An Alternative Proof for the NP-completeness of the Grid Subgraph Problem

M.Sc. Thesis

by Dimitris A. Chatzidimitriou

supervised by Professor Dimitrios M. Thilikos, UoA

Graduate Program in Logic, Algorithms and Computation

National and Kapodistrian University of Athens

Department of Mathematics

Athens, October 2016

Η παρούσα Διπλωματική Εργασία
εκπονήθηκε στα πλαίσια των σπουδών
για την απόκτηση του
Μεταπτυχιακού Διπλώματος Ειδίκευσης
στη
Λογική και Θεωρία Αλγορίθμων και Υπολογισμού
που απονέμει το
Τμήμα Μαθηματικών
του
Εθνικού και Καποδιστριακού Πανεπιστημίου Αθηνών

Εγκρίθηκε την ..... από Εξεταστική Επιτροπή
αποτελούμενη από τους:

Ονοματεπώνυμο	**Βαθμίδα**	**Υπογραφή**
1.	.....	.....
2.	.....	.....
3.	.....	.....

Abstract

In the field of Graph Drawing, there is great interest for results regarding the embedding of a given graph on a grid, mainly due to the applications on the VLSI circuit design. Moreover, determining whether a graph accepts a unit-length embedding, i.e., a matching of its vertices and edges to vertices and edges of a large enough grid, is the same as asking whether the graph is a subgraph of that grid.

We consider the Grid Subgraph problem, in which given a planar (not necessarily connected) graph G, we need to determine if G is isomorphic to a subgraph of a large enough grid. We prove that this problem is NP-complete by employing simple and intuitive gadgets to perform a reduction from a SAT-variant. In addition we prove that a special case of that problem, the $(k \times k)$-Grid Subgraph problem, in which the size of the grid is given in the input, is also NP-complete.

Περίληψη

Στον τομέα της Γραφικής Αναπαράστασης Γραφημάτων υπάρχει μεγάλο ενδιαφέρον για αποτελέσματα σχετικά με την εμβάπτιση ενός δοθέντος γραφήματος πάνω σε μία σχάρα, κυρίως λόγω των εφαρμογών στο σχεδιασμό κυκλωμάτων VLSI. Πιο συγκεκριμένα, το ερώτημα αν ένα γράφημα επιδέχεται εμβάπτιση μοναδιαίου μήκους, δηλαδή μια αντιστοίχιση των κορυφών και των ακμών του γραφήματος σε κορυφές και ακμές μιας αρκετά μεγάλης σχάρας, ταυτίζεται με το ερώτημα αν το γράφημα είναι υπογράφημα της συγκεκριμένης σχάρας.

Θεωρούμε το πρόβλημα **Υπογράφημα Σχάρας**, στο οποίο δοθέντος ενός επίπεδου (όχι απαραίτητα συνεκτικού) γραφήματος G, καλούμαστε να αποφανθούμε αν το G είναι ισόμορφο με κάποιο υπογράφημα μιας αρκετά μεγάλης σχάρας. Αποδεικνύουμε ότι το πρόβλημα αυτό είναι NP-πλήρες χρησιμοποιώντας απλά και διαισθητικά για να ανάγουμε σε αυτό μία παραλλαγή του προβήματος SAT (ικανοποίησης λογικής φόρμουλας). Προς αυτό αποδεικνύουμε ότι και η ειδική περίπτωση του προβλήματος στην οποία το μέγεθος της σχάρας είναι προκαθορισμένο, γνωστό και ως το πρόβλημα **Υπογράφημα** $(k \times k)$**-Σχάρας**, είναι επίσης NP-πλήρες.

Ευχαριστίες

Πρώτα και κύρια θα ήθελα να ευχαριστήσω την οικογένεια μου και τους φίλους μου για την αγάπη τους και την αμέριστη και αδιαπραγμάτευτη στήριξή τους σε κάθε μου εγχείρημα. Επίσης, τη Σοφία, τον Αντώνη και το Νίκο που μου γνώρισαν την ομορφιά των Μαθηματικών. Τέλος, τον επιβλέποντά μου κο Θηλυκό και τα υπόλοιπα μέλη της Graphka για τη συνεχή βοήθεια και καθοδήγηση.

Contents

Chapter 1

An introduction to Graph Theory

In this chapter we introduce a few basic concepts and definitions of Graph Theory.

Basic graph definitions. A *graph* G is defined as an ordered pair (V, E) such that $E \subseteq V \times V$ and $V \cap E = \emptyset$. The set V (also denoted $V(G)$) is called the graph's *vertex set* while the set E (also denoted $E(G)$) is called the *edge set*. Let $e = \{u, v\} \in E(G)$, then the vertices u and v will be called the *endpoints* of e. Given a graph G and a vertex $v \in V(G)$, the edges in $E(G)$ that contain v as an endpoint will be called *incident to* v. Also, the *degree of* v *in* G, denoted by $d_G(v)$ (or simply $d(v)$ if G is implied), is the number of edges incident to v. A graph G' is *a subgraph of* G (denoted by $G' \subseteq G$) if $V(G') \subseteq V(G)$ and $E(G') \subseteq E(G)$.

In this thesis we only deal with *simple graphs*, i.e., graphs that don't contain multiple edges connecting the same vertices or loops (an edge whose endpoints are the same).

There are many ways to represent a graph. The most visual is by drawing a dot for each member of its vertex set and an arc connecting two dots for each corresponding member of the edge set. A graph G is called *planar* if it can be embedded on the plane, i.e., it can be drawn on the plane in such a way that its edges do not intersect apart from any common endpoints.

We consider $\mathbb{N}$ to be the set of non-negative integers and for each positive integer k we denote by $[k]$ the set $\{1, 2, \ldots, k\}$. In Set Theory, a *partition* of a set S is a collection $\{S_1, \ldots, S_k\}$, $k \in \mathbb{N}$, of subsets of S such that: $\bigcup_{i=1}^{k} S_i = S$ and $S_i \cap S_j = \emptyset$, $\forall i, j \in [k]$ with $i \neq j$. A graph is called *bipartite* if its vertices can be partitioned into two sets such that every edge of the graph has exactly one endpoint in each set.

A key concept in Graph Theory is that of the isomorphism. Essentially, it is a way to formulate when two distinct graphs are considered to be the same. We say that two graphs G, H are *isomorphic* (and denote it by $G \simeq H$) if there is a bijection $\phi : V(G) \to V(H)$ such that for every $u, v \in V(G)$ it holds that $\{u, v\} \in E(G)$ if and only if $\{\phi(u), \phi(v)\} \in E(H)$.

A *walk* in a graph is an alternating sequence of vertices and edges that begins and ends with a vertex, and each edge in the walk precedes and is preceded by its two endpoints, respectively. A walk in which all edges are distinct is called a *trail*. Furthermore, a trail in which all vertices are distinct is called a *path* and a trail with the same endpoints and all internal vertices distinct is called a *cycle*. A graph is *connected* if for every pair of its vertices there is a path in the graph connecting them. A graph is called a *forest* if it contains no cycles. A connected, acyclic graph is called a *tree* and its vertices of degree one are called *leaves*. It is easy to see that in a tree there is a unique path between any two of its vertices. A *rooted tree* is a tree with a specific vertex designated to be the root. In a rooted tree, the *parent* of a vertex is the next vertex in the unique path towards the root. Obviously, every vertex apart from the root has a parent and it is unique. The *children* of a vertex are the vertices that have that vertex as a parent. A *binary tree* is a rooted tree in which each vertex has at most two children.

Common graph classes and operations. Let us now consider some of the most common classes of graphs. We define the following classes for every positive integers r and l.

- The graph $P_r = (\{u_1, \ldots, u_{r+1}\}, \{\{u_1, u_2\}, \ldots, \{u_r, u_{r+1}\}\})$ is called a *path* of length r and the vertices u_1, u_{r+1} are called its *endpoints*.

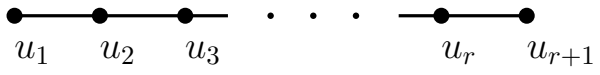

Figure 1.1: The graph P_r.

- The graph $C_r = (\{u_1, \ldots, u_r\}, \{\{u_1, u_2\}, \ldots, \{u_{r-1}, u_r\}, \{u_r, u_1\}\})$ is called a *cycle* of length r.

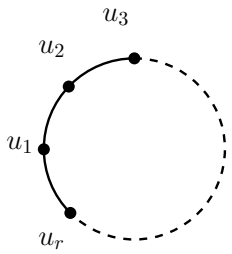

Figure 1.2: The graph C_r.

- The graph $K_r = (\{u_1, \ldots, u_r\}, \{\{u_i, u_j\} \mid \forall i, j \in \{1, \ldots, r\}, \text{ such that } i \neq j\})$ is called a *clique* of size r.

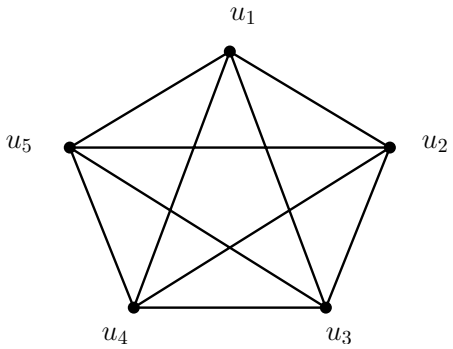

Figure 1.3: The graph K_5.

- Let $V_k = \{1, \ldots, k\}$ and $V_l = \{1, \ldots, l\}$. The graph $(V_k \times V_l, \{\{(x_i, y_i), (x_j, y_j)\} \mid \forall (x_i, y_i), (x_j, y_j) \in V_k \times V_l, \text{ such that } |x_i - x_j| + |y_i - y_j| = 1\})$ is called the $(k \times l)$-*grid*. A graph G is called a *grid* if it is isomorphic to a $(k \times k)$-grid for some $k \geq 1$. A grid is a planar bipartite graph.

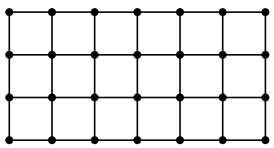

Figure 1.4: The (4×7)-grid.

On a given graph G, we can define the following operations:

1. vertex removal,
2. edge removal,
3. vertex deletion, and
4. edge contraction.

Of these four operations, the first two are self-explanatory and for a graph G with a vertex $v \in V(G)$ and an edge $e \in E(G)$, they are denoted by $G \setminus v$ and $G \setminus e$, respectively. For the third, let G be a graph, v a vertex of degree 2 and a, b its two neighbours. The *deletion of v from G* results in the graph $G' = \{V(G \setminus v), E(G) \cup \{a, b\} \setminus \{\{v, a\}, \{v, b\}\}\}$, which is denoted by $G - v$. For the fourth, let G be a graph with at least one edge and $e = \{u, v\} \in E(G)$. The *contraction of e from G* results in the graph $G' = \{V(G) \cup \{w\} \setminus \{u, v\}, E(G) \cup \{\{w, x\} \mid \forall x \in N_{G \setminus e}(u) \cup N_{G \setminus e}(v)\} \setminus \{e\}\}$, which is denoted by G/e.

We say that a graph G' is *a minor* of G, and denote it by $G' \leq G$, if G' can be obtained from G after applying any number of the above four operations in any order. Also, if G' can be obtained from G by applying only the first three operations, we say that G' is an *immersion* of G, and denote it by $G' \leq_{im} G$. Obviously, if G' can be obtained from G by applying only the first two operations, then G' is a subgraph of G.

Given two graphs G_1 and G_2, we define the *union graph* of G_1, G_2 to be the graph $\{V(G_1) \cup V(G_2), E(G_1) \cup E(G_2)\}$ and denote it by $G_1 \cup G_2$. In the special case where $V(G_1) \cap V(G_2) = \emptyset$, we will call this operation the *disjoint union* of G_1, G_2 and denote it by $G_1 + G_2$.

Graph drawing. An *orthogonal grid embedding* is a mapping that maps the vertices of a (planar) graph G into grid points and the edges into interior disjoint path segments of the grid. If all the paths of the mapping have length of one, we call the embedding a *unit-length embedding*. We refer to such set of points and unit-length segments as a grid drawing. For simplicity, we will not distinguish between the mapping and the resulting drawing of the graph. We consider two embeddings to be the same if they correspond to the same drawing up to rotation, translation, and reflection. In an orthogonal grid embedding, two adjacent edges form a *bend* if they are

drawn perpendicularly. A path is called *straight* if it contains no bends, otherwise it is called *bending*.

A careful study of the above definitions reveals the connection between the two terminologies, namely that a graph accepts an orthogonal grid embedding if and only if it is an immersion of some grid. Similarly, a graph accepts a unit-length embedding if and only if it is a subgraph of some grid. A graph with the last property is sometimes also called a *partial grid* in bibliography (see for example [4]).

Chapter 2

An introduction to Theory of Computational Complexity

Intuitively, an algorithm is a set of instructions or commands that one needs to follow in order to solve a specific problem for many possible inputs, like a recipe. The first question one will naturally ask when tackling a problem is whether the problem can be solved or not. But what does it mean for a problem to be solvable?

There was a great effort over the first half of the 20^{th} century by distinguished mathematicians such as Alonzo Church, Alan Turing, Stephen Kleene, Emil Post, and others, to rigidly define the concept of "solvable", or "computable". Many different models were proposed independently but nearly all of them turned out to be equivalent. Arguably the simplest and most elegant of those is the Turing Machine. A *deterministic Turing Machine* (or TM for short) accepts an infinite tape as input and consists of a head which reads a single cell of the tape at a time and the cell may contain just one of two symbols, for example $\{0, 1\}$, or be empty. Then, according to a predefined set of instructions, the machine can write a (new) symbol on that cell and move to the next cell in either direction to continue its function, or halt. Although at first glance it may appear that such a crude and simple computing machine can not be very powerful, on the contrary its computing capabilities are equivalent to the best modern computer! Of course its computation is extremely slow, but the important thing is that any problem that can be solved by the latter, can be solved by a TM as well.

Measuring the complexity. Having determined the computability of a certain problem, the next natural step would be to focus on its complexity. The measures

we usually employ to that end are the running time of the machine and the required memory, representing the *time complexity* and the *space complexity*, respectively. At this point we should elaborate a little on what we mean exactly by "running time". It is customary to count a machine's running time in *steps*, where in each step the machine performs an elementary operation. As we hinted before, we have built a problem-solving machine with certain instructions, so that it is able to solve any *instance* of the problem (i.e., any possible data-input for this specific problem), not just one. For example, if we designed a TM that computes the size of the largest cycle in a graph, the machine could perform its computation on any possible graph given as an input. Evidently, the machine's running time might differ greatly between inputs of the same size. Therefore, in order to measure its running time, we can use the fastest computation over all instances (best-case scenario), the slowest computation (worst-case scenario), or the expected time (average-case scenario). Of those, the most common and the one we are going to employ, is the worst-case analysis. One last thing we need to consider is that the running time should not be an absolute measure, in the sense that it is normal for larger inputs to need more time than smaller ones. Hence, we measure the running time with respect to the size of the input, and since constants don't matter much when the size of the input tends to infinity, we will use the $\mathcal{O}$-*notation*. Saying, for example, that a TM runs in linear time (or, equivalently, in time $\mathcal{O}(n)$), means that on an input of size n, there is a constant c, such that the machine will stop after at most $c \cdot n$ steps (see Figure 2.1).

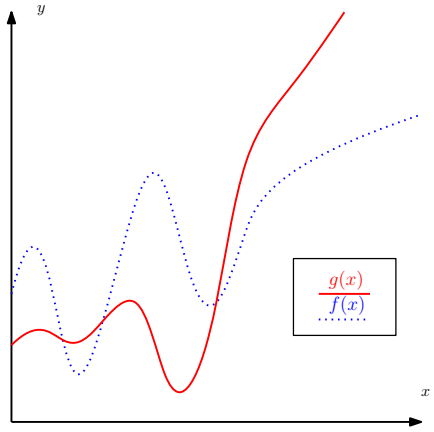

Figure 2.1: $f(x) = \mathcal{O}(g(x))$ since $f(x) \leq g(x)$ as x tends to infinity.

The $\mathcal{O}$-notation is very useful in this setting, since for very big values of n, say of 1000 or 2^{1000} digits, it doesn't make much difference if a TM's running time is $(3n^2 + 1)$ or $(2n^2 + 500)$, but it makes a huge difference if it is $(n^3/100)$. All that matters is that the running time is quadratic to the size of the input instead of cubic, i.e., $\mathcal{O}(n^2)$ instead of $\mathcal{O}(n^3)$. But up to now we have only talked about the running time of a specific TM, while there can be many (and in fact infinitely many) TMs that solve the same problem. So what is the complexity of a problem? To answer this question, we are going to use a min-max parameter: the *complexity of a problem* is the worst-running time over all inputs, of the fastest TM that solves the problem.

Complexity classes. Now that we have defined properly complexity measures for the problems, we can start to categorize them. For simplicity we will only deal with *decision classes*, i.e., classes that contain problems whose output is only yes or no and not, for example, the value of a function. The class of all such problems that can be solved in polynomial time by a deterministic Turing Machine is P. Similarly, the class of all problems that can be solved in exponential time by a deterministic TM is EXP. Obviously, $\mathsf{P} \subseteq \mathsf{EXP}$ and since there are problems that belong in $\mathsf{EXP} \setminus \mathsf{P}$, we conclude that $\mathsf{P} \subsetneq \mathsf{EXP}$. Up until now we only talked about deterministic Turing Machines, but what would non-determinism mean for a TM? A deterministic TM was defined as having a *transition function* which, depending on the state of the machine and the symbol that is currently reading, "tells" the machine what to do next, i.e., which symbol to write on the current cell (if any) and whether to move on the previous or next cell, or to halt. In a *non-deterministic TM* (or NTM for short), the transition function is replaced by a *transition relation*, so instead of having a unique response to every possible *configuration* (i.e., a combination of machine state - tape symbol), it may have more than one! One way to understand that is by imagining that in each step the machine can perform many actions simultaneously. It is like being inside a labyrinth and upon arriving at a crossroads, the TM doesn't need to follow just one of the paths ahead but can follow all of them simultaneously or it can "guess" the correct path in advance and simply follow that. Non-deterministic complexity classes are defined analogously with their deterministic counterparts. For example, the class NP contains the problems that can be solved in polynomial time by an NTM. The exotic notion of non-determinism might seem extremely powerful at first, and one would think that it gives Turing Machines much better computing capabilities, but that is yet to be proven. It is easy to see that these three complexity classes have the

following relation (Figure 2.2):

$$\mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{EXP},$$

but none of the two inclusions has been proven to be proper yet, although we know that at least one of them must be proper (since $\mathsf{P} \subsetneq \mathsf{EXP}$).

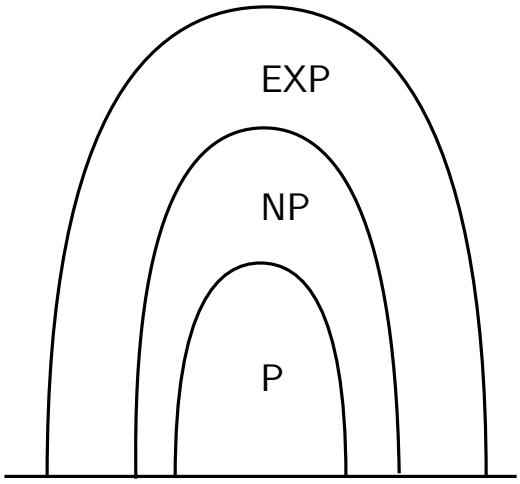

Figure 2.2: The relation between P, NP, and EXP.

Reductions and NP-completeness. An equivalent definition for NP is the following: NP is the class of problems that have polynomial size certificates. A *certificate* for a yes-instance is a string that can be checked deterministically in polynomial time and verifies that this is indeed a yes-instance. Consider for example the Hamilton Path problem, in which given a graph, we are asked to determine whether it contains a path that includes every vertex of the graph. By the latter definition, this problem is obviously in NP since its certificate is the Hamilton path itself, which can be checked in polynomial (or linear to be exact) time if it is indeed a subgraph of the input graph, but we don't know if it is in P (and we believe it is not) since no one seems to be able to design a polynomial-time algorithm to find it.

The complexity classes is of course a useful tool for categorizing problems. It would be of little use though if we didn't have a means to compare two problems directly. For that we define the concept of *reduction* and we will say that a problem A is at least as hard as B (and denote it by $A \leq B$) if B reduces to A, that is if there

exists a polynomially computable transformation f from instances of B to instances of A, such that for every instance x of B, x is a yes-instance of B if and only if $f(x)$ is a yes-instance of A. For simplicity, we shall call f *the reduction from* B *to* A. Intuitively, the existence of a reduction means that if we could solve problem A, we would transform instances of B to equivalent instances of A, solve them, and in doing so we would have solved B. That is why we say that A is easier than B.

As we mentioned earlier, we know of problems that belong in P and of problems that belong in NP, but we know of no problems that belong in $\mathsf{NP} \setminus \mathsf{P}$. Having said that, one can observe that not all problems in NP have the same difficulty. Some appear to be harder than others. In fact, there are some to which we can reduce any other problem in NP! We call such problems NP-*hard*, and one way to show that $\mathsf{P} = \mathsf{NP}$ would be to solve one of those problems in polynomial time. So, a problem being NP-hard means that it is at least as difficult as any other problem in NP. But of course that is true for any problem outside NP as well. We will call the NP-hard problems that actually belong in NP, NP-*complete*. Interestingly enough, the majority of interesting problems that are not obviously extremely difficult, end up to be NP-complete. The easiest way to show that a problem is NP-complete (and in fact the one that we will use in our proofs) is by the following procedure.

1. Show that the problem has polynomially verifiable certificates.
2. Present a reduction from a known NP-complete problem to the problem at hand.
3. Prove that the reduction creates equivalent instances.
4. Argue that the reduction can be constructed in polynomial time.

Chapter 3

Preliminaries

In this chapter we present the main problem, our approach, and some known results and a lemma that we will need for our proof.

The problem we examine in this thesis is the following.

Grid Subgraph
Input: A graph G
Question: Is G a subgraph of some grid?

And we will prove the following theorem.

Theorem 3.1. *The problem* Grid Subgraph *is* NP*-complete.*

In fact by tweaking our gadgets a little one can prove something more general, namely that the problem Grid Subgraph is NP-complete even when the input graph is restricted to be a tree. Unfortunately, to do that the gadgets and the proof become too complex to be valuable since this result follows directly from [7].

First we will prove that if we restrict the grid in question to a certain size the problem can easily be shown to be NP-complete. The formulation of the problem is the following:

($k \times k$)-Grid Subgraph
Input: A graph G and a positive integer k.
Question: Is G a subgraph of the $(k \times k)$-grid?

Theorem 3.2. *The problem* $(k \times k)$-Grid Subgraph *is* NP*-complete.*

Proof. First we show that $(k \times k)$-Grid Subgraph is in NP. Indeed, a certificate is a unit-length embedding of the graph, which can easily be checked in polynomial time for being legitimate. To prove that it is NP-complete, we will describe a reduction from the famous Partition problem. In this, we need to decide whether a given set S of positive integers can be partitioned into two subsets S_1 and S_2 such that the sum of the numbers in S_1 equals the sum of the numbers in S_2.

Let $S = \{s_1, \ldots, s_n\}$, for some $n \in \mathbb{N}$, be an instance of the Partition problem. From S we will construct a graph G_S such that S is a yes-instance for Partition if and only if G_S is a yes-instance for $(k \times k)$-Grid Subgraph, for some $k \in \mathbb{N}$. Let $m = \Sigma_{i=1}^{n} s_i$. If m is odd or $m = 0$, then let $G_S = K_3$, otherwise the construction is the following.

- Construct the $(k \times k)$-grid, for $k = (m+1)/2$.
- Remove from the grid the vertices $(i, 1)$ and $(1, j)$ for every $i, j \in \{3, \ldots, k\}$ resulting in the graph G_0 (see also Figure 3.1). Notice that these vertices are exactly m.
- For each $i \in [n]$, perform the operation $G_{i-1} + Ps_i$. In other words, for each $s_i \in S$ we add to the above partial grid a path graph of length s_i.
- Let $G_S = G_n$.

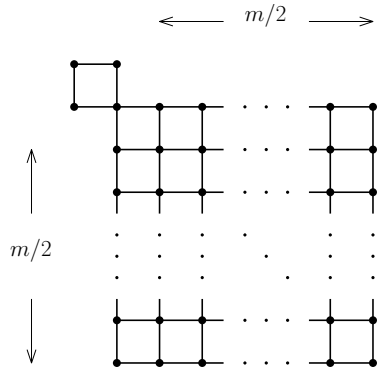

Figure 3.1: The graph G_0.

The above construction is obviously polynomial.

To complete the reduction, let us first assume that S is a yes-instance of PARTITION. Then, there are sets $S_1, S_2 \subseteq S$, such that the sum of the numbers in S_1 is equal to the sum of the numbers in S_2. Equivalently, the sum of the paths that correspond to the members of S_1 is equal to the sum of the paths that correspond to the members of S_2 and is, in fact, equal to $m/2$. Then all the paths that come from S_1 can vertically "fit into" the first column that is missing from the partial grid G_0, while the paths that come from S_2 can horizontally "fit into" the missing first line. Therefore, G_S is indeed a subgraph of the $(k \times k)$-grid.

Conversely, suppose that G_S is a subgraph of the $(k \times k)$-grid. The only way that is possible is if the vertices of the paths that correspond to the members of S were associated to the vertices of the first line and first column of the grid in the subgraph mapping (other than the first two vertices of the line and column). Let us call A the members of S that correspond to the paths associated to the first line and B the members of S that correspond to the paths associated to the first column. Then the sum of the length of the paths corresponding to A is equal to the sum of the length of the paths corresponding to B. Equivalently, the sum of the members of A is equal to the sum of the members of B, and since $\{A, B\}$ is a partition of S, it follows that S is a yes-instance of PARTITION. □

Moreover, $(k \times k)$-GRID SUBGRAPH is shown to be NP-complete even when restricted to connected graphs in [12].

Our reduction. In order to show the NP-completeness of the GRID SUBGRAPH, we will reduce to it the problem PLANAR($\leq 3, 3$)-SAT, first introduced by Dahlhaus, Johnson, Papadimitriou, Seymour, and Yannakakis [3]. This is a restriction of the PLANAR 3-SAT, proven to be NP-complete by Lichtenstein [9]. In the general problem of 3-SAT, given a formula ϕ consisting of a set $X = \{x_1, x_2, \ldots, x_n\}$ of variables and a set $C = \{c_1, c_2, \ldots, c_m\}$ of 3-element clauses (i.e., subsets of the set of literals for X, where if x_i is a variable, the corresponding *literals* are x_i and $\overline{x}_i$), the question is whether there exists a satisfying truth assignment for X and C, where a *truth assignment* for X is a subset T of the literals for X that contains precisely one of x_i, $\overline{x}_i$ for each i, $1 \leq i \leq n$, and T *satisfies* C if for all clauses $c_j \in C, c_j \cap T \neq \emptyset$. We will say that the variable x_i is *true* if the literal $x_i \in T$ and x_i is false if $\overline{x}_i \in T$.

The natural graph to associate this problem with is the bipartite *incidence* graph G_ϕ consisting of a vertex for each element of $X \cup C$ and an edge between two vertices x_i, c_j if and only if the clause c_j contains either of the literals x_i or $\overline{x}_i$. A *planar*

3-SAT formula is a formula ϕ whose incidence graph G_ϕ is planar and the goal of the corresponding Planar 3-SAT problem is to determine whether a given planar 3-SAT formula is satisfiable. A *planar* $(\leq 3, 3)$*-SAT formula* is a formula ϕ whose incidence graph G_ϕ is planar and has the additional restriction that each variable appears in exactly 3 clauses, twice in its non-negated form and once in negated and also each clause can contain either 2 or 3 literals instead of exactly 3. The corresponding problem is the following.

Planar $(\leq 3, 3)$**-SAT**
Input: A planar $(\leq 3, 3)$-SAT formula C
Question: Is C satisfiable?

Not surprisingly, the problem was proven to be hard.

Proposition 3.3. [3] Planar $(\leq 3, 3)$-SAT *is* NP*-complete.*

Given a graph G and a set $X \subseteq V(G)$, we denote by $E_X(G)$ the edges of G that are incident to vertices in X. A *signed graph* is a triple (G, X, f) where G is a graph, $X \subseteq V(G)$ is an independent set in G, and $f : E_X(G) \to \{+, -\}$ is a function assigning positive or negative signs to the edges of $E_X(G)$. Given such a graph, we also set $C = V(G) \setminus X$.

Given a signed graph (G, X, f), a *subdivision of* (G, X, f) is any signed graph (G', X, f') obtained from G after replacing each edge e of $E(G)$ with a non-trivial path P_e and where f' assigns signs to $E_X(G')$ such that, for each edge $e = \{v, u\} \in E_X(G)$ where $v \in X$, the sign $f(e')$ of the unique edge e' of P_e that is incident to v is the same as $f(e)$. We call the vertices of G' introduced after this subdivision (i.e., the vertices of G' that are not in G) *subdivision vertices* of G.

Lemma 3.4. *Let (G, X, f) be a signed graph such that*

1. *each vertex of G that is not in X has degree 2 or 3*
2. *each vertex in X has degree 3,*
3. *and for every $v \in X$, the two of its incident edges are positively signed and the other one is negatively signed.*

Then there is a signed graph (G', X, f') that is a subdivision of (G, X, f), satisfies conditions 1, 2, and 3 above (where G is replaced by G' and X remains the same), and G' is a subgraph of a grid such that

- *all positively signed edges are vertical edges,*
- *all negatively signed edges are horizontal and have their rightmost endpoint in X,*
- *if $v \in C$ and e is a vertical edge incident to v, then v is not the upmost endpoint of e, and*
- *if $v \in C$ and v has degree 2 in G', then both edges of G' that are incident to v are horizontal.*

Proof. An algorithm of Shiloach efficiently embeds a planar graph of size n and degree at most 3 in a grid of size $k = \mathcal{O}(n^2)$ [11, 13]. The rest of the proof, regarding the orientation of the signed edges, is straightforward. If we subdivide every edge of the drawing thrice, we can replace the signed edges with paths whose first edges have the required orientation (see the case-by-case analysis for the vertices in X in Figure 3.2, the rest of the vertices are treated similarly). Hence, any signed planar graph of degree at most 3 that can be embedded in the $(k \times k)$-grid can also be embedded in the $(3k \times 3k)$-grid so that its signed edges have the required orientation. □

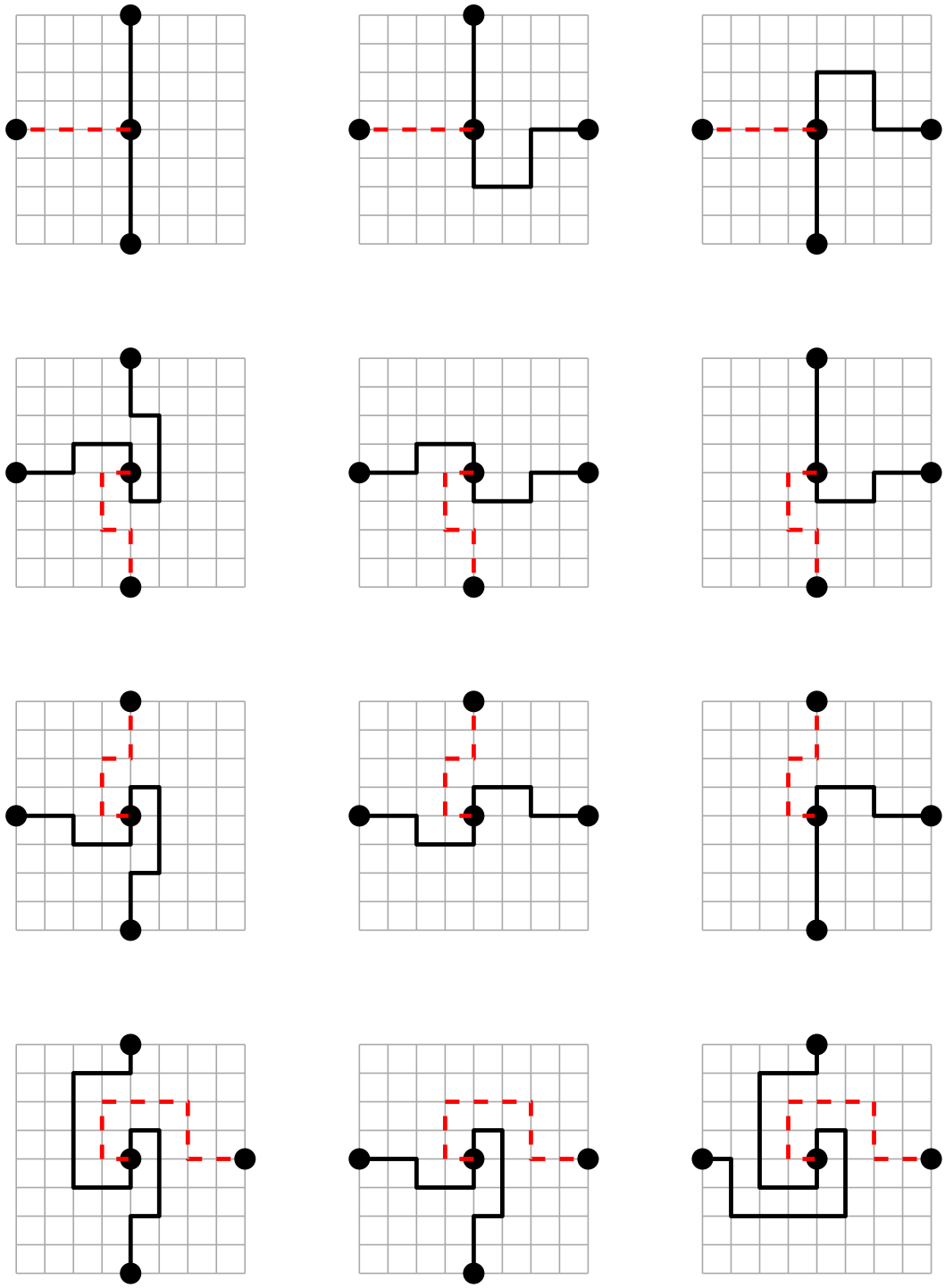

Figure 3.2: The case-by-case analysis. The vertices in X are the central vertices and the red dashed paths correspond to the negative edges.

Chapter 4

A brief look in the history of planar orthogonal drawings

Research on algorithms for drawing graphs and in particular for results on planar orthogonal drawings was heavily motivated during the '80s and '90s by problems in the VLSI circuit design and simulation of parallel architectures [5, 10, 13, 14]. This is only natural since an orthogonal graph drawing closely resembles the layout of a circuit. There are three parameters that are usually considered in such graph drawing: the area needed (i.e., the size of the grid in which the graph is embedded), the length of the paths of the embedding that correspond to edges of the graph, and the total number of bends. In this setting, a *bend* is an internal vertex of a path of the drawing that represents an edge, whose two adjacent edges are perpendicular. Avoiding bends is important in applications to light or microwave circuits where a separate device is required every time a corner is turned. Garg and Tamassia showed in [6] that is NP-complete to decide whether a a graph can be embedded in a grid without bends, therefore the corresponding minimization problem is also NP-complete.

The first question one would ask is whether a given graph accepts a planar orthogonal embedding. From [11, 17] we know that any planar graph with n vertices and maximum degree at most 4, accepts a planar orthogonal embedding in a grid of size at most $\mathcal{O}(n^2)$. Since the longest wire in a VLSI circuit can determine the performance of the circuit, the next logical step is to minimize the longest path of the embedding. Unfortunately, it is NP-hard to determine the minimum values of the total path length, maximum path length, and the required area over all possible planar orthogonal drawings of a given graph (see related results in [1, 2, 7, 8, 12]). The next step then would be to restrict the maximum path length to one, i.e., a

single edge. In that case we are talking about a unit-length embedding and this is possible only if the original graph is a subgraph of some grid. Not surprisingly, the first negative result in this direction came from Gregori [7] in 1989, who proved that the problem is NP-complete even if we restrict the input graph to be a binary tree. In fact, combining results from [1], [4], and [7] we have a complete dichotomy of when the problem is NP-complete or polynomially solvable, depending on the degrees of its vertices, which is presented in the following table (it also appears in [4]). A little surprising is the fact that the problem remains NP-complete even when restricted to most classes of trees, apart from the trivial ones.

Complete Complexity Dichotomy for the Grid Subgraph Problem		
Set of vertex degrees	General graphs	Trees
$\{1\}$	P	P
$\{2\}$	P	–
$\{3\}$	P	–
$\{4\}$	P	–
$\{1,2\}$	P	P
$\{1,3\}$	NP-complete	NP-complete
$\{1,4\}$	P	P
$\{2,3\}$	NP-complete	–
$\{2,4\}$	NP-complete	–
$\{3,4\}$	P	–
$\{1,2,3\}$	NP-complete	NP-complete
$\{1,2,4\}$	NP-complete	NP-complete
$\{1,3,4\}$	NP-complete	NP-complete
$\{2,3,4\}$	NP-complete	–
$\{1,2,3,4\}$	NP-complete	NP-complete

Graph-drawing algorithms. In 1974, Shiloach [11] provided an efficient algorithm that outputs a planar orthogonal embedding for any planar graph with maximum degree at most 3, which we are also using in the proof of Lemma 3.4. This algorithm is optimal in the sense that there are certain graph classes (of infinite size) that can only be embedded in a grid of size at least quadratic to its size. Of course for certain other classes there are algorithms that draw them in a smaller area. In particular, in the same work Shiloach also provided an algorithm for drawing trees of maximum

degree at most 3 in an area of size $\mathcal{O}(n \log n)$, while a few years later, in 1981, Valiant [17] presented an algorithm for drawing trees of maximum degree at most 4 in a linear area. Regarding algorithms that try to minimize the number of bends, Storer in [12] describes three algorithms that use different techniques to achieve that but they run in time $\mathcal{O}(n^3)$. Tamassia and Tollis improved on that in [15] and [16], by achieving linear time, using more complex techniques.

Chapter 5

The proof

Proof of Theorem 3.1. That Grid Subgraph is in NP is immediate since a certificate is a unit-length embedding of the graph. To complete the proof, we will provide a reduction to it by the Planar($\leq 3, 3$)-SAT. Namely, given an instance of Planar($\leq 3, 3$)-SAT, we convert it into an instance of Grid Subgraph, such that the constructed instance has a solution if and only if the Planar($\leq 3, 3$)-SAT instance had a solution.

Let ϕ be an instance of Planar($\leq 3, 3$)-SAT, X and C its variable and clause sets respectively, and G_ϕ its incidence graph. For the rest of the proof X and C will be used as the corresponding vertex sets that constitute $V(G_\phi)$. Since each variable appears in exactly 3 clauses and each clause contains at most 3 literals, the maximum degree of G_ϕ is 3. Due to Lemma 3.4, there is an orthogonal grid embedding S of G_ϕ of size $k = \mathcal{O}(n^2)$ with the following properties:

- For each vertex $x_i \in X$ there are 3 edges incident to x_i, two vertical and one horizontal with x_i as its rightmost endpoint. The two paths starting from the vertical edges connect x_i to the two vertices $c_j, c_k \in C$, such that the clauses c_j, c_k of ϕ contain the positive literal x_i. The third path connects x_i to the clause-vertex whose corresponding clause contains the negative literal $\overline{x}_i$.

- Each vertex $c \in C$ either has two incident edges, both horizontal, or three incident edges, with only one of them being vertical and with c as its lowest endpoint.

The vertices of $R = V(S) \setminus V(G)$ will be called *residual* vertices of S. From S we will construct a grid-like graph G' that will be able to "fit into" a grid (i.e., G' will be a

subgraph of some grid) if and only if the formula is satisfiable.

The construction is the following:

1. First, subdivide each edge of S 11 times. Now S is a subgraph of the $(11k \times 11k)$-grid. Let P be the set of the new vertices which will be called *peripheral* vertices. Obviously, up to this point, (X, C, R, P) is a partition of $V(S)$. Let m_i be the the maximum distance between c_i and any of the vertices x_j whose corresponding literals appear in the clause c_i.

2. For each vertex in $v \in X$, replace the closest 27 of its surrounding peripheral vertices (i.e., those that are at distance at most 9 from v), and the edges connecting those with a gadget as shown in Figure 5.1. Notice that these form a subgraph of the (19×19)-grid around v.

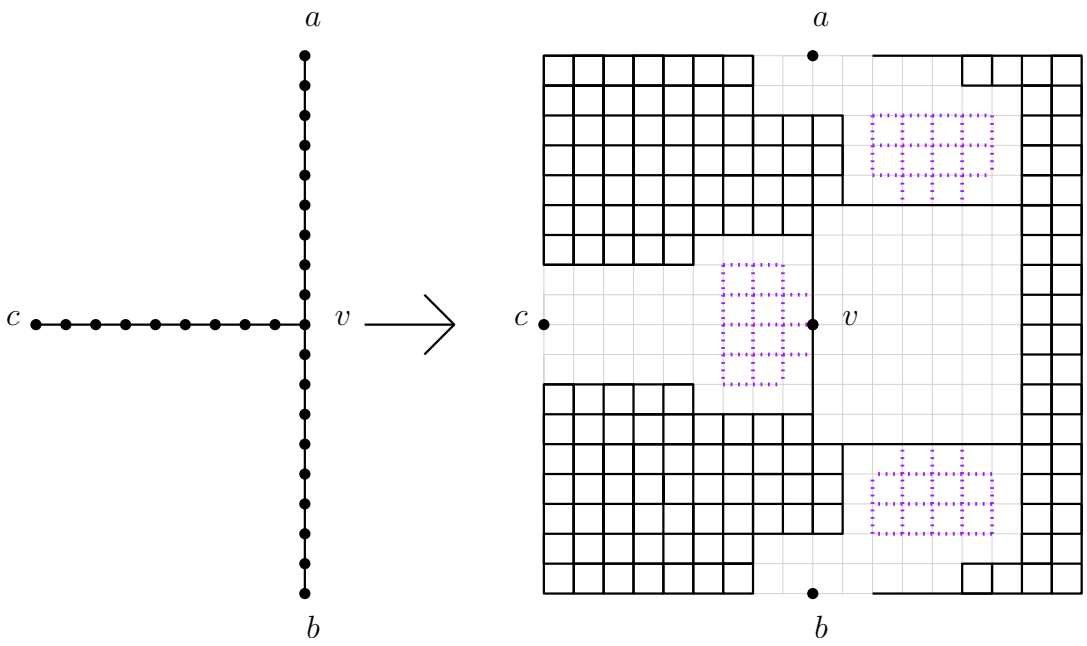

Figure 5.1: The variable gadget around the variable vertex v.

3. In this step we replace all remaining edges (not the ones added in the previous step) with paths, forming corridors around the vertices, hence we call this the *corridor gadget*. For each vertex in $C \cup R \cup P$ of degree 2, if it has two adjacent edges on a straight line, replace each edge with two parallel edges each at

distance 2 from the original edge and if it has two adjacent edges forming a bend (i.e., they are perpendicular), then replace them with two bending paths of length 2 and 10, and two perpendicular edges in the corner as shown in Figure 5.2. Also, for each vertex in C of degree 3, replace its adjacent edges with two bending paths of length 2 and a straight path of length 6, as shown in Figure 5.3. This will leave all the peripheral, residual, and clause vertices isolated.

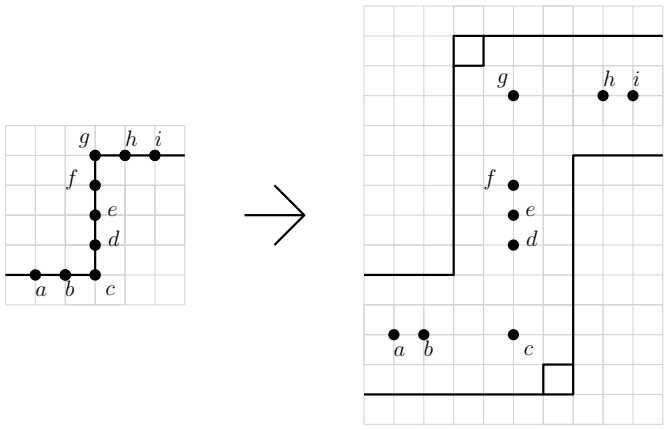

Figure 5.2: The corridor gadget for vertices of degree 2.

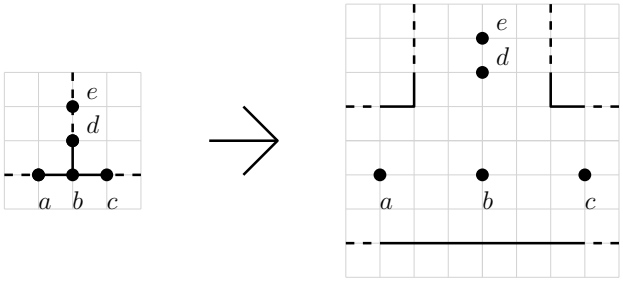

Figure 5.3: The corridor gadget for vertices of degree 3.

4. Remove the isolated peripheral and residual vertices, so that the corridors are empty inside, apart from the vertices of C and connect each clause vertex $c_i \in C$ with a path of size 2 to the closest vertex below (as shown by the dashed red

lines in Figure 5.4).

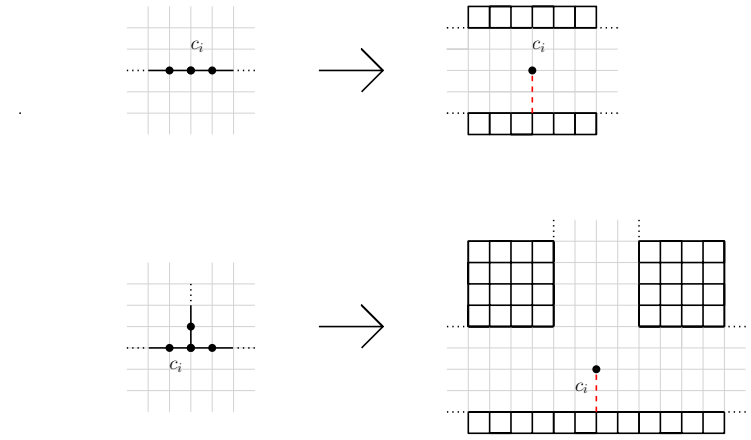

Figure 5.4: The clause gadget.

5. The graph created thus far is obviously a subgraph of the $(12k \times 12k)$-grid. We now add all the vertices and edges of the $(12k \times 12k)$-grid that contains our graph, that are not within any gadget (variable or corridor).

6. Finally, connect each vertex $c_i \in C$ to a (3×5)-grid with a path of length m_i as in Figure 5.5. This will be called the *anchor gadget*.

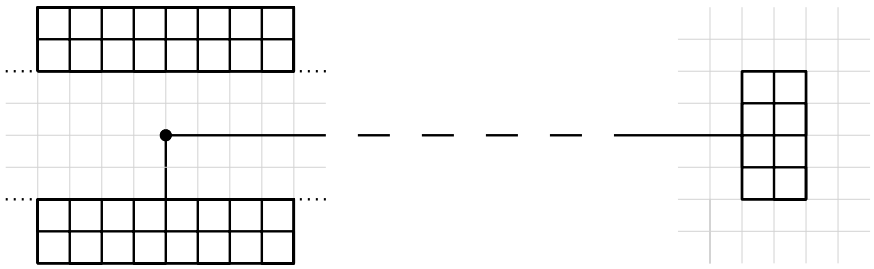

Figure 5.5: The anchor gadget.

The resulting graph G' is the corresponding instance of the Grid Subgraph problem. We shall now take a closer look at the gadgets employed.

- The variable gadget has 3 "entrances" which are connected to its surrounding corridor gadgets. At the end of each entrance there is an empty area containing a dashed purple-colored part. These areas will be called *ports* and the dashed parts *flaps*. The left entrance, port, and flap are called *negative* (since they correspond to the negative literal of the variable), while the two right ones are called *positive*. The big empty area in the center of the variable gadget is called *the hole*. In a drawing, the flaps can be placed on either side of the edges they are attached to. Also notice that if either of the two positive flaps is drawn inside the hole, then the negative must be drawn outside of it, i.e., in the negative port, but both of the positive flaps can be drawn there at the same time. Conversely, if the negative flap is drawn inside the hole, then both of the positive flaps must be drawn outside, i.e., in the positive ports.

- Each clause vertex has an anchor gadget attached to it. Obviously, these anchors are too big to be drawn inside the corridors and thus have to be drawn inside the variable gadgets and, more specifically, in their ports. Since each clause vertex is located in a corridor leading to the variables appearing in the clause, then the anchor must be drawn in a port of one of those variable gadgets.

To complete the proof, we have to prove that ϕ is satisfiable if and only if G' is a subgraph of the $(11k \times 11k)$-grid.

Suppose first that ϕ has a satisfying assignment. In that assignment, for every clause c_i there is at least one literal of a variable x_j appearing in c_i that is true. Then, in G' the anchor attached to the clause vertex c_i can be placed in the port of the corresponding literal of the variable vertex x_j. If this is the negative literal, then the negative flap will be drawn inside the hole and the positive flaps in their ports, hence no other anchor can be drawn inside the positive ports, which means that the positive literals cannot satisfy another clause. Likewise, if it is a positive literal, then the anchor will be drawn in a positive port, the corresponding positive flap will be drawn inside the hole, and the negative flap in its port, which means two things: first, the negative literal cannot be the true one in the clause it appears, and second, the other positive literal may be, since the second positive flap can also be drawn inside the hole. That way, all the anchors can be drawn in ports, thus G' is indeed a subgraph of the $(11k \times 11k)$-grid.

Suppose that G' is a subgraph of some grid. Then, it must also be a subgraph of the $(11k \times 11k)$-grid and all the anchor gadgets are drawn in ports of variable gadgets.

As we observed, inside each variable gadget can be drawn at most two anchors in the positive ports, or at most one anchor in the negative port. So, let us consider the following truth assignment for each variable:

- if an anchor was drawn in its negative port, then that variable is false,
- if at least one anchor is drawn in one of its positive ports, then that variable is true,
- and if no anchor was drawn in the ports of that variable gadget, then that variable can either be true or false, so we set it arbitrarily to true.

That way, all clauses contain at least one true literal (the one with the anchor inside the corresponding vertex gadget) and there is no variable with both literals true, since that would mean that an anchor has been placed in both the negative and a positive port of the corresponding variable vertex, which is absurd. Hence, ϕ is satisfiable.

Finally, it is evident that given a formula ϕ, then G_ϕ, S, and G' can all be constructed in quadratic time, thus the reduction is indeed polynomial. □

To clarify the gadgets used in the reduction, we provide a brief example. Consider the planar $(\leq 3, 3)$-SAT formula

$$\phi = (x_1 \vee x_2 \vee \overline{x}_3) \wedge (x_1 \vee \overline{x}_2) \wedge (x_2 \vee x_3) \wedge (\overline{x}_1 \vee x_3),$$

which contains exactly 4 clauses and 3 variables. In the formula's incidence graph G_ϕ (see Figure 5.6), every variable-vertex is connected to each clause-vertex whose corresponding clause in the formula contains at least one of the variable's literals.

Then, using Shiloack's algorithm, we derive the graph S from Figure 5.7, which is an orthogonal grid embedding of G_ϕ on the (6×10)-grid.

After we subdivide each edge 11 times, we get a subgraph of the (72×120)-grid.

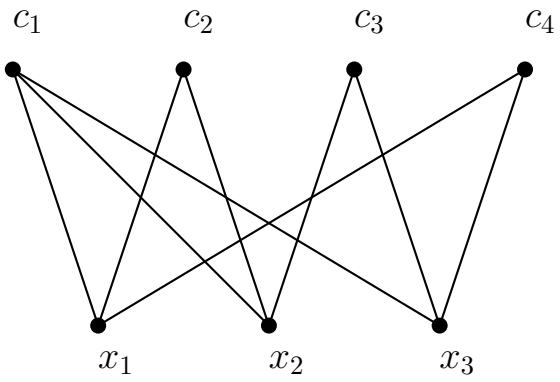

Figure 5.6: The incidence graph G_ϕ. The square vertices correspond to clauses and the disc vertices to variables.

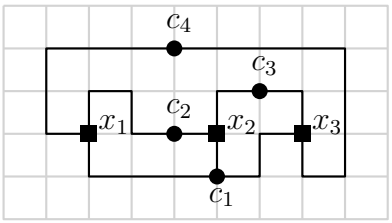

Figure 5.7: The orthogonal grid embedding S of G_ϕ. Again, the square vertices correspond to clauses and the disc vertices to variables.

Next we add the variable, clause, and corridor gadgets (see Figure 5.9).

Finally, we "fill" the rest of the grid and add the anchor gadgets (see Figure 5.10). Notice that all the anchor gadgets can be placed in the ports, hence the graph is a yes-instance of Grid Subgraph and therefore ϕ is satisfiable. Moreover, by observing in which ports the anchors were placed, we can obtain a truth assignment that satisfies the formula, namely $x_1 \leftarrow$ True, $x_2 \leftarrow$ False, and $x_3 \leftarrow$ True.

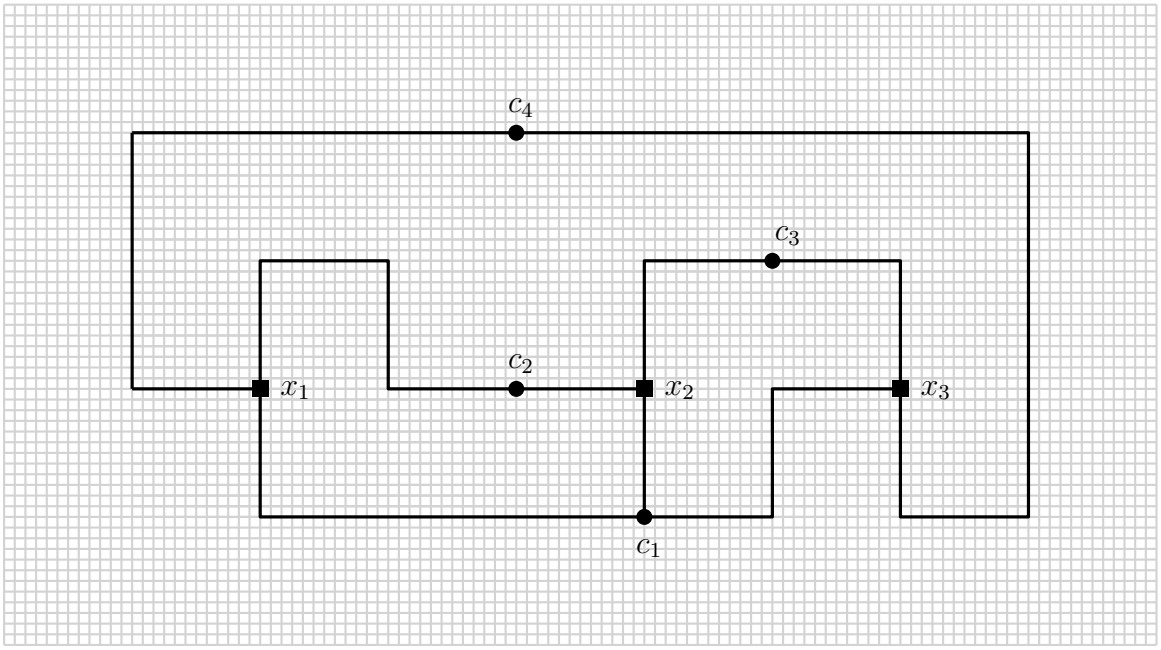

Figure 5.8: After subdividing each edge of S 11 times.

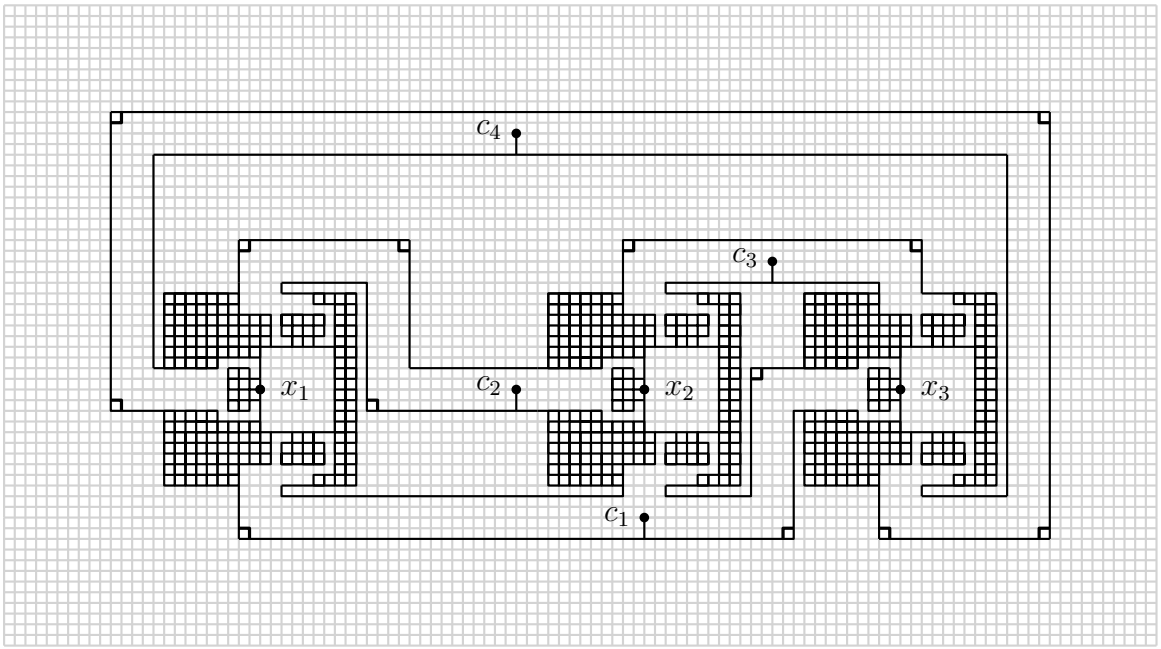

Figure 5.9: After adding the variable, clause, and corridor gadgets.

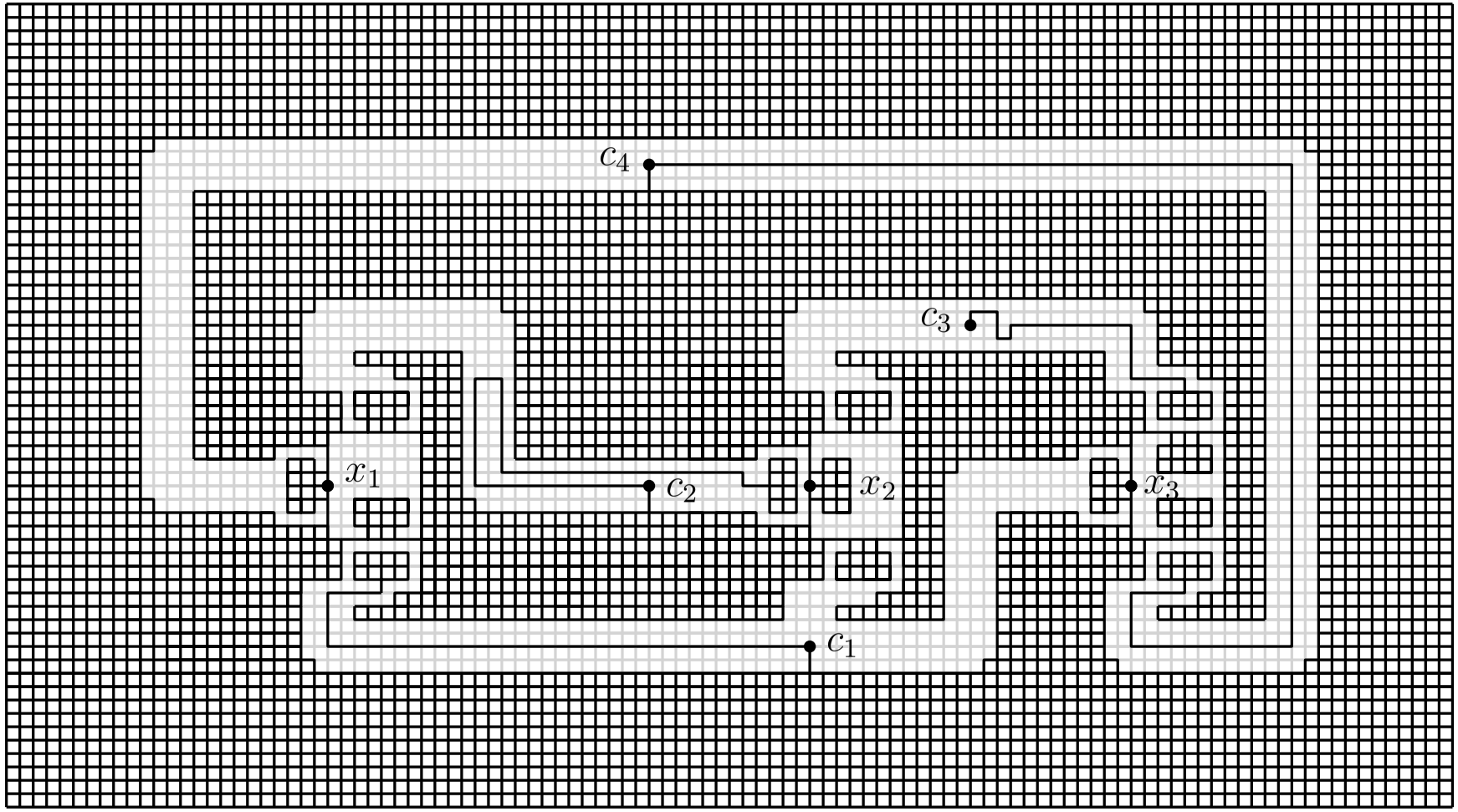

Figure 5.10: The formula is satisfiable since the graph can be embedded in a grid.

Bibliography

[1] S. N. Bhatt and S. S. Cosmadakis. The complexity of minimizing wire lengths in vlsi layouts. *Information Processing Letters*, 25(4):263 – 267, 1987.

[2] F. J. Brandenburg. *Nice drawings of graphs are computationally hard*, pages 1–15. Springer Berlin Heidelberg, Berlin, Heidelberg, 1990.

[3] E. Dahlhaus, D. S. Johnson, C. H. Papadimitriou, P. D. Seymour, and M. Yannakakis. The complexity of multiterminal cuts. *SIAM Journal on Computing*, 23:864–894, 1994.

[4] V. G. de Sá, G. D. da Fonseca, R. C. Machado, and C. M. de Figueiredo. Complexity dichotomy on partial grid recognition. *Theoretical Computer Science*, 412(22):2370 – 2379, 2011.

[5] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. Algorithms for drawing graphs: An annotated bibliography. *Computational Geometry: Theory and Applications*, 4(5):235–282, Oct. 1994.

[6] A. Garg and R. Tamassia. On the computational complexity of upward and rectilinear planarity testing. *SIAM J. Comput.*, 31(2):601–625, Feb. 2002.

[7] A. Gregori. Unit-length embedding of binary trees on a square grid. *Information Processing Letters*, 31(4):167 – 173, 1989.

[8] P. J. Idicula. Drawing trees in grids. *Masters Thesis*, 1990.

[9] D. Lichtenstein. Planar formulae and their uses. *SIAM Journal on Computing*, 2(11):329–343, 1982.

[10] Y.-B. Lin, Z. Miller, M. Perkel, D. Pritikin, and I. Sudborough. Expansion of layouts of complete binary trees into grids. *Discrete Applied Mathematics*, 131(3):611 – 642, 2003.

[11] Y. Shiloach. Arrangements of planar graphs on the planar lattice. *Ph.D. Thesis*, 1976.

[12] J. A. Storer. On minimal-node-cost planar embeddings. *Networks*, 14(2):181–212, 1984.

[13] R. Tamassia. Planar orthogonal drawings of graphs. In *IEEE International Symposium on Circuits and Systems*. IEEE, 1990.

[14] R. Tamassia, G. Di Battista, and C. Batini. Automatic graph drawing and readability of diagrams. *IEEE Trans. Syst. Man Cybern.*, 18(1):61–79, Jan. 1988.

[15] R. Tamassia and I. Tollis. Efficient embedding of planar graphs in linear time. In *Proc. IEEE Int. Symp. on Circuits and Systems*, pages 495–498, 1987.

[16] R. Tamassia and I. G. Tollis. Planar grid embedding in linear time. *IEEE Transactions on Circuits and Systems*, 36(9):1230–1234, Sep 1989.

[17] L. Valiant. Universality considerations in vlsi circuits. *IEEE Transactions on Computers*, C-30:135–140, 1981.