steps of the advanced method, follows a similar analysis to Algorithm A, introduced in Chapter 3. That is because the running time of the first phase

Algorithm 3 The optimization phase of the VRM algorithm (one round)

Input: The output of the first phase, containing the root-vertices and available points of vertices of the input graph $H = (V_H, E_H)$, point-permits of vertices of the target graph $G = (V_G, E_G)$, and the found shortest paths

Output: Optimized minor-embedding solution

```
1:  $O$ = a random permutation of $V_H$
2:  for all $v \in O$, in order do
3:    for all $u \in A_v$, satisfying $|P_u| = 1$ do
4:      Compute $\psi(u)$.
5:    end for
6:    $r_v = \text{argmin}_{u \in A_v, |P_u|=1} \psi(u)$.
7:    for all $u \in A_v$ do
8:      for all $w \in P_u$ do
9:        Remove $u$ from $A_w$.
10:     end for
11:     Set $P_u = \emptyset$.
12:     for all edges $(x, u) \in E_G$ incident to $u$ do
13:       if $P_x = \emptyset$ then
14:         Set the weight of $(x, u)$ to unit weight.
15:       end if
16:     end for
17:   end for
18:   Set $A_v = \{r_v\}$, and make $r_v$ unavailable.
19:   $P_{r_v} = \{v\}$.
20:   for all $z \in N_v$ do
21:     Compute $d_E(r_v, r_z)$.
22:   end for
23:   Initiate a priority queue pq of elements of $N_v$, prioritizing neighbours with minimum
      Euclidean distance of root-vertices to $r_v$.
24:   while pq is not empty do
25:     $z$ = find and remove the minimum element from pq.
26:     Make vertices in $A_v$ and $A_z$ available by setting the weights of the proper sets of
        edges incident to each vertex to $\epsilon > 0$.
27:     Use A* with Manhattan distance heuristic to find the shortest path between $r_v$ and
        $r_z$.
28:     Apply the dynamic arm assignment steps to the vertices in the found path.
29:     Set the weights of edges made available to $\infty$.
30:   end while
31:   if iteration was unsuccessful then
32:     Do not modify the existing solution.
33:   end if
34: end for
```

is dominated by running the A* algorithm, for $|E_H|$ times, and G has $|V_H| * |E_H|$ vertices. Using an argument similar to the one employed in Section 3.4, in the VRM algorithm, each A* search has $O(|V_H|x|E_H|)$ stages, and each stage runs in $O(\log|V_H||E_H|)$ time. Furthermore, H is a planar graph, so for $|V_H| > 3$, each A* search runs in $O(|V_H|^2 \log|V_H|)$, and the time complexity of the embedding phase is $O(|V_H|^3 \log|V_H|)$. Also in the VRM algorithm, similar to Algorithm A, the target graph is a grid graph, and Manhattan distance of vertices is used as the heuristic for A*. As a result, the branching factor is expected to be close to 1, and in practice, the embedding phase using the advanced method is expected to run in close-to-linear time in E_H.

In order to analyze the time complexity of the second phase, time complexities of the three described steps of each iteration of each round of the optimization phase are analyzed separately. At each iteration, a vertex $v \in V_H$ is re-embedded, in 3 steps. In the first step, a new root-vertex r_v is selected for the vertex-model to replace the previous one. In the second step, the existing set of available points A_v is safely removed, as described in section 4.3. In the third step, the computed distances between the selected r_v and the roots of neighbouring vertex-models are employed to initialize a priority queue of vertices of H adjacent to v, based on the minimum distances of their root-vertices to r_v. Finally, edges incident to v are embedded, in the same way that edges are embedded in the embedding phase, except for the assignment of weights to the available edges of G and the order of embedding edges, incident to v, which is determined by the priority queue.

While iterating over V_H, for each $v \in V_H$, the first step requires $|A_v| \sum_{u \in N_v} |A_u|$ $O(1)$ computations (calculating the Euclidean distance of two vertices), where N_v denotes the set of vertices of H adjacent to v. Throughout the first steps of all iterations of one round, $\sum_{v \in V_H} |A_v| \sum_{u \in N_v} |A_u|$ of such computations are carried out. For each $v \in V_H$, the value of $\sum_{u \in N_v} |A_u|$ can be upper bounded by $C_1|V_G|$, where C_1 is a constant. This will upper bound the number of computations of one round, required for the first steps of all iterations, to $\sum_{v \in V_H} |A_v| * C_1|V_G| = C_1|V_G| * \sum_{v \in V_H} |A_v|$. Moreover, the value of $\sum_{v \in V_H} |A_v|$ can also be upper bounded by $C_2|V_G|$, where C_2 is a constant. Therefore, the first step of re-embedding vertices, throughout all iterations of one round, runs in $O(|V_G|^2)$ time. The second step runs in $O(|V_G|)$ time for each round. In the third step, the priority queue is initiated in $O(|V_H|)$ time for each of the $|V_H|$ iterations. The time complexity of embedding edges incident to v is similar to the time complexity of embedding edges in the embedding phase, except for the required time for removing the neighbour with the minimum distance from the priority queue, which runs in $O(\log|V_H|)$ time. The time of the optimization phase is an added $O(|V_G|^2 + |V_H|^2)$ per round. However, since $|V_G| > |V_H|$, it can be simply stated as $O(|V_G|^2) = O(|V_H|^4)$.

It should be mentioned that this is a rather naive implementation of the idea, and improving the running time is suggested as future work using pre-computations and different choice of data structures. On the other hand, in practice, due to the sparseness of input graphs, running time is much faster than the introduced upper bound.

Chapter 5

Numerical Results

This chapter presents an experimental evaluation of the VRM algorithm. Necessary definitions and outline of experiments are presented, followed by the essential information for replicating these experiments. Empirical results and analysis will conclude the chapter.

5.1 Definitions and Outline

The conducted experiments aimed to test the effectiveness of the introduced techniques in the VRM algorithm, and also compare the VRM with the D-Wave algorithm on the same input and target graphs. We will define a set of measurements for the output solution of any algorithm that minor-embeds planar graphs in grid graphs, and compare the solutions produced by the two algorithms on the basis of those measurements. Also to evaluate the performance of the optimization techniques in the second phase of the VRM, the outputs of phase 1 (before optimization) and phase 2 (optimized embedding) of the algorithm will be compared using the same measures.

For assessment of both algorithms, our experiments are carried out on graphs of 6 different families, and a number of individually named graphs from the literature. Shortly, we will observe that the structural attributes of the input graph can have a significant role in the probability of success and quality of the minor-embedding.

Following are the measurements based on which we can assess and compare outputs of the algorithms. The first two are the primary ones, which we focus on more, and the other three will help with further analysis.

5.1.1 Sum of Sizes of Vertex-Models (Solution Size)

This parameter was also used in [12] as an indicator of the quality of a minor-embedding. Reducing the sizes of vertex-models is directly related to the capability of the algorithm in minor-embedding larger input graphs in a fixed size target graph. Minimizing the solution size is one of the main objectives of both algorithms.

5.1.2 Probability of Success

The simplest indicator of effective performance of a minor-embedding algorithm is the ratio of successful runs, in which a feasible solution is found, to all attempts. This parameter states the success rate of the algorithm which can be influenced by certain attributes of an input graph such as maximum degree, number and boundaries of faces, as well as number of vertices and edges. The two algorithms under analysis here are able to manage the effects of these factors to different extents.

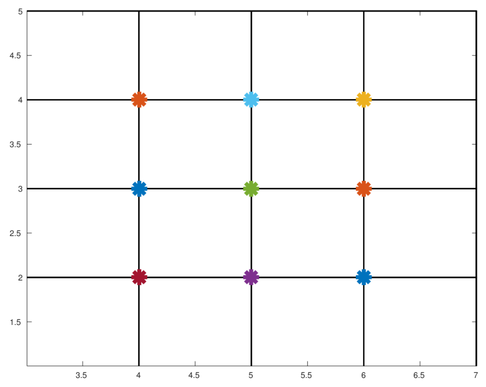
5.1.3 Embedding Area

Suppose we find the minimum axis-aligned rectangular surface on the plane containing G, which confines all vertex-models of the minor-embedding. The number of all the grid vertices on and inside this rectangle is called the embedding area. This definition is derived from that of the area of a grid drawing [28]. The planar embedding area is important in many applications, such as VLSI design in which embedding in the smallest physical rectangular surface is desirable [26]. Similarly, in case of minor-embedding, smaller embedding area is preferable.
In almost all existing and many probable applications, more consumed area, e. g. a larger chip, or a larger subset of qubits of a quantum computer, implies greater costs. Furthermore, employed techniques to reduce the embedding area have a high probability of reducing the average length of paths representing edges. Particularly in the D-Wave application, long paths are discouraged since they have a negative effect on the annealing process.

5.1.4 Area Utilization Percentage

This parameter represents a notion of compactness. We define the area utilization to be the ratio of the solution size to the embedding area, expressed as a percentage. It is important to discuss why area usage can illustrate a quality aspect of the embedding, but should not be an independent optimization target. Higher area usage indicates that a smaller number of "unused" vertices are enclosed in the embedding area. Any vertex in the embedding area, and not in a vertex-model, is considered unused by the solution, and utilizing them for other purposes faces many limitations, particularly those inside the faces of the embedding. For instance, an unused vertex inside a face cannot be used for minor-embedding another input graph in the same grid, unless a complete embedding is to be found in that specific face.
Although this parameter can assess the density of an embedding, the changes made to it by the optimization phase does not essentially convey a meaningful message. Consider a case in which the optimization procedure has decreased both the size of vertex-models and the embedding area. Both are positive changes, however, if the former parameter is reduced more than the later, the changes made to the area usage percentage will be

Figure 5.1: Perfect minor-embedding of $G_{3,3}$



negative.Therefore, this parameter will not be deployed as a global indicator of the quality of the embedding, but it helps us to carry out a deeper analysis in some cases.

5.1.5 Expansion Ratio

Another way of assessing a minor-embedding is by measuring certain attributes of the solution, in regards to similar attributes of the input graph. Informally, the goal is to compare the size of the embedding with that of H. An intuitive way of measuring the size of $H = (V_H, E_H)$ is using $n = |V_H|$ and $m = |E_H|$. Then, since a minor-embedding can be considered as expanding each vertex to a vertex-model, satisfying the same adjacency matrix as the original graph, we can define the two similar measurements of $n' = \sum_{v \in V_H} |\phi(v)|$ and $m' = m = |E_H|$ for the minor-embedding.
The expansion ratio of a minor-embedding is defined as follows:

$$\eta = \frac{n' + m}{n + m}$$

Although m represents the same value for both the graph and its minor-embedding, leaving it out of the equation will lead to losing information about the quality of the solution. For example, a relatively dense graph of n vertices is more likely to have larger vertex-models versus a sparse graph with the same number of vertices. Judging the quality of a minor-embedding, based only on a comparison of n and n', will not take into account the density of the input graph. Moreover, the introduced η has a meaningful range, as it takes a minimum value of 1, which corresponds to the perfect solution in which the size of each vertex-model is exactly 1. Figure 5.1 illustrates a perfect minor-embedding of $H = G_{3,3}$ with embedding area of 9.

5.2 Experiment Details

The details of implementation and experimental parameters of both algorithms are presented in this section. Features of both algorithms are implemented, as presented in earlier chapters. However, the time complexity and the used implementation techniques might be different than what was presented, and we will report the major differences, shortly. The main reason for such differences was to make the implementation simpler and more testable. In a particular case, an additional step has been added to the implementation of the D-Wave algorithm to make sure that the presented feature is exactly carrying out what is expected of it.

5.2.1 The D-Wave Algorithm

The D-Wave algorithm was implemented in MATLAB 2015b, as explained in Section 2.5. The implementations of some graph algorithms, such as A* search, in MatlabBGL library [37] were used in the implementation of the D-Wave algorithm. The D-Wave algorithm requires both the input and the target graphs as inputs. Input graph is set to an arbitrary planar graph H, and the target graph will be a grid graph G of the same dimensions as required by the VRM algorithm for the given input graph. In a number of experiments, the D-Wave algorithm is provided with larger target grid graphs to test if this leads to higher rates of success.
Recall that the D-Wave method runs in two phases. The second phase is in charge of transforming the possibly infeasible output of the first phase into a feasible minor-embedding. The second phase is finished after t consecutive iterations without any further improvement. In our experiments, $t = 10$, exactly as used in [12]. Considering the randomness in the D-Wave algorithm, to get a better understanding of the algorithm's performance we run it more than once, to be exact, 25 times for each instance. The average of the best results, found in each run, will be the reported result of the experiment. The best minor-embedding is the one which minimizes the solution size. The success rate of the algorithm for each instance is trivially the percent of successful runs out of 25.
In [12], the details of the main technique for computing the vertex-models was presented, but this description is ambiguous in parts. The multisource A* procedure, as outlined earlier in Section 2.5, was introduced in [12] to find the root of each vertex-model and the shortest path tree, connecting a vertex to its neighbours. We implemented the primary well-presented parts of this technique, along with the Dijkstra algorithm to make sure the algorithm reaches the desired output, in the sense that it minimizes the distance function and the crossings, as presented in [12].

5.2.2 The VRM algorithm

The VRM algorithm was implemented in MATLAB 2014a. The implementations in MatlabBGL library [37] were also employed in the implementation of the VRM algorithm. The only parameter required to be set for the VRM is the length of the optimization phase. As for the D-Wave algorithm, due to the random behaviour of the VRM in optimization phase, it has been set to run 20 times on any given instance, and the optimization phase of each run has 25 rounds. The reported output sizes are the average of all 20 runs. Having the algorithm outlined in Chapter 4, we continue with further details of implementation.
The dimensions of the target graph are always set to $n + 2$ rows and $m + 2$ columns, where $n = |V_H|$ and $m = |E_H|$. The extra constant 2 rows and columns are simply added because doing so made the experimental implementation easier, by making it possible for us to reuse parts of our earlier implementations of other experimental algorithms. However, as mentioned earlier, the required grid dimensions are $m \times n$. Since our algorithm will optimize the area after the embedding phase, and these extra rows and columns are omitted afterwards, we chose to take this opportunity to simplify our implementation. The same grids, with extra 2 rows and columns, are provided to the D-Wave algorithm as inputs.
Another point concerns the use of priority queues, in the optimization phase of the VRM, to improve the time complexity of this phase as proposed in 3. Our implementation uses a slightly less efficient approach, based on sorting the vector of neighbours. While this modification had no practical effects on our experiments, it made the implementation of the feature much simpler.

5.3 Results and Analysis

In this section, the results of applying the D-Wave and the VRM algorithms to instances of several families of graphs are presented. The families are:

- $k \times k$ Grid graphs, $k \in \mathbb{N}$,
- Random planar graphs, generated by initializing a set of vertices of a fixed size, and adding a fixed number of randomly selected edges to the graph, one-by-one, without violating the planarity.
- Wheel graphs: A wheel graph with n vertices is denoted by W_n in which $n-1$ vertices, forming a cycle, are adjacent to another vertex in a way that $n - 1$ vertices are of degree 3, and the other vertex is of degree $n - 1$;
- Random Apollonian networks: A random Apollonian network can be generated by embedding a triangle in the 2D-plane, and for an arbitrary number of times, adding a vertex in a random face of the embedding and connecting it to the three vertices

bounding that face. The degree distributions of these graphs obey a power law, and the average distance between the vertices is relatively small;

- Random series-parallel graphs: These st-graphs (graphs with two terminals, referred to as source and sink) are generated recursively. A path graph P_2 is the smallest series-parallel graph(sp-graph) with its ends as the sink(t) and the source(s). On two sp-graphs, performing a series composition (merging one's sink with the other's source) or a parallel composition (merging ones source and sink with other's counterparts) generates a new sp-graph;

- A family of maximal sp-graph: Given a sp-graph H_{sp}, if addition of any extra edges to H_{sp} does not preserve its status of being a series-parallel graph, then H_{sp} is a maximal series-parallel graph;

- Several graphs from the literature, plus two of our own construction. The graphs from the literature are Dürer graph, Frucht graph, Errera graph, and Bidiakis cube.

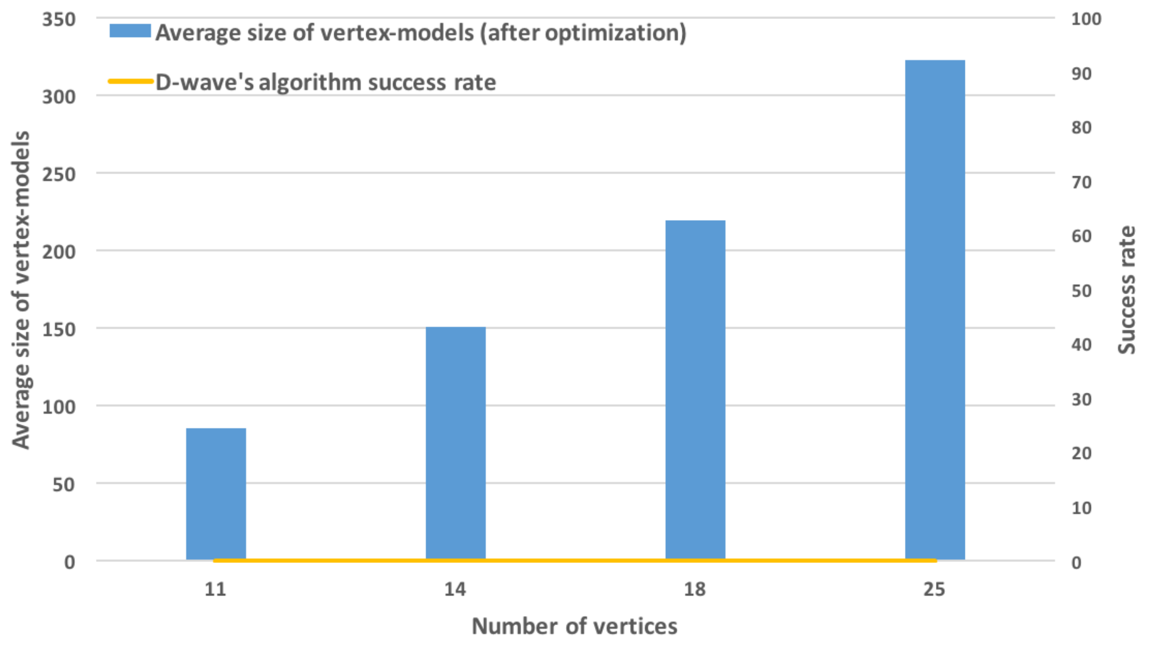
Considering the success rate and the solution size as the high priority parameters, the following figures illustrate the total size of vertex-models generated by the D-Wave algorithm and the VRM, along with the probability of success for the D-Wave algorithm in finding a feasible solution for each instance. Given a planar graph, the VRM algorithm is always successful at finding a minor-embedding, and hence, there is no change of performance to be reported on the plots. We remind the reader that in Chapter 4, the introduced algorithm for finding a feasible solution was referred to as the advanced method, and a less efficient complete alternative, called the simple method, was described to verify the completeness of the VRM. In our experiments, the advanced method was always successful at finding a minor-embedding, and we never had to use the simple method on an instance.

The D-Wave algorithm failed to minor-embed any instance of the random Apollonian networks. Figure 5.2 depicts the average sizes of solutions for each instance of this family, found by the VRM algorithm.

Figure 5.3 illustrates the performance of both algorithms on instances of grid graphs, using the average vertex-model sizes and the success rate of the D-Wave algorithm. When it was successful at finding a feasible embedding of a grid graph, the D-Wave embedding strategy generated a smaller size solution than VRM.

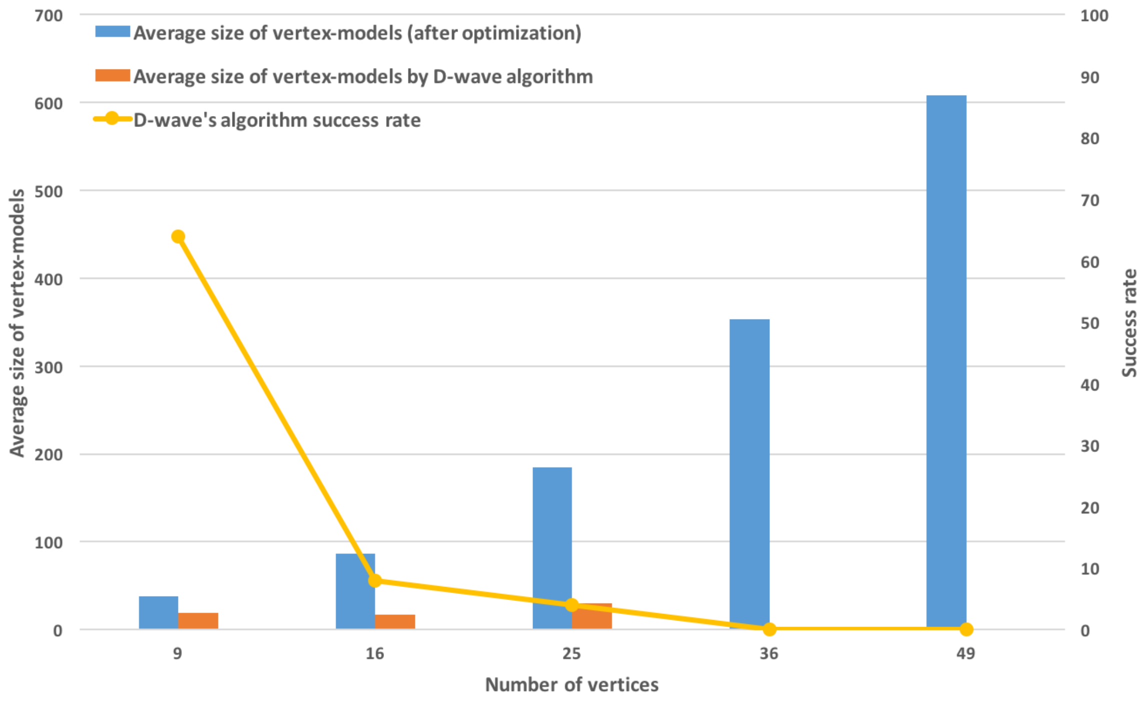
The size of vertex-models reported for each instance of the families of random and random sp-graphs is the average of solution sizes of ten random instances with the same number of vertices and edges. Figure 5.4 indicates the success rate of the D-Wave algorithm in minor-embedding only one instance of the random graphs. As can be seen by comparing Figures 5.5 and 5.3, the differences between average solution sizes of the two algorithms, for each instance of the random sp-graphs, are not as large as those of the grid graphs. Further, the probability of success of the D-Wave algorithm is fairly low, even for very small instances.

Figure 5.2: The D-Wave algorithm fails at minor-embedding any of the random Apollonian networks. The average sizes of solutions, found by the VRM algorithm, for each instance of this family, is depicted in this figure.



In the plots, instances of each family are represented by the number of their vertices. That is because for most graph families used in our experiments, instances with more vertices are larger instances and have more edges as well. Based on our experiments, minor-embedding of instances of a family with more vertices and edges have larger solution sizes, comparing to instances with fewer vertices and edges. However, for random planar graphs, there is no strict correlation between the growth in numbers of edges and vertices. That is why in Figure 5.4, the sizes of vertex-models for random instances with 19 and 21 vertices are less than those instances with 18 vertices. To make sense of this observation, we will borrow our rationalization for presenting the expansion ratio, and compare the sums of edges and vertices of these instances. The count of edges for random graphs with 18, 19, and 21 vertices are 48,23, and 33, respectively. Now, we can observe that the total sizes of vertex-models grow with the sequence of $19 + 23 = 42$, $21 + 33 = 54$, and $18 + 48 = 66$. Same situation applies to instances with 25 and 27 vertices which have 69 and 43 edges, respectively.
The results indicate that as the size of instances grow, the success rate of the D-Wave algorithm decreases rapidly. In fact, for all graph families, the D-Wave algorithm might minor-embed only the very small instances of each, but if it is successful, the results are often better than those of the VRM. That is mainly because the D-Wave algorithm embeds the vertices in a random order, but embeds each root-vertex as close as possible to its already embedded neighbours. This embedding strategy is not guaranteed to find a

Figure 5.3: Average vertex-model sizes of grid graph instances, found by the two algorithms, and the success rate of the D-Wave algorithm.



feasible solution, but its greedy nature might make the results of successful attempts fairly desirable. The relative positioning of vertex-models of a minor-embedding in the grid can be associated with a planar embedding of H. As the number of vertices and edges of H are increased, the probability of generating a solution by the D-Wave algorithm, associated with a planar embedding, is reduced. Table 5.1 represents the results for all instances on which the D-Wave algorithm had a non-zero success rate. Table 5.2 demonstrates the D-Wave results for all instances minor-embedded at least once by the D-Wave algorithm, when the dimensions of the target graph were multiplied by 2. This means a target grid area 4 times bigger than the original area is made available to the D-Wave method to assess its performance and probability of success with more available resources. Comparing tables 5.1 and 5.2 indicates no meaningful difference of obtained results, and all instances in table 5.2 also appear in table 5.1. The comparison confirms that low success rates of the D-Wave algorithm are not caused by lack of available embedding area.

Comparing the expansion ratios for different algorithms is not necessary, since comparison of solution sizes will convey the same message. However, among different families of graphs, growth rates of expansion ratios as instance size increases are interesting to compare. This comparison is conducted only for the VRM, since the other algorithm fails too often as the instance size grows. In all families, the expansion ratio tends to increase as the sum of number of edges and vertices grows larger. However the rate of increase in almost all

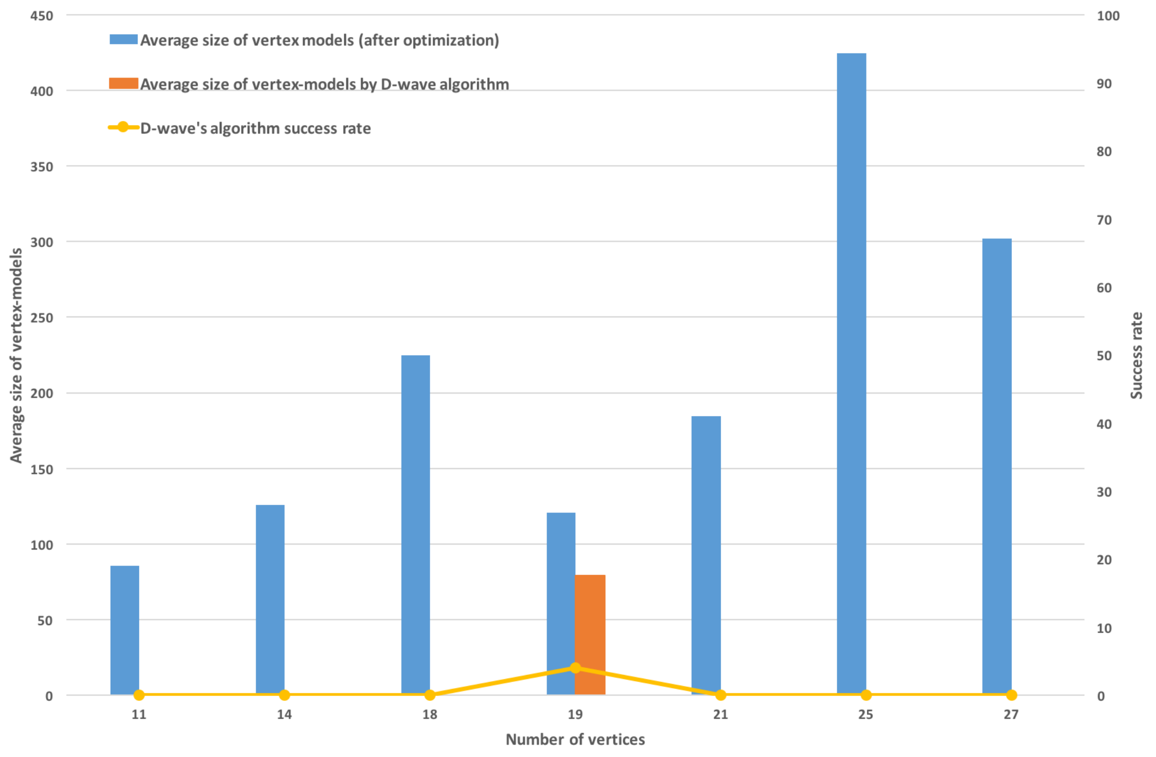
Table 5.1: Comparison of results of the VRM and the D-Wave algorithm, on all instances which were minor-embedded by the D-Wave algorithm at least once. For each instance, target graphs of the same dimensions were made available to each algorithm.

Graph Family	No. of Vertices	Avg. Size of Vertex-Models Before Optimization (VRM)	Avg. Size of Vertex-Models After Optimization (VRM)	Avg. Size of Vertex-Models (D-Wave Algorithm)	Success Rate (D-Wave Algorithm)
Grid	9	49	37.8	19.1	64
Grid	16	141	86.1	17	8
Grid	25	322	184.9	30	4
Wheel	5	24	16.9	14.5	88
Wheel	10	70	55.9	24	4
Maximal Series-Parallel	5	29	16.8	15.05	68
Dürer	12	101	68.4	47.5	8
Frucht	12	83	56.4	56.8	28
Random Series-Parallel	13	79.90	58.8	50.8	26.8
Random Series-Parallel	16	104.5	83.6	55.7	19.2
Random Series-Parallel	19	137.6	108.2	100.5	0.4
Random Planar	19	144.5	120.8	79	4

Table 5.2: Comparison of results of the VRM and the D-Wave algorithm on all the instances which were minor-embedded by the D-Wave algorithm at least once. For each instance, the dimensions of the target graph made available to the D-Wave algorithm are twice as large as those assigned to the VRM.

Graph Family	No. of Vertices	Avg. Size of Vertex-Models Before Optimization (VRM)	Avg. Size of Vertex-Models After Optimization (VRM)	Avg. Size of Vertex-Models (D-Wave Algorithm)	Success Rate (D-Wave Algorithm %)
Grid	9	49	37.8	17.8	68
Grid	16	141	86.1	74	8
Grid	25	322	184.9	74.6	12
Wheel	5	24	16.9	17.4	88
Wheel	10	70	55.9	24	4
Maximal Series-Parallel	5	29	16.8	15.1	72
Dürer	12	101	68.4	49	48
Frucht	12	83	56.4	56.8	28
Random Series-Parallel	13	79.9	58.8	57.7	44.6
Random Series-Parallel	16	104.5	83.6	69.6	38.6
Random Series-Parallel	19	137.6	108.2	133	5.33
Random Planar	19	144.5	120.8	161.9	6.6

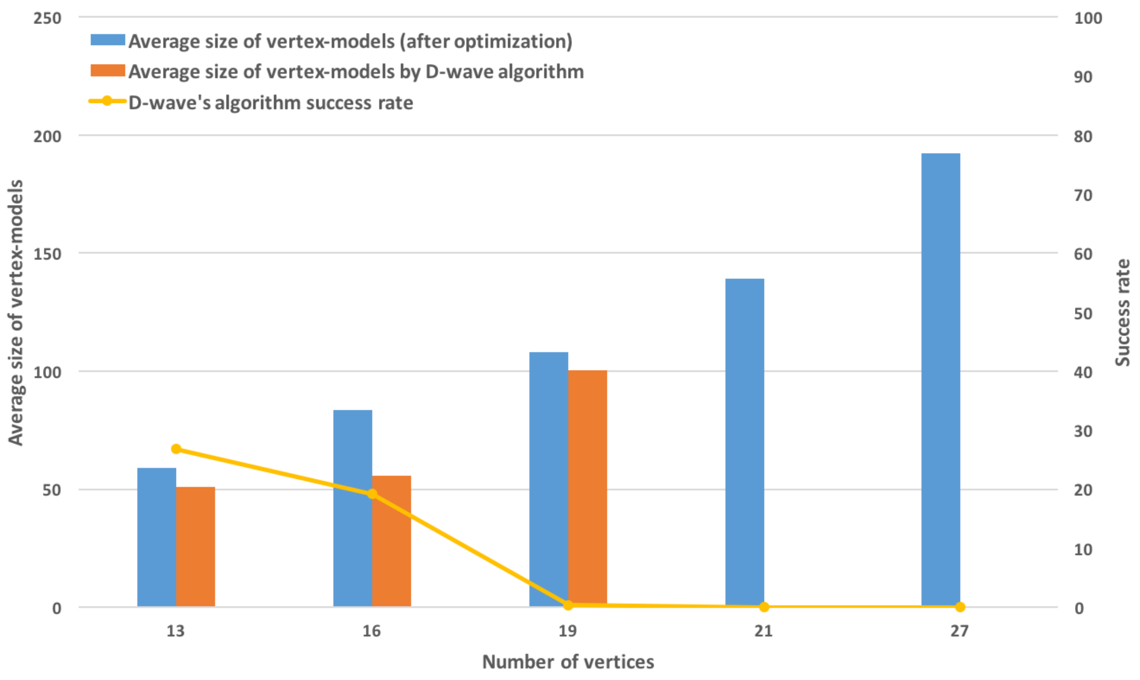
Figure 5.4: Comparison of the results obtained by the two algorithms on random planar graphs. Number of vertices is not enough to demonstrate the complexity of minor-embedding an instance.



families declines as instance size increases. The difference is in the slope of the curves representing these changes. Figure 5.6 shows that the expansion ratios of the random Apollonian networks tend to converge to a constant, fairly quickly, whereas for grid graphs, figure 5.7 shows that the curve is closer to a linear function. Other families follow a pattern similar to that of the random Apollonian networks. While the sum of edge and vertex count of input graph can give us some information about its minor-embedding, studying other attributes of H, such as sequence of degrees and faces, might lead to a more universal relation than the expansion ratio.

In our experiments, we are also interested in performance of the optimization phase of the VRM in reducing the embedding area. For that purpose, embedding area was measured before and after optimization, and the reduction in embedding area was recorded. Table 5.3 demonstrates the average reduction of embedding area for each family of graphs, expressed as a percentage. The average area decreases are between 25 and 64 percent, which illustrates the effectiveness of the proposed technique in reducing embedding area. Figures 5.11, 5.12, 5.13 and 5.14 compare the sizes of vertex-models and embedding areas of instances of the 4 graph families, generated by the VRM algorithm, before and after optimization. The optimization phase was successful at decreasing the embedding area for all instances,

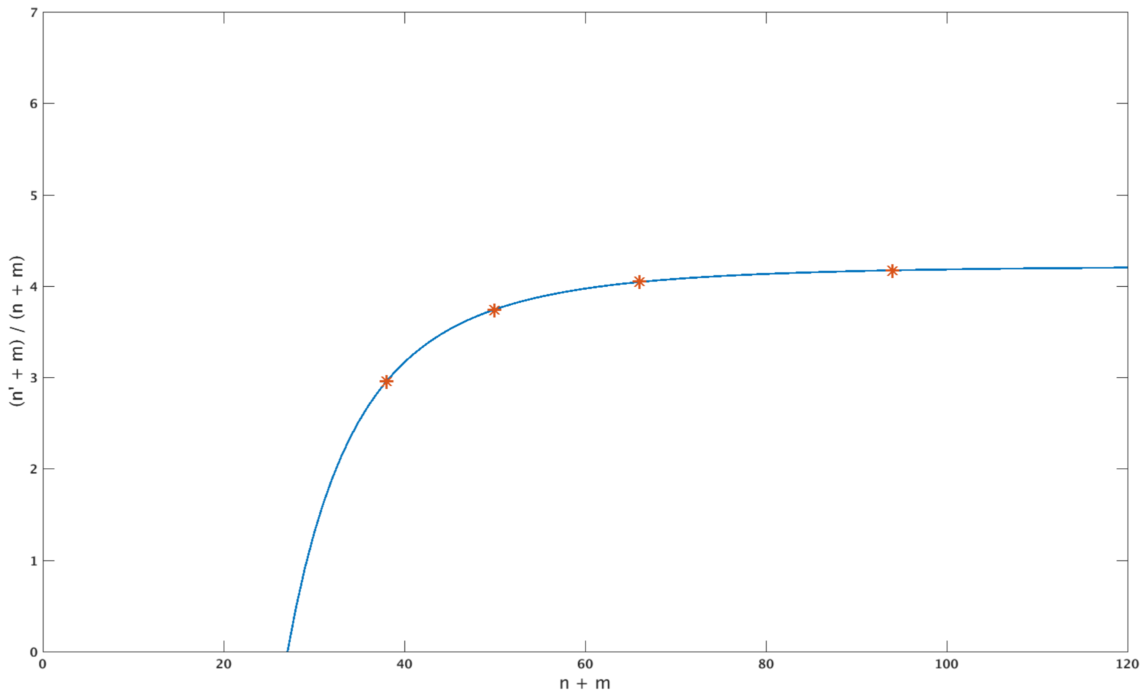
Figure 5.5: Average vertex-model sizes of random sp-graph instances, found by the two algorithms, and the success rate of the D-Wave algorithm.



in some cases up to 80 percent. Figures 5.8, 5.9 and 5.10 compare outputs of the VRM algorithm after its first and second phases for Errera graph, a 7×7 grid, and a maximal sp-graph. Both vertex-model sizes and embedding areas made improved for all instances. Area utilization of the embedding, before and after optimization, has been computed for the three instances to illustrate how the results of each phase can be influenced by the input graph structure. In cases of Errera and the grid graphs, the optimization seems to have made the embedding more compact, and the area utilization changes verify the visual observations. It is clear, both from the figures and the changes in area utilization, that the optimization phase was more successful at increasing the compactness of Errera graph than the other two.

The results are different for instances of maximal sp-graph family, which have the minimum average of area reduction after optimization. The maximal sp-graph instances used in our experiments have a very specific structure. An instance of this family with n vertices has an edge between the source (s) and sink (t) vertices, and all other vertices are only connected to s and t. Hence, any instance of this family has two vertices with degree $n-1$, and $n-2$ vertices with degree 2. This structure will lead to a minor-embedding, similar to the one depicted in figure a of Figure 5.9, which is then optimized to look similar to figure b. The arrangement of vertex-models in figure a leads to greater reductions in the sizes of vertex-models compared to the embedding area, in figure b, which results in a negative change of

Figure 5.6: Growth rate of expansion ratio with $|V| + |E|$ for instances of the random Apollonian network



area utilization. However, we can observe in figure *b* that the number of unused vertices, bounded by the faces of the embedding, have been reduced to almost zero.

Figure 5.7: Growth rate of expansion ratio with $|V| + |E|$ for instances of the grid graph family

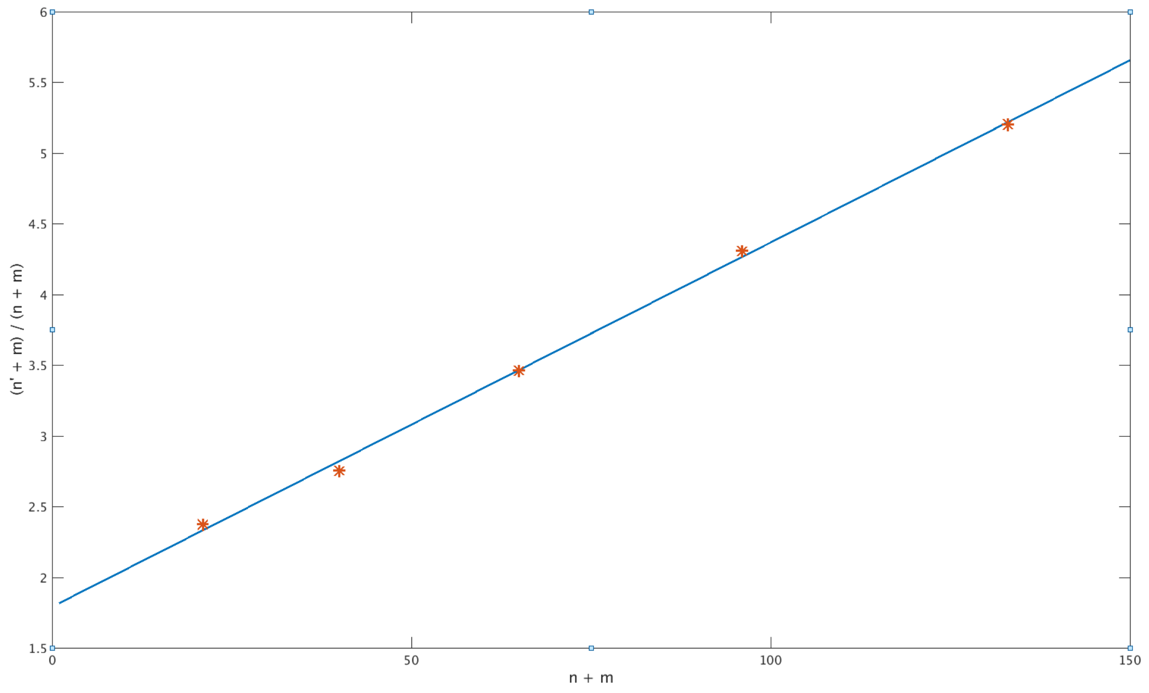


Figure 5.8: Generated minor-embedding by the VRM algorithm, for a 7×7 grid graph, before and after optimization. Area utilization of the embedding equals 0.24 in figure a, and 0.27 in figure b.

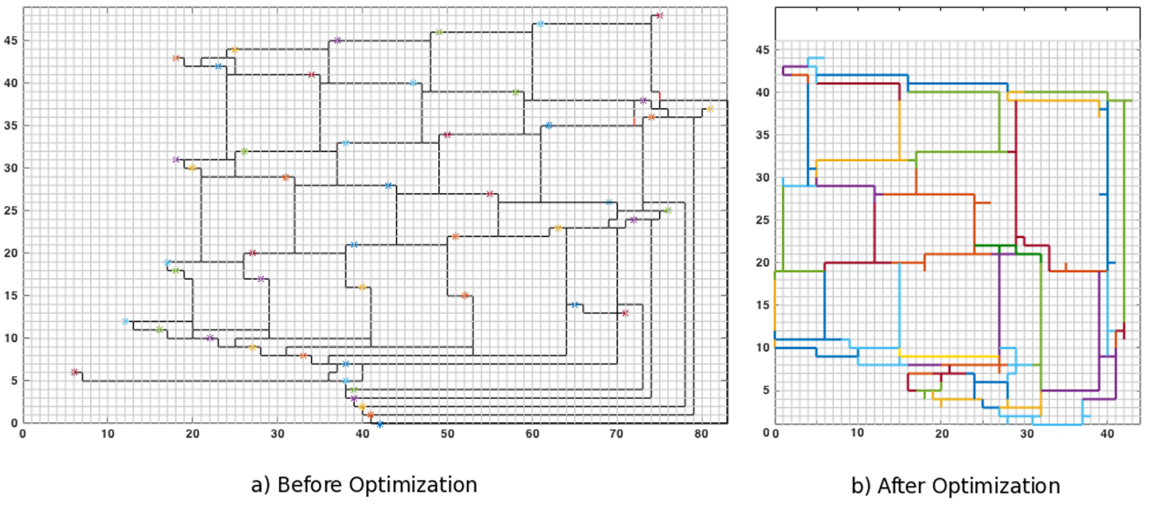


Figure 5.9: Generated minor-embedding by the VRM algorithm for a maximal sp-graph, before and after optimization. Area utilization of the embedding equals 0.25 in figure *a*, and 0.20 in figure *b*. The embedding area has not been reduced enough to result in a positive change of area utilization.

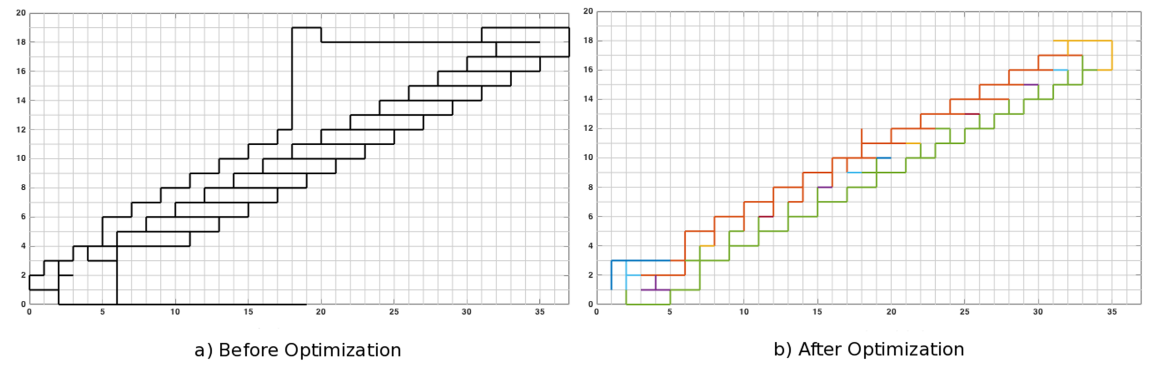


Figure 5.10: Generated minor-embedding by the VRM algorithm for Errera graph, before and after optimization. Area utilization of the embedding equals 0.41 in figure *a*, and 0.83 in figure *b*. The embedding density has been increased more than 100 percents.

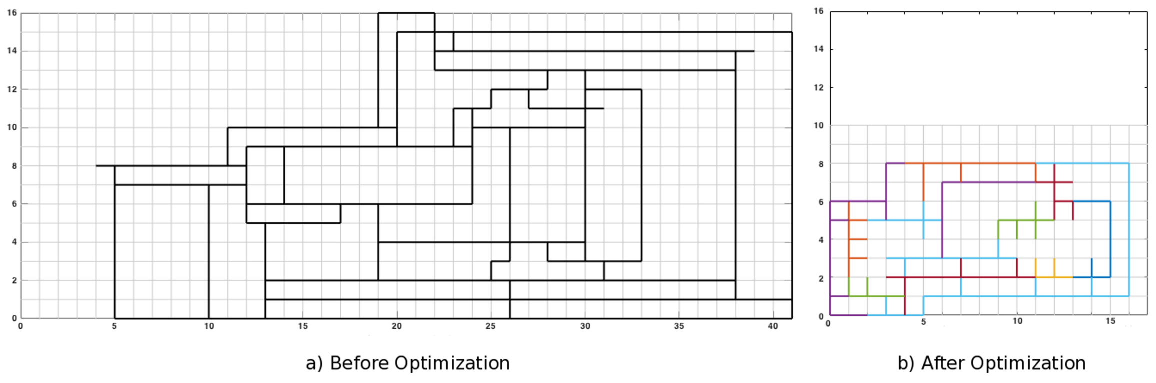


Table 5.3: Average decrease in embedding area, for instances of each graph family, as a result of the optimization phase of the VRM algorithm. The numbers state the ratio of reduction in embedding area after optimization, in comparison to values before optimization, in percentages.

Graph Family	Random Apollonian	Grid	Random Planar	Random SP	Maximal SP	Wheel	Named Graphs
Average Area Decrease After Optimization (Percentage)	**46.21**	**63.43**	**52.33**	**37.35**	**25.99**	**52.14**	**52.87**

Figure 5.11: Average sizes of vertex-models and embedding areas, for random Apollonian networks, before and after the VRM optimization phase

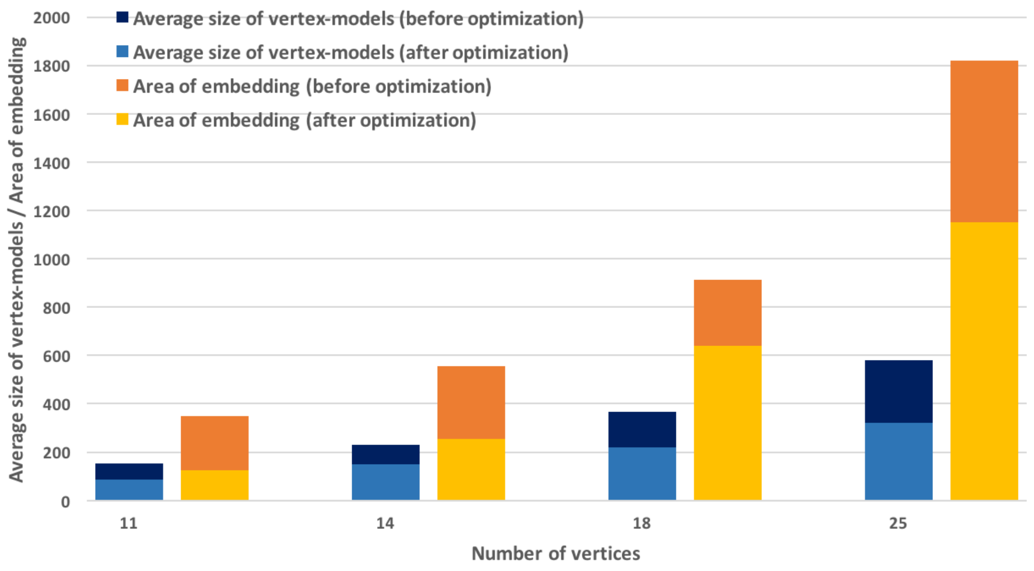


Figure 5.12: Average sizes of vertex-models and embedding areas, for grid graphs, before and after the VRM optimization phase

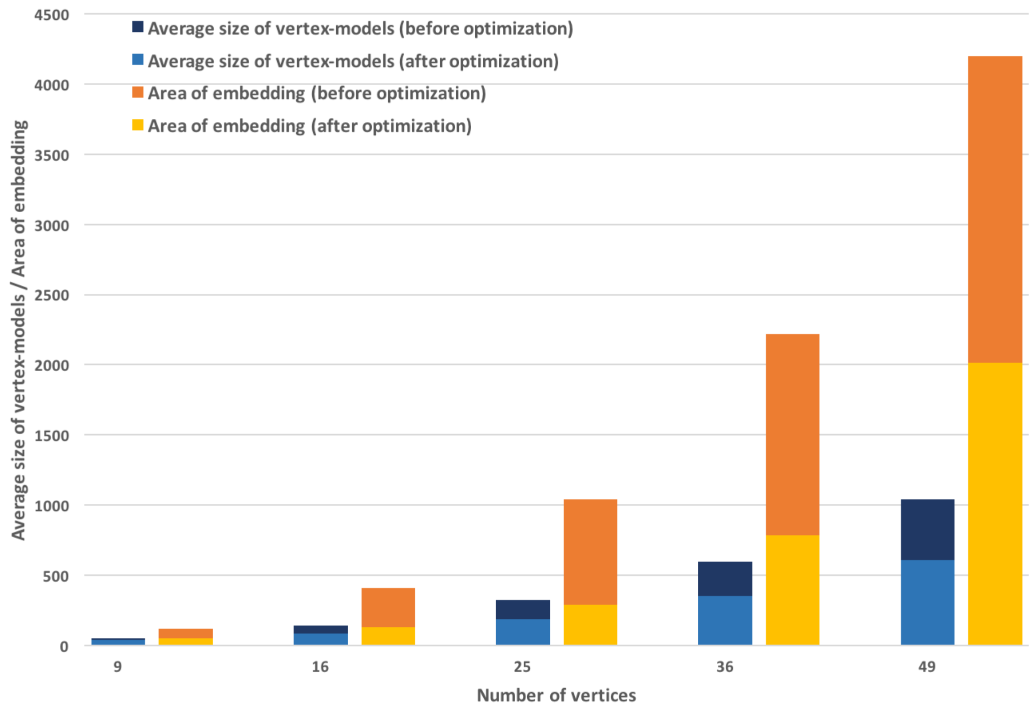


Figure 5.13: Average sizes of vertex-models and embedding areas, for random planar graphs, before and after the VRM optimization phase

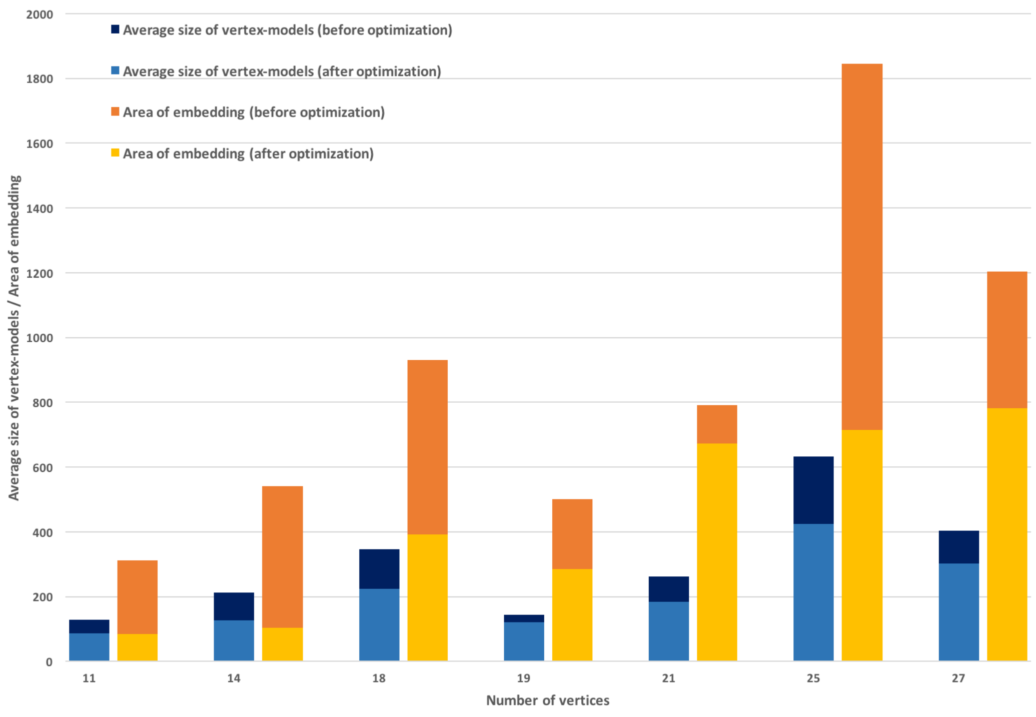
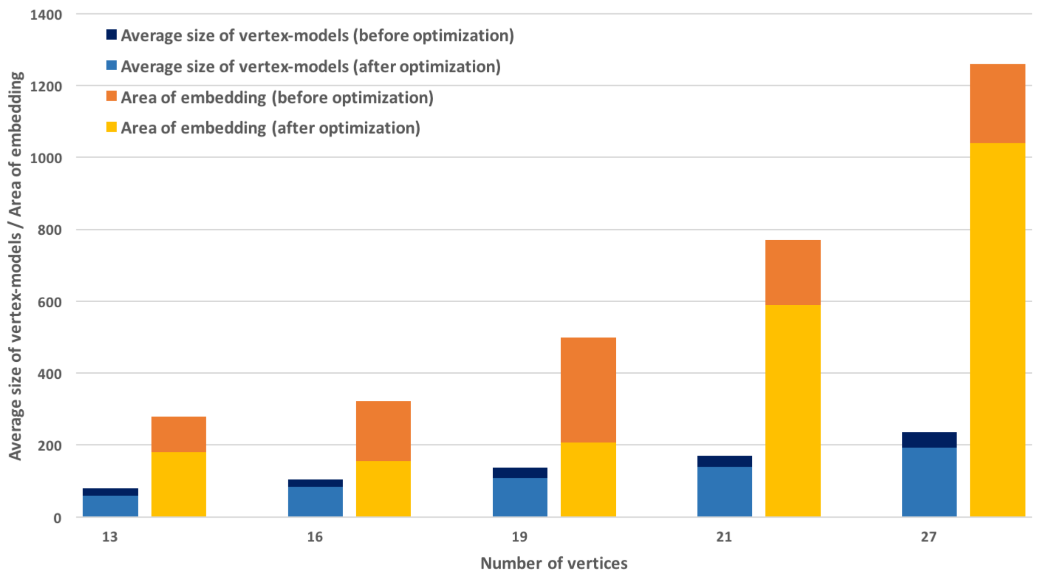


Figure 5.14: Average sizes of vertex-models and embedding areas, for random series-parallel graphs, before and after the VRM optimization phase



Chapter 6

Conclusions and Future Work

In this thesis, the problem of minor-embedding a planar graph in a grid graph was studied, inspired by its application to the process of adiabatic quantum annealing. Our approach was to take advantage of graph embedding and drawing techniques to maximize the success rate of finding a solution, and then applying optimization techniques to increase the embedding quality.

The first algorithm relied on a constrained planar representation of the input graph, inspired by a planar embedding method which generates an embedding by adding edges to a planar representation, while preserving its planarity. Using this minor-embedding algorithm, the embedding area does not depend on the number of edges of the input graph, as a result of applying the introduced technique called, dynamic arm assignment. The proposed procedure has been designed to finalize vertex-model assignments in multiple stages to minimize the size of the minor-embedding, and can arguably be used by a wide range of minor-embedding heuristics. Since the first algorithm operates based on certain preconditions, it is not immediately applicable to the family of all planar graphs. However, our findings and the introduced techniques, from the first algorithm, were used in our second and most effective minor-embedding algorithm.

The visibility representation, which can be generated for any planar graph in linear time, was the main basis of our second algorithm. Dynamic arm assignment was applied to this algorithm as well, and a complementary optimization procedure decreased the embedding size and area. The introduced optimization method is also designed to be adjustable for use in other minor-embedding algorithms. The results of experiments on the D-Wave heuristic and our latest algorithm demonstrated our method's success in minor-embedding all 145 instances from various graph families, while the other algorithm fails at embedding all instances, except for the very small ones. Our optimization technique is also proven to be effective in a clear improvement of the total size of the vertex-models and the embedding area.

The following are suggestions for future work, aimed at extending what was proposed in this thesis:

- As explained earlier, our first algorithm works contingent on being provided with a planar representation of the graph, which satisfies certain conditions. Two strategies were proposed to modify and extend the algorithm to minor-embed any planar graph in a bounded grid graph. The approach we chose to explore resulted in the presented algorithm in Chapter 4. An outline was provided for the other approach, which did not change the algorithm's dependence on the specific planar representation. Instead, it was suggested to compute a set of planar representations, corresponding to subgraphs of the input graph, which satisfy the conditions for minor-embedding their associated subgraphs in the target graph. Finding certain paths between the minor-embedded subgraphs and adding them to the solution will result in a solution for the main minor-embedding problem. While this approach is promising, based on the presented attributes of the BM and the minor-embedding algorithms in Chapter 3, further investigation is required for designing a detailed algorithm which achieves the desired objective or showing that such algorithm does not exist.

- The introduced algorithm in Chapter 4 is designed for minor-embedding a planar graph in a grid graph, without inactive vertices. An inactive grid vertex corresponds to an inactive qubit of the device and has no edges incident to it. The algorithm is based on use of pathfinding techniques in order to be adjustable to compensate for a reasonable number of inactive vertices. Inactive grid vertices cause two main problems for the algorithm. The first problem occurs when the algorithm is embedding the root-vertices, based on the coordinates of the horizontal line segment in the visibility representation. Coordinates of a grid vertex, calculated to be the root of a vertex-model, might belong to an inactive vertex. The second problem is that the A* algorithm might fail to find a shortest path between two root-vertices, due to the missing edges in the grid, especially when the input graph is rather dense.
  We suggest investigation of following approaches towards resolving these problems. For the first problem, selecting an active (not inactive) grid vertex, on the same grid row, may resolve the problem without affecting the vertical order of vertices. The set of grid vertices, from which an alternate is chosen, must be selected carefully so as to preserve the ordering of paths which prevents vertices from being trapped. The second problem can be solved by expanding the grid dimensions by increasing the distances between certain root-vertices. The expansion can be informally explained as inserting rows or columns in the grid graph. The problem of assigning new coordinates to root-vertices, in order to resolve occurrences of the two described problems, with minimum increase in dimensions of the target graph is suggested as the subject of a future work to extend the minor-embedding algorithm.

- The optimization method, explained in Chapter 4, has been designed with focus on the optimization strategy, rather than running time efficiency. Use of additional or different data structures, accompanied by pre-computations, while or before the embedding phase, might reduce the current time complexity. Investigation of the prospects of such improvements are recommended for industrial applications of the algorithm.

- Eliminating a currently existing vertex-model and replacing it with a new one describes the basics of the optimization method, introduced in Chapter 4. Removing the vertex-models, in the way explained, might end up leaving some *extra vertices* in some of the vertex-models. An extra vertex of a vertex-model is one which can be removed from the vertex-model, without making the minor-embedding infeasible. A subset of permanently assigned vertices of a vertex-model, which connect the root-vertex to at least two adjacent root-vertices, may constitute the set of extra vertices of the vertex-model. While re-embedding a vertex v in the optimization phase, permanently assigned vertices of arms of neighbours of v are not removed from their arms, even if after re-embedding v there is no reason for those vertices to be permanently labelled. Such permanently labelled vertices might remain in the arms of a vertex, as extra vertices, even if all of its neighbours which were using those vertices to connect to its root-vertex are re-embedded. Two approaches are suggested to deal with this issue. First solution is to modify the algorithm, in a way that the permanently assigned vertices can be demoted to being a shared vertex if after re-embedding a neighbour they are connecting only two vertex-models together. The second approach is to use a post processing technique to detect and remove extra vertices from vertex-models.

Bibliography

[1] Isolde Adler, Frederic Dorn, Fedor V. Fomin, Ignasi Sau, and Dimitrios M. Thilikos. Faster parameterized algorithms for minor containment. *Theor. Comput. Sci.*, 412(50):7018–7028, November 2011.

[2] Patrizio Angelini, Carla Binucci, Giordano Da Lozzo, Walter Didimo, Luca Grilli, Fabrizio Montecchiani, Maurizio Patrignani, and Ioannis G. Tollis. Algorithms and bounds for drawing non-planar graphs with crossing-free subgraphs. *Computational Geometry*, 50:34 – 48, 2015.

[3] Giuseppe Di Battista, Peter Eades, Roberto Tamassia, and Ioannis G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs.* Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 1998.

[4] Sandeep N. Bhatt and Frank Thomson Leighton. A framework for solving VLSI graph layout problems. *Journal of Computer and System Sciences*, 28(2):300 – 343, 1984.

[5] Daniel Bienstock and Michael A. Langston. Chapter 8 Algorithmic implications of the graph minor theorem. In *Network Models*, volume 7 of *Handbooks in Operations Research and Management Science*, pages 481 – 502. Elsevier, 1995.

[6] Norman Biggs. Topics in Algebraic Graph Theory, by Lowell W. Beineke and Robin J. Wilson (Academic Consultant: Peter J. Cameron), Encyclopedia of Mathematics and Its Applications 102, CUP 2005, 257Pp. *Comb. Probab. Comput.*, 16(1):171–172, January 2007.

[7] Kellogg S. Booth and George S. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using pq-tree algorithms. *Journal of Computer and System Sciences*, 13(3):335 – 379, 1976.

[8] John Boyer. Edge addition planarity suite. *https://github.com/john-boyer-phd/edge-addition-planarity-suite*, 2015.

[9] John M. Boyer. *Graph Drawing: 13th International Symposium, GD 2005, Limerick, Ireland, September 12-14, 2005. Revised Papers*, chapter A New Method for Efficiently Generating Planar Graph Visibility Representations, pages 508–511. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[10] John M. Boyer and Wendy J. Myrvold. On the cutting edge: Simplified O(n) planarity by edge addition. *Journal of Graph Algorithms and Applications*, 8(3):241–273, 2004.

[11] Franz J. Brandenburg, David Eppstein, Andreas Gleißner, Michael T. Goodrich, Kathrin Hanauer, and Josef Reislhuber. *Graph Drawing: 20th International Symposium, GD 2012, Redmond, WA, USA, September 19-21, 2012, Revised Selected Papers*, chapter On the Density of Maximal 1-Planar Graphs, pages 327–338. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[12] Jun Cai, William G. Macready, and Aidan Roy. A practical heuristic for finding graph minors. *arXiv:1406.2741 [quant-ph]*, 2014.

[13] C. C. Cheng, C. A. Duncan, M. T. Goodrich, and S. G. Kobourov. Drawing planar graphs with circular arcs. *Discrete & Computational Geometry*, 25(3):405–418, 2001.

[14] Norishige Chiba, Kazunori Onoguchi, and Takao Nishizeki. Drawing plane graphs nicely. *Acta Informatica*, 22(2):187–201, 1985.

[15] Norishige Chiba, Tadashi Yamanouchi, and Takao Nishizeki. Linear algorithms for convex drawings of planar graphs. *Progress in graph theory*, pages 153–173, 1984.

[16] Vicky Choi. Minor-embedding in adiabatic quantum computation: I. the parameter setting problem. *Quantum Information Processing*, 7(5):193–209, October 2008.

[17] Vicky Choi. Minor-embedding in adiabatic quantum computation: II. minor-universal graph design. *Quantum Information Processing*, 10(3):343–353, June 2011.

[18] M. Chrobak and T. H. Payne. A linear-time algorithm for drawing a planar graph on a grid. *Inf. Process. Lett.*, 54(4):241–246, May 1995.

[19] F. Javier Cobos, J. Carlos Dana, Ferrán Hurtado, Alberto Márquez, and F. Mateos. On a visibility representation of graphs. In *Proceedings of the Symposium on Graph Drawing*, GD '95, pages 152–161, London, UK, UK, 1996. Springer-Verlag.

[20] Sabine Cornelsen and Andreas Karrenbauer. Accelerated bend minimization. *Journal of Graph Algorithms and Applications*, 16(3):635–650, 2012.

[21] Sanjeeb Dash. A note on qubo instances defined on chimera graphs. *arXiv:1306.1202v2 [math.OC]*, 2013.

[22] Matthew Dickerson, David Eppstein, Michael T. Goodrich, and Jeremy Yu Meng. *Graph Drawing: 11th International Symposium, GD 2003 Perugia, Italy, September 21-24, 2003 Revised Papers*, chapter Confluent Drawings: Visualizing Non-planar Diagrams in a Planar Way, pages 1–12. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[23] Walter Didimo. Density of straight-line 1-planar graph drawings. *Information Processing Letters*, 113(7):236 – 240, 2013.

[24] Walter Didimo and Giuseppe Liotta. *Thirty Essays on Geometric Graph Theory*, chapter The Crossing-Angle Resolution in Graph Drawing, pages 167–184. Springer New York, New York, NY, 2013.

[25] Reinhard Diestel. *Graph Theory (Graduate Texts in Mathematics)*. Springer, August 2005.

[26] D. Dolev and H. Trickey. *On Linear Area Embedding of Planar Graphs*. Computer Science Department: Report STAN-CS. Computer Science Department, Stanford University, 1981.

[27] P. Duchet, Y. Hamidoune, M. Las Vergnas, and H. Meyniel. Representing a planar graph by vertical lines joining different levels. *Discrete Mathematics*, 46(3):319 – 321, 1983.

[28] Christian A. Duncan and Michael T. Goodrich. Planar orthogonal and polyline drawing algorithms. In Roberto Tamassia, editor, *Handbook of Graph Drawing and Visualization*, chapter 7, pages 223–247. CRC Press, Providence, Rhode Island, USA, 2014.

[29] Christian A. Duncan and Stephen G. Kobourov. *Graph Drawing: 9th International Symposium, GD 2001 Vienna, Austria, September 23–26, 2001 Revised Papers*, chapter Polar Coordinate Drawing of Planar Graphs with Good Angular Resolution, pages 407–421. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002.

[30] Guy Even, Sudipto Guha, and Baruch Schieber. Improved approximations of crossings in graph drawings and VLSI layout areas. *SIAM Journal on Computing*, 32(1):231–252, 2002.

[31] Shimon Even and Robert Endre Tarjan. Computing an st-numbering. *Theoretical Computer Science*, 2(3):339 – 344, 1976.

[32] Edward Farhi, Jeffrey Goldstone, Sam Gutmann, and Michael Sipser. Quantum computation by adiabatic evolution. *arXiv:quant-ph/0001106*, 2000.

[33] István Fáry. On straight-line representation of planar graphs. *Acta Univ. Szeged. Sect. Sci. Math.*, 11:229–233, 1948.

[34] H. Fraysseix, J. Pach, and R. Pollack. How to draw a planar graph on a grid. *Combinatorica*, 10(1):41–51, 1990.

[35] H. De Fraysseix and P. Rosenstiehl. A depth-first-search characterization of planarity. In Bèla Bollobás, editor, *Graph TheoryProceedings of the Conference on Graph Theory*, volume 62 of *North-Holland Mathematics Studies*, pages 75 – 80. North-Holland, 1982.

[36] Ashim Garg and Roberto Tamassia. *Graph Drawing: Symposium on Graph Drawing, GD '96 Berkeley, California, USA, September 18–20, 1996 Proceedings*, chapter A new minimum cost flow algorithm with applications to graph drawing, pages 201–216. Springer Berlin Heidelberg, Berlin, Heidelberg, 1997.

[37] David Gleich. Matlabbgl library. *http://www.mathworks.com/matlabcentral/fileexchange/10922-matlabbgl*, 2008.

[38] Olivier Goldschmidt and Alexan Takvorian. An efficient graph planarization two-phase heuristic. *Networks*, 24(2):69–73, 1994.

[39] Michael T. Goodrich and Christopher G. Wagner. *Graph Drawing: 6th International Symposium, GD' 98 Montréal, Canada, August 13–15, 1998 Proceedings*, chapter A Framework for Drawing Planar Graphs with Curves and Polylines, pages 153–166. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998.

[40] Carsten Gutwenger and Petra Mutzel. Planar polyline drawings with good angular resolution. In *Graph Drawing (Proc. GD '98), volume 1547 of LNCS*, pages 167–182. Springer-Verlag, 1998.

[41] Seok-Hee Hong and Hiroshi Nagamochi. Two-page book embedding and clustered graph planarity. *http://www-or.amp.i.kyoto-u.ac.jp/members/nag/Technical_report/TR2009-004.pdf*, 2009.

[42] John Hopcroft and Robert Tarjan. Efficient planarity testing. *J. ACM*, 21(4):549–568, October 1974.

[43] M. Jünger and P. Mutzel. Maximum planar subgraphs and nice embeddings: Practical layout tools. *Algorithmica*, 16(1):33–59, 1996.

[44] G. Kant. Drawing planar graphs using the canonical ordering. *Algorithmica*, 16(1):4–32, 1996.

[45] Michael Kaufmann and Dorothea Wagner, editors. *Drawing Graphs: Methods and Models*. Springer-Verlag, London, UK, UK, 2001.

[46] A. Lempel, S. Even, and I. Cederbaum. An algorithm for planarity testing of graphs. In *Theory of Graphs, Int. Symposium (Rome, 1966)*, pages 215–232. Gordon and Breach, New York, NY, USA, 1967.

[47] F Luccio, S Mazzone, and C.K Wong. A note on visibility graphs. *Discrete Mathematics*, 64(2):209 – 219, 1987.

[48] Catherine C McGeoch and Cong Wang. Experimental evaluation of an adiabiatic quantum system for combinatorial optimization. In *Proceedings of the ACM International Conference on Computing Frontiers*, page 23. ACM, 2013.

[49] R. H. J. M. Otten and J. G. Van Wijk. Graph representations in interactive layout design. In *IEEE International Symposium on Circuits and Systems*, pages 914–918. IEEE, 1978.

[50] Kristen L. Pudenz, Tameem Albash, and Daniel A. Lidar. Error-corrected quantum annealing with hundreds of qubits. *Nat Commun*, 5, Feb 2014. Article.

[51] Helen C. Purchase. Which aesthetic has the greatest effect on human understanding? In *Proceedings of the 5th International Symposium on Graph Drawing*, GD '97, pages 248–261, London, UK, UK, 1997. Springer-Verlag.

[52] Ben W. Reichardt. The quantum adiabatic optimization algorithm and local minima. In *Proceedings of the Thirty-sixth Annual ACM Symposium on Theory of Computing*, STOC '04, pages 502–510, New York, NY, USA, 2004. ACM.

[53] Mauricio G.C. Resende and Celso C. Ribeiro. Graph planarization. *AT&T Lab Reports*, 1998.

[54] Neil Robertson and P. D. Seymour. Graph minors. XIII: The disjoint paths problem. *J. Comb. Theory Ser. B*, 63(1):65–110, January 1995.

[55] Walter Schnyder. Embedding planar graphs on the grid. In *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '90, pages 138–148, Philadelphia, PA, USA, 1990. Society for Industrial and Applied Mathematics.

[56] T. Stutzle and H. Hoos. Max-min ant system and local search for the traveling salesman problem. In *Evolutionary Computation, 1997., IEEE International Conference on*, pages 309–314, Apr 1997.

[57] Yoshiyasu Takefuji and Kuo-Chun Lee. A near-optimum parallel planarization algorithm. *Science*, 245(4923):1221–1223, 1989.

[58] Roberto Tamassia. On embedding a graph in the grid with the minimum number of bends. *SIAM Journal on Computing*, 16(3):421–444, 1987.

[59] Roberto Tamassia and Ioannis G Tollis. A unified approach to visibility representations of planar graphs. *Discrete & Computational Geometry*, 1(4):321–341, 1986.

[60] Carsten Thomassen. *Plane representations of graphs.* Danmarks Tekniske Højskole. Matematisk Institut, 1982.

[61] William Thomas Tutte. How to draw a graph. *Proc. London Math. Soc*, 13(3):743–768, 1963.

[62] Eric W Weisstein. Grid graph. *MathWorld–A Wolfram Web Resource*, 2014.

[63] Stephen K. Wismath. Characterizing bar line-of-sight graphs. In *Proceedings of the First Annual Symposium on Computational Geometry*, SCG '85, pages 147–152, New York, NY, USA, 1985. ACM.

[64] Arman Zaribafiyan, Dominic J.J. Marchand, and Seyed Saeed Changiz Rezaei. Systematic and deterministic graph-minor embedding for cartesian products of graphs. *arXiv:1602.04274 [cs.DM]*, 2016.

Appendix A

Supplementary Material

The accompanying compressed file contains a Matlab implementation of the VRM algorithm and a read me file, explaining all necessary steps and requirements for using the software. File name: The VRM Suite.zip