Ukrainian Catholic University

Bachelor Thesis

Recognition of Outer k-planar Graphs

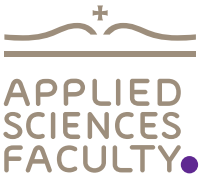
Author:
Ivan Shevchenko

Supervisor:
Prof. Dr. Alexander Wolff
Yuto Okada

A thesis submitted in fulfillment of the requirements for the degree of Bachelor of Science

in the

Department of Computer Sciences and Information Technologies
Faculty of Applied Sciences



Lviv 2025

UKRAINIAN CATHOLIC UNIVERSITY

Faculty of Applied Sciences

Bachelor of Science

Recognition of Outer k-planar Graphs

by Ivan Shevchenko

Abstract

The Thesis Abstract is written here (and usually kept to just this page). The page is kept centered vertically so can expand into the blank space above the title too...

Contents

Chapter 1

Introduction

1.1 Contributions

1.2 Structure of the thesis

Chapter 2

Related Work

Already in the 1980s, researchers in the field of graph drawing acknowledged the importance of reducing the edge crossings for improving visualisation clarity [2]. The suspicion that a drawing with fewer edge crossings was easier to comprehend was later confirmed by several experimental studies [13]. These studies showed that minimising the crossings in graph representations significantly improves the ability of humans to interpret the structure, particularly when dealing with complex or large graphs.

The ideal form of crossing minimising drawings – planar ones, has been focused on by the research community for a long time, with the first of linear-time algorithms for recognising planar graphs presented already in 1974 [10]. However, requiring the drawing to be completely crossing free imposes severe limitations on an underlying graph. While providing a clean structure, these restrictions are often too constraining for many real-world graphs, especially large ones. This has led to a growing interest in exploring graphs close to being planar; see the survey by Didimo et al. [5]. Such graphs allow a limited number of crossings while still retaining some of the beneficial structural properties of planar graphs.

2.1 Difficulty of dealing with beyond-planar graphs

Most relaxations of strict planarity dramatically increase the complexity of recognising such graphs. So, the general problem of minimising edge crossings in a graph drawing was known to be computationally intractable already in 1983 when Garey and Johnson [7] demonstrated that the Crossing Number problem determining whether a given graph can be drawn with at most k crossings, is NP-hard. Their proof relies on a reduction from the Optimal Linear Arrangement problem, which is known to be NP-hard.

Minimising the number of local crossings is also hard. Korzhik and Mohar [12] showed that testing the 1-planarity, recognising whether a graph can be drawn with at most one crossing per edge, is NP-hard. Later, Cabello and Mohar [3] showed that testing 1-planarity is still NP-hard even for near-planar graphs, that is, graphs that can be obtained from planar graphs by adding a single edge.

Given the complexity of recognising k-planarity, researchers considered exploring more restrictive classes of graphs, hoping that imposed limitations could simplify the recognition. One of the considered classes is the class of outer k-planar graphs, a subclass of k-planar graphs that admit drawings where all vertices lie on the outer face.

2.2 Efficient recognition of some outer k-planar graph

Although the general recognition problem for outer k-planar graphs is NP-hard, efficient algorithms have been developed for specific values of k.

For $k = 0$, the recognition task simplifies to an outerplanarity test. Regocnition can be accomplished by augmenting the graph with a new vertex connected to all original vertices and testing whether the resulting graph is planar. An alternative approach, described in [14], introduces the concept of 2-reducible graphs, which are totally disconnected or can be made totally disconnected by repeated deletion of edges adjacent to a vertex with a degree at most two. The proposed outerplanarity test is based on an algorithm for testing 2-reducibility.

In the case of $k = 1$, two research groups independently presented linear-time algorithms [1, 8]. Both algorithms use the SPQR decomposition of a graph for the test. Notably, the latter solution extends the graph to a maximal outer 1-planar configuration if such a drawing exists, unlike the former, which employs a bottom-up strategy which does not require any transformations of the original graph.

Considering a special case of this problem, Hong and Nagamochi [9] proposed a linear-time algorithm for recognising full outer 2-planar graphs. An outer k-planar drawing is *full* if no crossings lie on the boundary of the outer face. Later, Chaplick et al. [4] extended their result by introducing an algorithm for recognising full outer k-planar graphs for every k. Their algorithm runs in $O(f(k) \cdot n)$ time, where f is a computable function.

For the general version of the problem and values of $k > 1$, no research had been conducted until recently when a group of researchers proposed an algorithm for the general case, which we discuss in the next section.

2.3 Recognizing general outer k-planar graphs

For a given outer k-planar drawing of a graph, Firman et al. [6] proposed a method for constructing a triangulation with the property that each edge of the triangulation is crossed by at most k edges of the graph drawing. Since the edges of the triangulation do not necessarily belong to the original graph, they are termed *links* to distinguish them from the original graph edges. The construction is done recursively.

Initially, the algorithm selects an edge on the outer face and labels it as the active link. At each recursive step, the active link partitions the graph into two regions: a left part already triangulated and a right part not yet explored. The objective of each step is to triangulate the right portion. To achieve this, a splitting vertex is chosen within the right region, dividing it into two smaller subregions. The splitting vertex is selected so that the two new links, connecting the split vertex with the endpoints of the active link, are each intersected by at most k edges, which allows including them into the triangulation. The algorithm then recurses, treating each of these newly formed links as the active one.

Later, Kobayashi et al. [11] extended this approach to address the recognition problem for outer k-planar graphs. In contrast to the triangulation task, where the drawing is provided, the recognition problem requires determining whether a given graph admits an outer k-planar drawing. Although the core idea remains analogous, the absence of a drawing requires the exploration of all possible configurations. Here, each recursive step verifies whether the unexplored right portion of the graph can be drawn as an outer k-planar graph that is compatible with the left part.

Moreover, instead of relying on recursion, the method utilises a dynamic programming approach. This framework combines solutions of smaller subproblems retrieved

from a table to solve larger ones. To populate this table, the algorithm iterates over all possible configurations corresponding to different recursion steps. Several parameters characterise each such configuration. The first parameter is the active link – a pair of vertices that divides the graph into a left and a right region. The second parameter is a set of vertices in the right part, which is not uniquely determined as in the triangulation case. Additionally, the configuration depends on the order in which edges intersect the active link and the number of intersections on the right side for each one of them. These parameters are used to ensure that the drawing of the right part is compatible with the left part. For each configuration, the algorithm considers all possible ways to split the right region further. For each of them, the method checks whether these splits are compatible with each other and the left part of the drawing.

Using the restriction on the number of edges crossing each link, the authors demonstrated that for a fixed k, the number of possible right subgraphs grows only polynomially with respect to the size of the graph. They proceeded by arguing that the overall time complexity of the algorithm is $2^{O(k \log k)} n^{3k+O(1)}$, showing that the algorithm is efficient for any fixed parameter k.

2.4 Our contribution

A common drawback of the methods described above is the lack of practical validation. Although these algorithms have been analysed and discussed in a theoretical context, they have not been implemented or empirically tested. We address this gap by implementing the most recent recognition algorithm and introducing two alternative approaches based on Integer Linear Programming (ILP) and Satisfiability (SAT) formulations. While it is NP-hard to solve the general integer linear program or to find a satisfying truth assigment for general Boolean formulas, there are very advanced solvers for such formulations that could allow us to find exact solutions for small- to medium-sized instances within an acceptable amount of time. Then we evaluate the performance and efficiency of these methods, demonstrating their practical applicability and limitations.

Chapter 3

Theoretical Background

Chapter 4

Proposed Solution

This chapter will discuss the methods this work considers to recognise the outer k-planar graphs. Besides recognition, these methods provide the outer k-planar drawings of the given graphs if possible. We represent the drawing as a sequence of vertices in the circular order they appear on the boundary of the outer face. Considering only straight-line drawings, this order uniquely determines all the edge crossings and, thus, also the number of crossings per edge.

4.1 ILP Formulation (unchanged)

The result of the integral linear program should be an arrangement of vertices on a line. Given the arrangement of the vertices, we can definitively identify whether two edges cross each other. Indeed, consider two edges, uv and st. Without loss of generality, we can assume that u is located before v in the arrangement, s before t, and u before s. Under these assumptions, the edges uv and st cross if and only if s is located before v and t after v.

The arrangement is represented using the so-called "ordering variables". For every pair of vertices u and v, we create a binary variable $a_{u,v}$ that defines the order in which they appear in the arrangement. Equality $a_{u,v} = 1$ indicates that u is located before v and vice versa, and $a_{u,v} = 0$ indicates that v is located before u or u and v is the same vertex.

For these variables, it is crucial to ensure transitivity. That is, if $a_{u,v} = 1$ and $a_{v,w} = 1$ which means u is located before v and v is located before w, then u must be located before w, so the following should hold $a_{u,w} = 1$. This can be ensured by the following constraint: $a_{u,w} \geqslant a_{u,v} + a_{v,w} - 1$. This constraint will restrict the value of $a_{u,w}$ if and only if both $a_{u,v}$ and $a_{v,w}$ equal 1. Otherwise, the constraint will have no impact on the system at all.

Having the arrangement of the vertices, we can now deduce for each pair of edges uv and st whether they intersect or not. Naturally, there are 24 different arrangements of the vertices, as demonstrated in figure **??**. Among them, there are only eight in which the edges intersect. Thus, the edge uv crosses the edge st if and only if one of the following holds:

- $a_{u,s} = 1$, $a_{s,v} = 1$, and $a_{v,t} = 1$
- $a_{u,t} = 1$, $a_{t,v} = 1$, and $a_{v,s} = 1$
- $a_{v,s} = 1$, $a_{s,u} = 1$, and $a_{u,t} = 1$
- $a_{v,t} = 1$, $a_{t,u} = 1$, and $a_{u,s} = 1$
- $a_{s,u} = 1$, $a_{u,t} = 1$, and $a_{t,v} = 1$
- $a_{t,u} = 1$, $a_{u,s} = 1$, and $a_{s,v} = 1$
- $a_{s,v} = 1$, $a_{v,t} = 1$, and $a_{t,u} = 1$
- $a_{t,v} = 1$, $a_{v,s} = 1$, and $a_{s,u} = 1$

To describe this in the linear program, we create a binary variable $c_{uv,st}$ for every pair of edges uv and st. Equality $c_{uv,st} = 0$ indicates that the edge uv does not cross st. To ensure the correctness of these values, we impose eight constraints on each variable, one for each case from above. For example, considering the case $a_{u,s} = 1$, $a_{s,v} = 1$, and $a_{v,t} = 1$, we impose the following restriction: $c_{uv,st} \geqslant a_{u,s} + a_{s,v} + a_{v,t} - 2$, which ensures that $c_{uv,st}$ equals 1 whenever the vertices are ordered as $usvt$. Making this for each case restricts $c_{uv,st}$ to the value 1 if the vertices are arranged in one of the eight "intersecting" configurations, ensuring that $c_{uv,st} = 0$ is possible only if uv does not cross st.

The algorithm's objective is to minimize the maximal number of crossings per edge. This value can be written as follows: $\max_{uv \in E(G)} \sum_{st \in E(G)} c_{uv,st}$. Unfortunately, it cannot represent an objective function for a linear program as the max operation is not linear. To solve this, we introduce a new integer variable k. To ensure that it is equal to the objective value, we impose the following constraint on k: $k \geqslant \sum_{st \in E(G)} c_{uv,st}$ for each edge $uv \in E(G)$.

So, the integer linear program can be described as follows:

$$
\begin{aligned}
&\textbf{minimize} \quad k \\
&\textbf{subject to} \quad k \geqslant \sum_{st \in E(G)} c_{uv,st}, && \forall uv \in E(G) \\
&c_{uv,st} \geqslant a_{u,s} + a_{s,v} + a_{v,t} - 2, && \forall uv, st \in E(G) \\
&c_{uv,st} \geqslant a_{u,t} + a_{t,v} + a_{v,s} - 2, && \forall uv, st \in E(G) \\
&c_{uv,st} \geqslant a_{v,s} + a_{s,u} + a_{u,t} - 2, && \forall uv, st \in E(G) \\
&c_{uv,st} \geqslant a_{v,t} + a_{t,u} + a_{u,s} - 2, && \forall uv, st \in E(G) \\
&c_{uv,st} \geqslant a_{s,u} + a_{u,t} + a_{t,v} - 2, && \forall uv, st \in E(G) \\
&c_{uv,st} \geqslant a_{t,u} + a_{u,s} + a_{s,v} - 2, && \forall uv, st \in E(G) \\
&c_{uv,st} \geqslant a_{s,v} + a_{v,t} + a_{t,u} - 2, && \forall uv, st \in E(G) \\
&c_{uv,st} \geqslant a_{t,v} + a_{v,s} + a_{s,u} - 2, && \forall uv, st \in E(G) \\
&a_{u,w} \geqslant a_{u,v} + a_{v,w} - 1, && \forall u, v, w \in V(G) \\
&c_{uv,st} \in \{0, 1\}, && \forall uv, st \in E(G) \\
&a_{u,v} \in \{0, 1\}, && \forall u, v \in V(G)
\end{aligned}
$$

4.2 SAT Formulation (unchanged)

Another approach to solving this problem is to check for a specific k whether the given graph is outer-k-planar. This check can be encoded as a boolean satisfiability problem. This problem asks whether it is possible to assign logic values TRUE or FALSE so that all disjunctive clauses are satisfied. A disjunctive clause is a single literal or a disjunction of several. Literal is either a variable or a negation of a variable, with the former being the positive and the latter the negative literal.

Similarly to the ILP algorithm described in 4.1, this algorithm uses the same "ordering variables" $a_{u,v}$ for each pair of vertices u and v that represent the arrangement of the vertices. If the boolean variable $a_{u,v}$ is TRUE, the vertex u is located before the vertex v and vice versa otherwise.

Similarly, these variables must account for transitivity, which means that for every triple of vertices u, v, and w $a_{u,v} \equiv$ TRUE and $a_{v,w} \equiv$ TRUE implies $a_{u,w} \equiv$ TRUE. This can be written as follows: $a_{u,v} \wedge a_{v,w} \rightarrow a_{u,w}$. Expanding the implication,

this transforms into $\overline{a_{u,v} \wedge a_{v,w}} \vee a_{u,w}$. After applying De Morgan's law, we receive $\overline{a_{u,v}} \vee \overline{a_{v,w}} \vee a_{u,w}$, which represents a clause in the SAT problem.

The next step is to represent the crossing variables $c_{uv,st}$ in terms of the ordering ones for each pair of edges uv and st. Similarly to the ILP algorithm, we can restrict $c_{uv,st}$ to TRUE if uv and st cross by adding new clauses to the problem. The clauses are constructed by making the implications for each of the eight intersecting cases shown in figure **??**, expanding them, and applying De Morgan's law. For example, for the case $a_{u,s} = 1$, $a_{s,v} = 1$, and $a_{v,t} = 1$, we start with the logical equation as follows: $a_{u,s} \wedge a_{s,v} \wedge a_{v,t} \rightarrow c_{uv,st}$. Afterwards, we expand the implication: $\overline{a_{u,s} \wedge a_{s,v} \wedge a_{v,t}} \vee c_{uv,st}$. Finally, we apply De Morgan's law: $\overline{a_{u,s}} \vee \overline{a_{s,v}} \vee \overline{a_{v,t}} \vee c_{uv,st}$ receiving one of the eight clauses for $c_{uv,st}$.

The last step in the construction of the problem is to count the number of crossings for each edge. The goal of this solver is to check whether the number of crossings can be smaller or equal to some constant k for each edge. To ensure this, we can build a set of clauses that prevent the problem from being satisfiable if the value k is too small. To do so, for every edge e_0, we consider all combinations of $e_1, e_2, \ldots, e_{k+1}$ for each of which we construct the following clause: $\overline{c_{e_0,e_1}} \vee \overline{c_{e_0,e_2}} \vee \cdots \vee \overline{c_{e_0,e_{k+1}}}$. Doing so, we ensure that for each edge e_0, no $k+1$ different edges intersect e_0, which effectively means that each edge has at most k crossings if all clauses are satisfied.

4.3 Bicomponent decomposition

For complex problems, decomposing into smaller, independent subproblems often leads to a significant performance boost. In our context of recognising outer k-planar graphs, an effective strategy to do so is to partition the graph into subgraphs in such a manner that allows us to process them independently by the recognition algorithm. As the smallest part of the graph that can be processed by any algorithm independently of the remainder is a biconnected component, we use block-cut decomposition for this purpose, which splits the graph into biconnected components, referred to as blocks. It is worth noting that each graph edge belongs to a single block. However, any two bicomponents may share a vertex, referred to as a cut vertex. Considering blocks and cut vertices as graph nodes, we can construct a so-called block-cut tree, wherein a block node is connected to a cut node if and only if the corresponding biconnected component contains a corresponding vertex.

The advantage of this decomposition is that any method for recognising outer k-planar graphs can be applied separately to each component, and results can be combined easily afterwards. Specifically, if some component does not admit an outer k-planar drawing, neither does the whole graph. Otherwise, if all components admit such a drawing, they can be merged by combining duplicates of each cut vertex. This merging process does not introduce any additional edge crossings since both components are located on the outer face of each other. Moreover, as no new faces are created during this process – due to the acyclic structure of the block-cut tree – every vertex remains on the outer face of the graph during this process. Consequently, the resulting drawing of an original graph is outer k-planar, and it exists if and only if each biconnected component of the graph admits such a drawing.

In this work, we implemented this decomposition using BICONNECTED_COMPONENTS[1] function from Boost [**boost**]. This function assigns an index of the bicomponent to each edge to which it belongs. Additionally, it provides a list of cut vertices. Afterwards, we copy each block as an independent graph and create mappings to translate

[1] https://www.boost.org/doc/libs/1_87_0/libs/graph/doc/biconnected_components.html

new *local* vertices back to their original identifiers. Finally, we construct a supergraph representing the structure of a block-cut tree wherein each node references a copied block along with corresponding mapping or a cut vertex.

To construct a final graph drawing, we perform a depth-first search on the block-cut tree, recording the predecessor for each node upon discovery. Also, each time a block vertex is discovered, we use one of the methods described in other sections of this chapter to check whether the component admits an outer k-planar drawing and obtain it if so. Afterwards, we merge the new drawing with the already existing one by combining the corresponding cut vertex as described before. If the considered block is the first encountered one, its drawing is directly copied into a sequence that will form the final drawing. Otherwise, the block necessarily has a predecessor. Due to the structure of a tree, it is a cut node corresponding to a vertex that is shared with some other block that has already been considered and thus added to a final drawing. As a result, we can find a corresponding cut vertex in both global and local drawings. Since each drawing is represented as a cyclic sequence of vertices, we can rotate the local drawing so that the corresponding cut vertex appears as the first item in a sequence. Finally, we insert the local drawing starting from the second element into the global one immediately after the cut vertex.

> Alexander, maybe we should not exclude the first element from the local drawing but include it at the end of it, so for two touching loops ABC, and BDE we would have the following order $ABDEBC$ rather than $ABDEC$ or $ADEBC$ as vertex B indeed occurs twice on the boundary?

Chapter 5

Experiments and Results

Chapter 6

Conclusions

Bibliography

[1] Christopher Auer, Christian Bachmaier, Franz J. Brandenburg, Andreas Gleißner, Kathrin Hanauer, Daniel Neuwirth, and Josef Reislhuber. "Outer 1-Planar Graphs". In: *Algorithmica* 74.4 (2016), pp. 1293–1320. DOI: 10.1007/s00453-015-0002-1.

[2] Carlo Batini. "A layout algorithm for data flow diagrams". In: *IEEE Transactions on Software Engineering* 12.1 (1986), pp. 538–546. DOI: 10.1109/TSE.1986.6312901.

[3] Sergio Cabello and Bojan Mohar. "Adding One Edge to Planar Graphs Makes Crossing Number and 1-Planarity Hard". In: *SIAM Journal on Computing* 42.5 (2013), pp. 1803–1829. DOI: 10.1137/120872310.

[4] Steven Chaplick, Myroslav Kryven, Giuseppe Liotta, Andre Löffler, and Alexander Wolff. "Beyond Outerplanarity". In: *25th International Symposium on Graph Drawing and Network Visualization (GD)*. Ed. by Fabrizio Frati and Kwan-Liu Ma. Vol. 10692. LNCS. Springer, 2018, pp. 546–559. DOI: 10.1007/978-3-319-73915-1_42.

[5] Walter Didimo, Giuseppe Liotta, and Fabrizio Montecchiani. "A Survey on Graph Drawing Beyond Planarity". In: *ACM Computing Surveys* 52.1 (2019), pp. 1–37. DOI: 10.1145/3301281.

[6] Oksana Firman, Grzegorz Gutowski, Myroslav Kryven, Yuto Okada, and Alexander Wolff. "Bounding the Treewidth of Outer k-Planar Graphs via Triangulations". In: *32nd International Symposium on Graph Drawing and Network Visualization (GD)*. Ed. by Stefan Felsner and Karsten Klein. Vol. 320. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024, 14:1–14:17. DOI: 10.4230/LIPIcs.GD.2024.14.

[7] Michael R. Garey and David S. Johnson. "Crossing Number is NP-Complete". In: *SIAM Journal on Algebraic Discrete Methods* 4.3 (1983), pp. 312–316. DOI: 10.1137/0604033.

[8] Seok-Hee Hong, Peter Eades, Naoki Katoh, Giuseppe Liotta, Pascal Schweitzer, and Yusuke Suzuki. "A Linear-Time Algorithm for Testing Outer-1-Planarity". In: *Algorithmica* 72.4 (2015), pp. 1033–1054. DOI: 10.1007/s00453-014-9890-8.

[9] Seok-Hee Hong and Hiroshi Nagamochi. "A linear-time algorithm for testing full outer-2-planarity". In: *Discrete Applied Mathematics* 255.C (2019), pp. 234–257. DOI: 10.1016/j.dam.2018.08.018.

[10] John Hopcroft and Robert Tarjan. "Efficient Planarity Testing". In: *Journal of the ACM* 21.4 (1974), pp. 549–568. DOI: 10.1145/321850.321852.

[11] Yasuaki Kobayashi, Yuto Okada, and Alexander Wolff. "Recognizing 2-Layer and Outer k-Planar Graphs". In: *41st Annual Symposium on Computational Geometry (SoCG)*. Ed. by Oswin Aichholzer and Haitao Wang. Leibniz International Proceedings in Informatics (LIPIcs). To appear. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. DOI: 10.48550/arXiv.2412.04042.

[12] Vladimir P. Korzhik and Bojan Mohar. "Minimal Obstructions for 1-Immersions and Hardness of 1-Planarity Testing". In: *Journal of Graph Theory* 72.1 (2013), pp. 30–71. DOI: 10.1002/jgt.21630.

[13] Helen Purchase. "Which aesthetic has the greatest effect on human understanding?" In: *5th International Symposium on Graph Drawing and Network Visualization (GD)*. Ed. by Giuseppe Di Battista. Vol. 1353. LNCS. Springer, 1997, pp. 248–261. DOI: 10.1007/3-540-63938-1_67.

[14] Manfred Wiegers. "Recognizing outerplanar graphs in linear time". In: *Graph-Theoretic Concepts in Computer Science (WG)*. Ed. by Gottfried Tinhofer and Gunther Schmidt. Vol. 246. LNCS. Springer, 1987, pp. 165–176. DOI: 10.1007/3-540-17218-1_57.