Embedding Non-planar Graphs: Storage and Representation

Đorđe Klisura
University of Primorska,
Faculty of Mathematics, Natural Sciences
and Information Technologies,
Glagoljaška ulica 8, 6000 Koper, Slovenia
klisuradjordje10@gmail.com

Abstract

In this paper, we propose a convention for representing non-planar graphs and their least-crossing embeddings in a canonical way. We achieve this by using state-of-the-art tools such as canonical labelling of graphs, Nauty's Graph6 string and combinatorial representations for planar graphs. To the best of our knowledge, this has not been done before. Besides, we implement the mentioned procedure in a SageMath language and compute embeddings for certain classes of cubic, vertex-transitive and general graphs. Our main contribution is an extension of one of the graph data sets hosted on MathDataHub, and towards extending the SageMath codebase.

Keywords planar graph, graph representation, crossing number, graph database

1 Introduction

Computers are increasingly used by mathematicians to assist them in their studies. This includes most aspects of a researcher's work, from publishing and reading papers to computations in mathematical software. Perhaps surprisingly, mathematicians also generate and use data, and the production and processing of massive datasets are becoming increasingly important in several areas of mathematics.

The primary applications for these mathematical datasets and databases are exploratory in nature. They are used by researchers to test hypotheses or to discover patterns and counterexamples. It's not difficult to find such "datasets" that even predate computers: the *Atlas of Graphs* and the *Foster census* are two examples from graph theory. The Online Encyclopedia of Integer Sequences (OEIS) is a modern mathematical database that represents an online database of integer sequences. The OEIS now contains over 334000 sequences that are useful to both professional and amateur mathematicians, making it the largest database of its kind. The sequences in the database serve as fingerprints for the records they are associated with. A somewhat similar project in graph theory is the *House of Graphs*. An important notion is that of using mathematical objects, such as integer sequences, or graphs, to search for mathematical theorems. This has been introduced as theorem fingerprinting by Billey and Tenner [2] as a way to improve the efficiency of searching for mathematical knowledge. In a broader sense, fingerprints are used in many fields of science, ranging from computer science to chemistry, archaeology, and genetics. Computer documentation, reducing duplication in web search results, and surely DNA fingerprinting are a few examples.

Because of its applications in physics, biochemistry, biology, electrical engineering, astronomy, operations research, and computer science, graph theory is rapidly moving into the core of mathematics. The theory of planar graphs is based on Euler's polyhedral formula, which is related to the polyhedron edges, vertices and faces. Planar graphs are used in a variety of applications in the modern era, including constructing and organizing sophisticated radio electronic circuits, railway maps, planetary gearboxes, and chemical molecules. Pipelines, railway lines, subway tunnels, electric transmission lines, and metro lines are all vitally crucial for modelling an urban city. For further readings on this topic, look at Trudeau and Richard [13], and Barthelemy [1].

1.1 Related work and contributions

Some of the most significant projects that act as mathematical databases which are of assistance to researchers in their research projects are the SageMath platform [12], American Mathematical Society MathSciNet [11], abovementioned Encyclopedia of Integer Sequences [6], House of Graphs [5], and Atlas of Graphs [10].

The above-named tools are far from perfect and are many times subject to important work of the open-source community. This project aims to provide another toolset for researchers, via improving the platform MathDataHub which will, in the future, provide our database containing planar embeddings minimising the number of crossings. Those embeddings are hard to compute and such a database of precomputed embeddings does not exist in any mathematical database.

The paper is structured as follows. In Section 2, we present the central technique and ideas of embedding non-planar graphs. Moreover, we give a concrete algorithm that uses them and evaluates them in terms of space and time complexity. In Section 3, we talk about the impact of our results so far. Finally, in Section 4, we present some output samples of the algorithm that has been evaluated before we make some concluding remarks in Section 5.

University of Maribor Press
DOI https://doi.org/10.18690/978-961-286-516-0.13
ISBN 978-961-286-516-0

2 Embedding non-planar graphs

In this section, we present our main result, namely the algorithm for calculating the canonical embedding of non-planar graphs.

Data: Non-planar graph G
Result: Planar embedding, crossing number and added vertices of G

$V \leftarrow V(G)$
$edgePairs \leftarrow \{ab, cd$, where $a, b, c, d \in V(G)$ *and* $ab, cd \in E(G)\}$
$k \leftarrow 0$
while *G is not planar* **do**
  $S \leftarrow$ all k-subsets from $edgePairs$
  for $\{a_1, a_2, \ldots, a_k\}$ ***in*** S **do**
    $E(G) \leftarrow E(G) \setminus \bigcup_{i=1}^{k} a_i$
    for a_i *element* ***in*** $\{a_1, a_2, \ldots, a_k\}$ **do**
      $\{\{a_i^1, a_i^2\}, \{a_i^3, a_i^4\}\} \leftarrow a_i$
      $v_i \leftarrow$ *vertex such that* $v_i \notin V(G)$
      $V(G) \leftarrow V(G) \cup \{v_i\}$
      $E(G) \leftarrow E(G) \cup \{a_i^1 v_i, v_i a_i^2, a_i^3 v_i, v_i a_i^4\}$
    end
    if *G is planar* **then**
      return $G, k, V(G) \setminus V$
    end
  end
  $k \leftarrow k + 1$
end

Algorithm 1: Algorithm for calculating the canonical embedding of non-planar graphs.

After investigating several non-planar graphs with up to five vertices in SageMath we came up with the notion of representing their embeddings. Combinatorial embedding is a key concept in the study of such graph embeddings. The significance stems from the fact that, when combined with canonical labelling, combinatorial embeddings can be utilized to generate a unique representation of (planar) embeddings for (planar) graphs. For further reading on combinatorial representations and planar embeddings, refer to Mutzel and Weiskircher [9], Didjev [8], Duncan, Goodrich and Kobourov [4], and to Hopcroft and Tarjan [7].

The concept is as follows: we first construct all non-incident pairs of edges of a graph; then, we go through those pairs of edges and for each crossing of two edges, we delete those edges and add a new vertex to which we connect vertices of deleted edges. We repeat the process until the graph is planar. Finally, if it is planar, in the end, we canonically reorient its vertices and save new embedding of a graph.

We show the approach and demonstrate how it works using a Petersen graph shown in Figure 1. The first step of Algorithm 1 is constructing all pairs of non-incident edges of G, meaning that the two different edges cannot share the same vertex. Those pairs of edges are $\{\{0,1\},\{2,3\}\}$, $\{\{4,9\},\{5,8\}\}$ etc. In the beginning, we initialise crossing number k to 0, to check if the graph is already planar if it is we return k and if it is not, k is increased by 1, and we are going through the set of pairs of non-incident edges. For each k we modify the graph until we get a planar embedding, in the following way: we take the first pair, in our example pair $\{\{0,1\},\{2,3\}\}$ and we delete edges $\{0,1\}$ and $\{2,3\}$. Then we add a new vertex v to which we connect the vertices of the deleted edges: $\{0\},\{1\},\{2\},\{3\}$. We check for planarity. If the checking confirmed a positive result, that is, confirmed that the graph is planar, the crossing number is returned and the algorithm terminates. However, if the checking confirmed a negative result, that is, confirmed that the graph is not planar we go to the next pair. If all pairs fail, we look for tuples of size 3 next, and so on.

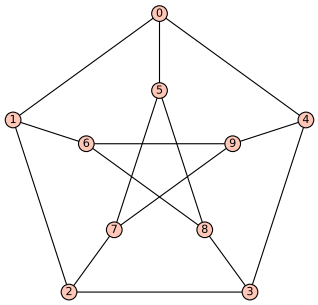


Figure 1: Petersen graph.

We get that graph G is planar after three iterations, hence, the crossing number of the Peterson graph is 2. In the end, we canonically relabel vertices and we obtain a planar embedding presented in Figure 2.

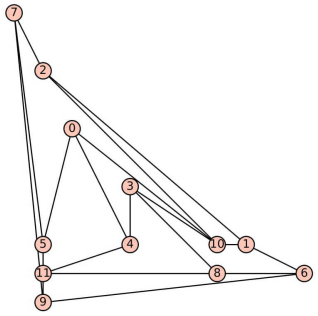


Figure 2: Planar embedding of Petersen graph after applying Algorithm 1.

See Section 3 for more details about the transformation.

2.1 Theoretical analysis of the algorithm

Consider first the space complexity of Algorithm 1. The amount of memory used by Algorithm 1 to execute and produce the result is linear with respect to the input instance. This is due to the fact that the input instance is a graph and most of the work on it is done in-place, by

modifying it locally and not taking more space even after many manipulations. Hence, we can say that Algorithm 1 does not take too much memory.

Next, let us evaluate the time complexity of Algorithm 1. To determine the time complexity, we need to consider all of the SageMath integrated functions we called in our main function. Function *is_planar* that is implemented runs in linear time, concerning the graph as an input instance, meaning it runs in time $\mathcal{O}(n+m)$ where n is a number of vertices and m is a number of edges of the graph. For more reading on the time complexity of the planarity algorithm, refer to Boyer and Myrvold [3]. To remove a vertex in a graph, we first need to find the vertex in the data structure and the time complexity depends on the structure we use; if we use a $HashMap$, the time complexity will be $\mathcal{O}(1)$. Then we remove the vertex, and we do it in $\mathcal{O}(n)$ time. Adding and removing an edge of the graph is done in constant time, $\mathcal{O}(1)$, time while adding a vertex to a graph takes $\mathcal{O}(n)$ time. Checking if there is an edge between vertices is done in $\mathcal{O}(n)$ time since a vertex can have at most $\mathcal{O}(n)$ neighbours. The time complexity of getting an embedding of the graph and of finding the neighbours is linear, that is $O(n+m)$, since we needed to perform the Breadth-First Search algorithm.

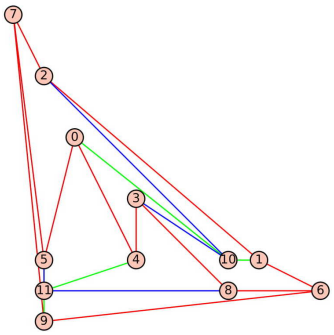
Let us analyze the lines from Algorithm 1. Line 1 has complexity $\mathcal{O}(n)$, as it assigns to a set n vertices, while line 2 assigns m^2 edge pairs to a set and hence has complexity $\mathcal{O}(m^2)$. Line 3 has constant, $\mathcal{O}(1)$, complexity.

Let us now analyse the complexity of the while loop. Inside the while loop, we see that line 5 generates all subsets of size k from the set of size m^2. Hence, the assignment of the k-subsets to the set has complexity equal to the size of the set which is $\mathcal{O}(\binom{m^2}{k}) = \mathcal{O}(m^{2k})$. Now, the first for loop goes through all k elements of the k-subset, and has complexity $\mathcal{O}(m^{2k}(k+kn+m+n)) = \mathcal{O}(m^{2k}(nk+m))$ because the line 7 has complexity $\mathcal{O}(k)$, the inner for loop has complexity $\mathcal{O}(nk)$ and the if statement in line 14, that checks whether the graph is planar, has complexity $\mathcal{O}(n+m)$. Finally, the while loop is executed k times meaning that the complexity of the whole while loop is $\mathcal{O}(m^{2k}k(nk+m))$.

3 Applications

Drawing: One of the applications is in graph colouring. In Algorithm 1, we labelled newly added edges, then we use the method *plot*() within SageMath and with the property colour by label, we get different colours for the newly added edges. In Figure 3 we see an example of the transformed Petersen graph from Figure 2. As it can be seen, the original graph embedding is coloured red, while the newly inserted edges are coloured green and blue.

Storing graphs: Another application of our approach is related to the storing of graphs with their combinatorial embedding, added vertices (if the graph is non-planar) and with Graph6 string. We constructed a function that stores data about an individual graph in a single text file that a computer can comprehend (Graph6 string, calculated embedding and added vertices - separated by



(1).jpg (1).jpg (1).jpg

Figure 3: Colouring of planar embedding of Petersen graph.

semicolons) since we aim to store graphs in the database. Furthermore, we added the certificate flag *verbose* so that we may provide a detailed output (when set to *True*) for users - with output explanations.

By now, we processed cubic graphs with up to 21 edges, vertex-transitive graphs with up to 20 edges and all graphs with up to 13 edges. These files can be stored in any database since we created a non-verbose mode of writing into them. In Table 1 we present an overall of the processed families of graphs by now.

Table 1: Processed families of graphs

family of graphs	# generated	up to edges
cubic	752	21
vertex-transitive	16	20
general	376899	13

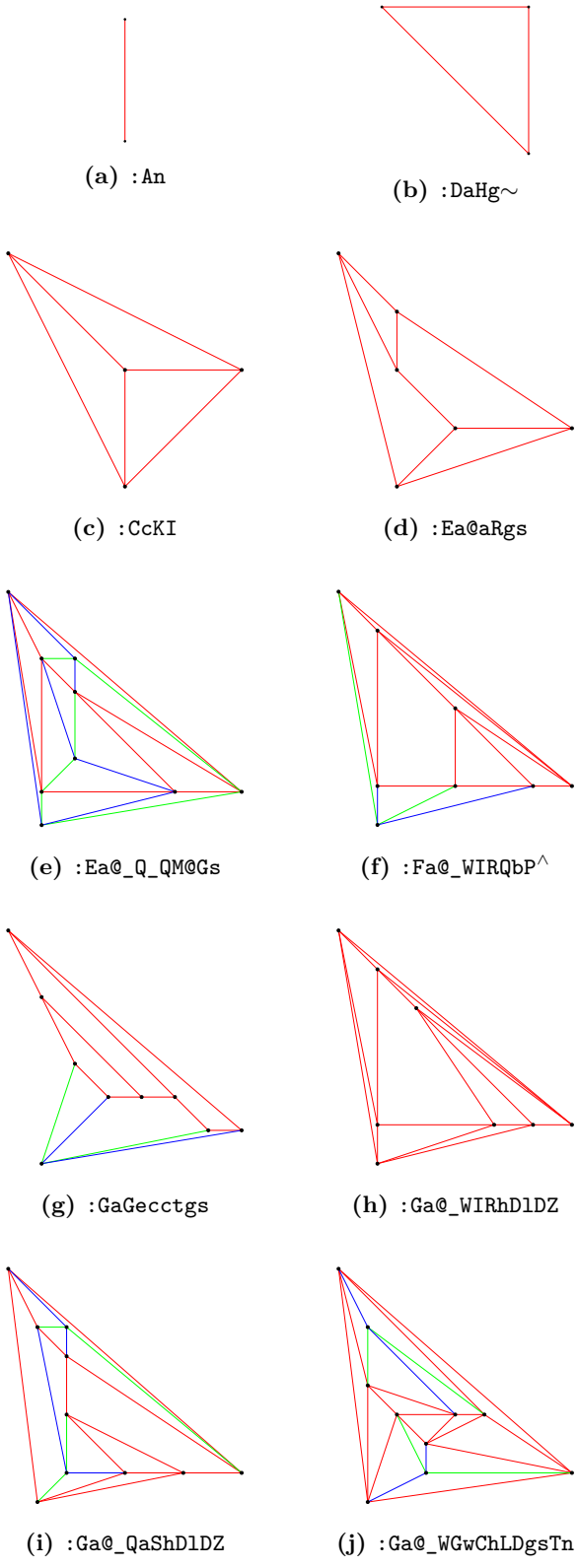
4 Output samples

Here we present some examples of the algorithm's final outputs, as previously detailed in the paper. We've successfully generated embeddings, saved them in files along with additional vertices and Graph6 strings, and plotted images of vertex-transitive graphs on less than 20 edges. In Figure 4 we present some of them individually, with the Graph6 string for each one written in the caption.

5 Conclusions

In the paper, we had a look at a non-planar graph embedding, its storage, and representation. We introduced an algorithm for constructing those embeddings and a function that writes them in both a human-readable way, or in the way suitable for the storage in the database. This contributes to the subject of representation theory because there was no standard way of encoding such embeddings.

We demonstrated how our method may be used to draw graphs and save graph data in various file formats. Our techniques can be used to enrich almost any graph



(a) `:An` **(b)** `:DaHg~` **(c)** `:CcKI` **(d)** `:Ea@aRgs` **(e)** `:Ea@_Q_QM@Gs` **(f)** `:Fa@_WIRQbP^` **(g)** `:GaGecctgs` **(h)** `:Ga@_WIRhDlDZ` **(i)** `:Ga@_QaShDlDZ` **(j)** `:Ga@_WGwChLDgsTn`

Figure 4: Coloured representations of several vertex-transitive graphs with up to 20 edges, with Graph6 string in captions.

database, and that is exactly what we were hoping to achieve. We've generated vertex-transitive graphs with up to 20 edges, all graphs with up to 13 edges, and all cubic graphs with up to 21 edges by now. In collaboration with Katja Berčič, PhD, our files will be uploaded to the *MathDataHub* database.

Currently, we are working on contributing our code to the SageMath project.

Acknowledgment

I'd like to thank Professor Matjaž Krnc for his advice and suggestions throughout the planning, development, and writing of this paper.

I'd like to thank Klemen Berkovič for his help in codifying this *Author's Guide*, and to Iztok Fister Jr. for his contribution to *Author's Guide* and `.tex` files.

References

[1] Barthelemy, M. *Morphogenesis of Spatial Networks*. New York: Springer, 2017.

[2] Billey, S. C., and Tenner, B. E. Fingerprint databases for theorems. *Notices of the AMS 60*, 8 (2013), 1034.

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

[4] Christian, D., T., G. M., and Stephen, K. Planar drawings of higher-genus graphs, graph drawing, 17th international symposium, gd 2009. vol. 5849, pp. 45–56.

[5] Goedgebeur, G. B. K. C. J., and Mélot, H. *House of Graphs: a database of interesting graphs, Discrete Applied Mathematics*, 2013.

[6] Inc., O. F. *The On-Line Encyclopedia of Integer Sequences*, 2021. `http://oeis.org`.

[7] John, H., and E., T. R. Efficient planarity testing. *Journal of the Association for Computing Machinery 21*, 4 (1974), 549–568.

[8] N., D. H. On drawing a graph convexly in the plane, graph drawing, dimacs international workshop, gd '94, princeton. vol. 894, pp. 76–83.

[9] Petra, M., and René, W. Computing optimal embeddings for planar graphs, computing and combinatorics, 6th annual international conference, cocoon 2000. vol. 1858, pp. 95–104.

[10] Read, R. C., and Wilson, R. J. *An Atlas of Graphs (Mathematics)*. Oxford University Press, Inc., USA, 2005.

[11] TePaske-King, Bert; Richert, N. *The Identification of Authors in the Mathematical Reviews Database*, 2001.

[12] The Sage Developers. *SageMath, the Sage Mathematics Software System (Version 9.4)*, 2021. `https://www.sagemath.org`.

[13] Trudeau, and J., R. *Introduction to Graph Theory*. New York: Dover Pub, 1993.