SIMPLE NUMERICAL ALGORITHM FOR Generating Hamiltonian Cycles And Edge Labels on Planar Cubic Maps

A Preprint

Emily Kendall
Department of Physics
The University of Auckland
Private Bag 92019
Auckland
New Zealand
eken000@auckland.ac.nz

ABSTRACT

In this work we present an algorithm with which any arbitrary cubic planar map may be constructed through successive edge insertion while simultaneously constructing a set of proper edge labels and Hamiltonian cycles for each configuration. We present a publicly available Python implementation of this algorithm, and discuss both theoretical and numerical support for its validity, with reference to the well-known Four Colour Theorem.

1 Introduction

The study of planar maps has been an active field of research for many decades, with famous results such as the four colour theorem generating ongoing interest [1]. Not only have the properties of planar maps been studied in the context of pure mathematics [2,3,4], but they have also been shown to have utility in the domain of theoretical physics, particularly in studies of quantum field theory and quantum gravity [5,6,7].

Given the importance of planar maps in both mathematics and physics, the development of numerical tools facilitating their study is of great importance. While there exists a wide variety of problems to be studied relating to planar maps, we focus here on the issue of Hamiltonicity of cubic planar maps [8,9], which itself is closely related to the problem of edge-colouring (edge-labelling).

Indeed, previous work has been undertaken to develop algorithms to determine Hamilton cycles within planar cubic maps [10]. Here we extend upon this work by introducing a simple algorithm with which one may build an arbitrary planar cubic map through successive edge insertion, whilst enumerating after every insertion the proper edge-labellings associated with the resulting map, as well as its Hamilton cycles. We also present a simple open-source Python implementation of this algorithm.

The structure of this paper is as follows. In Section 2, we outline the fundamental concepts of the algorithm with the aid of simple illustrations. In Section 3, we describe how this algorithm is implemented within a simple, standalone Python code. In Section 4, we describe tests undertaken on the Python implementation, and provide a simple example of the outputs generated from a given execution. Finally, in Section 5, we discuss the merits of this algorithm, and the scope for future work.

2 Algorithm description

The algorithm which we use for the construction and labelling of cubic planar maps may be decomposed into three main steps: 1) initialisation, 2) cycle construction, and 3) edge insertion and re-labelling. We describe each step in...
detail in the following subsections. The corresponding Python implementation is available at [https://github.com/erckendall/Edge_Labelling.git](https://github.com/erckendall/Edge_Labelling.git).

### 2.1 Step 1: Initialisation

In order to build an arbitrary cubic planar map using this algorithm, we must first define an initial configuration. The initial configuration consists of a planar cubic map and a set of mutually exclusive even cycles over that map, such that every vertex lies on exactly one cycle. The simplest possible initial configuration is the cubic planar map with two vertices, as shown in Figure 1. Clearly, constructing a set of even cycles through all vertices is trivial in this case.

![Figure 1: The simplest possible initial configuration. A single even cycle through all vertices is shown in green. Edges are enumerated in red, while vertices are enumerated in blue.](image)

While the trivial configuration of Figure 1 may always be used for initialisation, one may also initialise the algorithm using a more complicated map if desired. For the purposes of this explanation, it is instructive to choose the configuration illustrated in Figure 2 as the initial configuration. In this case, we have not one, but two even cycles, each of which traverses four of the eight map vertices. While Figure 2 is useful for illustrative purposes, we must encode this information in matrix form in order to feed it into the Python implementation of the algorithm. To do this, we construct a vertex-edge matrix and a face-edge matrix to describe the map configuration. We also construct a set of lists containing the edges of each cycle. Note that the lists of edges need not be in order, as the Python implementation contains a function to determine this independently. Note also that the numbering of faces, edges and vertices is arbitrary, and requires only that the matrix elements are consistent with the desired graphical representation. For the initial configuration illustrated in Figure 2, the list of edges which define the cycles is simply \( [1, 9, 10, 11], [3, 4, 5, 6] \).

Meanwhile, the vertex-edge and face-edge matrices are illustrated in Figures 3 and 4, respectively. Once the two matrices and list of edges on each cycle is input into the Python implementation of the algorithm, the initialisation step is complete.

### 2.2 Cycle construction

In the initialisation step of the algorithm, we defined a both a cubic planar map and a set of even cycles upon that map such that every vertex is traversed by exactly one cycle. Henceforth, we will refer to such a set of cycles as a ‘complete cycle group’ (CCG).

Before we proceed, we note that the initial CCG immediately gives us a proper edge labelling of the map; because each cycle is even, we can apply alternate \( a, b, a, b... \) labels around each cycle. Furthermore, because every map vertex is traversed by exactly one cycle, it is therefore also associated with exactly one edge which is not part of any cycle. Hence, the remaining edges may all be labelled \( c \), providing a proper edge labelling of the map. We will discuss this labelling procedure further in the following sections.

In the second step of the algorithm, we will use the initial CCG to construct a larger set of alternative CCGs. We describe this procedure below, and then provide a worked example to illustrate each step.

1. For each cycle in the initial CCG, we label consecutive edges alternately as \( a, b, a, b... \) such that we can define a set of ‘\( a \)’ edges and a set of ‘\( b \)’ edges for each cycle.

2. Choosing either \( a \) or \( b \) for each cycle, we enumerate all possible sets of choices, amounting to \( 2^n \) combinations, where \( n \) is the number of cycles. For example, if we have two initial cycles in our original CCG, our choices are \( \{ (a, a), (a, b), (b, a), (b, b) \} \).
Figure 2: An example of a more complicated initial configuration. Here we have a set of two even cycles covering the map (green). Edges are enumerated in red, while vertices are enumerated in blue. We include also an enumeration of the map faces on the right (pink).

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

Figure 3: Vertex-edge matrix encoding the map illustrated in Figure 2. Note that for a planar cubic map each column possesses exactly two ‘1’s, while each row possesses exactly three ‘1’s.

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

Figure 4: Face-edge matrix encoding the map illustrated in Figure 2. Columns representing external edges possess only one ‘1’, while internal edges possess two.

3. For each of the $2^n$ combinations, we retain (or ‘turn on’) the portions of each cycle which traverse edges with the chosen label, and delete (or ‘turn off’) those portions of the cycle traversing edges labelled with the other letter.
4. We now create a series of new CCGs by ‘turning on’ each of the edges which were not traversed by any of the cycles in the initial CCG.

5. We repeat the above steps for each new CCG obtained. As we do so, we find that some CCGs which have been obtained previously will emerge again. Eventually, no new configurations will arise, and we will converge onto a finite set of complete cycle groups.

We now provide an example of this process using the initial configuration of Figure 2. In this case, we have two cycles in the initial CCG. Therefore, we have four possible combinations of edges to ‘turn on’. We illustrate these combinations and the resulting CCGs they generate in Figures 5 to 8. After having obtained the four new CCGs as illustrated, we then take each of these in turn as our new initial configuration, and repeat the process to produce further CCGs. We do this for every distinct CCG created, until we converge upon a finite set. For this particular example, we converge upon a set of nine distinct CCGs, as illustrated in Figure 9. We note that of these nine complete cycle groups, six of these are Hamiltonian cycles.

At this point it is pertinent to highlight an important conjecture upon which this algorithm relies:

**Conjecture 1:** If we possess a single CCG for a given planar cubic map, it is possible to produce all possible CCGs for that map through the iterative procedure described in steps 1 to 5, which will always converge to the final complete set of CCGs.

That is to say, we conjecture that the cycle construction procedure discussed above is exhaustive. We will discuss this notion further in later sections. At present, we simply note that if Conjecture 1 holds, then this algorithm necessarily identifies all possible Hamiltonian cycles for any given map configuration, as all such cycles constitute a CCG.

While the construction of Hamiltonian cycles is in itself interesting, we note that in building the complete set of CCG’s, we also build a complete set of proper edge labels, since every proper edge labelling must contain a set of mutually exclusive \(a - b - a - b\) cycles, separated by edges which may be labelled \(c\).

While we have already discussed how one may identify a proper edge labelling from any given CCG, it is important to take into account the degeneracy introduced through the arbitrary assignment of ‘a’, ‘b’, and ‘c’. Hence, we must identify the three distinct sets of edges corresponding to each labelling. Any two labelings are equivalent provided that these sets contain the same elements, irrespective of set order. The Python implementation of this algorithm automatically identifies these equivalences, and outputs only the final set of distinct proper edge labellings.

Figure 5: Left: first possible combination of original cycle segments from from Figure 2. Right: the first new complete cycle group.

---

1This process necessarily creates even cycles through every vertex of the map, since we are always free to relabel each of the initial cycles independently through \(a \leftrightarrow b\), such that the final configuration amounts to a traversal along \(\{a - c\}\), or \(\{b - c\}\) paths, which are necessarily even when arising from an initial proper edge labelling.
2.3 Edge insertion and re-labelling

In this section we discuss our procedure for introducing new edges into the map while retaining a suitable CCG, and hence a proper edge labelling. This procedure relies on the following important conjecture:

**Conjecture 2:** For any two edges which lie on the same face of a planar cubic map, there exists at least one CCG such that both edges lie upon the same cycle within this group.

We will discuss this conjecture further Section 5. At present, however, we note that if Conjecture 2 holds true, it is always possible to choose a CCG such that the insertion of an edge results in the addition of two new vertices to the same cycle within the CCG, thereby preserving evenness, and creating a new planar cubic map for which we automatically have at least one CCG. We illustrate this principle in Figure 10.

Having chosen an appropriate CCG, we may then insert the new edge and re-label the map. We then re-define the CCG according to the labelling of the new map, and we thus obtain a new initial configuration to feed into the cycle construction step. Note that we must re-label not only the vertices and edges, but also the faces of the map, as the edge insertion leads to the creation of one additional face. The identification of an appropriate CCG, and the subsequent re-labelling is done automatically in the Python implementation of this algorithm.
3 Outline of the Python implementation

The Python implementation of our algorithm is designed to add edges at random to the initial map configuration until a specified number of edges has been added. After each edge addition, the corresponding CCGs and edge labellings are produced. The Python implementation may be accessed at [https://github.com/erckendall/Edge_Labelling](https://github.com/erckendall/Edge_Labelling).

A skeleton outline of the Python implementation is as follows:
1. The user inputs the initial vertex-edge matrix \( (\text{mat}_\text{in}) \), face-edge matrix \( (\text{mat}_\text{in}_\text{face}) \), and list of edges on each cycle of the initial CCG \( (\text{cycles}_\text{in}) \). The user also specifies the number of edge addition operations to perform using the \textit{iterations} parameter.

2. Outer loop begins:
   2.1. Inner loop begins:
      2.1.1. The validity of the input vertex-edge matrix is checked using \textit{func_check_validity}.
      2.1.2. The edges which are ‘turned off’ for the current CCG are determined using \textit{func_empties}.
      2.1.3. The edges within the list of cycles of the input CCG are ordered using \textit{func_ordering}.
      2.1.4. The \( 2^n \) \( a-b \) combinations are generated using \textit{func_combos}, where \( n \) is the number of cycles in the CCG.
      2.1.5. The corresponding set of edge labels for each combination is produced using \textit{func_labels}.
      2.1.6. A new list of ‘turned on’ edges is generated for each of the \( 2^n \) combinations using \textit{func_new_ons}.
      2.1.7. The corresponding new CCG for each combination is generated using \textit{func_more_cycles}.
   2.2. Inner loop terminates when set of CCGs converges. Output is the complete set of CCG’s for the given map and the corresponding edge labellings.
   2.3. Duplicates in the outputs of the inner loop are removed using \textit{func_duplicate_labels} and \textit{func_remove_duplicates}. Hamiltonian cycles are also identified.
   2.4. Outputs are printed to the screen for the existing map configuration
   2.5. One face of the existing map is chosen at random, as are two of its edges (may be the same edge) using \textit{func_choose_edges}.
   2.6. The order of the edges around the chosen face is determined using \textit{func_order_face}.
   2.7. A new vertex-edge matrix is generated corresponding to the addition of a new edge between the two randomly chosen existing edges using \textit{func_new_v_e_mat}.
   2.8. A new face-edge matrix is generated following edge addition using \textit{func_new_f_e_mat}.
   2.9. An appropriate CCG from the previous map configuration is chosen (such that the two randomly chosen edges both lie on the same cycle) using \textit{func_choose_cycle}.
   2.10. The chosen CCG is re-written to include the newly created edges using \textit{func_new_cycles}.
   2.11. The new matrices and CCG are initialised for the next iteration.

3. Outer loop terminates when specified number of edge additions has been completed. Outputs are printed to the screen after each edge addition.

Figure 10: Insertion of a new edge between edges 3 and 4 in the original map. From the previously generated set of CCGs, we choose one of these such that both new vertices lie upon the same cycle. We then re-label the map edges and vertices, and obtain a new planar cubic map with a corresponding CCG.
4 Verification of the Python implementation

The Python implementation was tested by inserting edges at random and re-labelling the map using the algorithm outlined above. The code can run up to 20 edge additions in under a minute on standard desktop hardware, calculating the set of CCGs and distinct edge labellings after each addition. As this process was randomised and repeated numerous times, we have tested a wide variety of possible map configurations, with the code successfully running to completion in every test.

Here we demonstrate part of the output of one such run graphically. We take the map illustrated in Figure 2 as our starting configuration, and present here the outcomes of the first four edge additions (Figure 11), complete with the first CCG assigned in each case. We omit vertex/edge/face labels for clarity.

![Figure 11: Demonstration of random edge addition and CCG generation using the Python implementation. Top left: 1 edge added, top right: 2 edges added, bottom left: 3 edges added, bottom right: 4 edges added.](image)

5 Discussion and conclusions

The problem of edge-labelling of graphs has been extensively studied, (see e.g. [11], [12], [13]). The edge-labelling of planar cubic graphs is of particular significance, as it has been established that the four colour theorem ([11]) is equivalent to the conjecture that every planar cubic bridgeless graph admits a proper edge labelling ([14], [15]). We show how a proper edge labelling of a cubic planar map can be mapped into a four colouring in Appendix C and we show how this implies a proper four colouring of arbitrary planar maps in Appendix B.
The four colour theorem has been proved through numerical methods ([16], [17], [18]), implying therefore that it is always possible to find a proper edge labelling of a cubic planar map. The algorithm discussed here represents a method for determining these proper edge labellings through the construction of even cycle coverings (CCGs) over the map.

The success of this algorithm rests upon the truth of two conjectures, which we reiterate here:

**Conjecture 1:** If we possess a single CCG for a given planar cubic map, it is possible to produce all possible CCGs for that map through the iterative procedure described in steps 1 to 5, which will always converge to the final complete set of CCGs.

**Conjecture 2:** For any two edges which lie on the same face of a planar cubic map, there exists at least one CCG such that both edges lie upon the same cycle within this group.

The first of these conjectures relates to the completeness of the algorithm, while the second may be viewed as a rephrasing of the four colour theorem. That is to say, if Conjecture 2 is true, it is always possible to build an arbitrary planar cubic map possessing a proper edge labelling through successive edge addition, provided the complete set of CCGs can be determined for every intermediate configuration (Conjecture 1). As the four colour theorem has only been proven through numerical means, it is assumed that the proof of Conjecture 2 cannot be reduced to a lesser degree of complexity, but could perhaps also be proved explicitly through computational means. We leave the search for such a proof for future work, but note that the success of the Python implementation of this algorithm presents strong numerical evidence for its validity.

Finally, we note that if Conjecture 1 holds true, then this algorithm presents a method for determining all possible Hamiltonian cycles for a given cubic planar map. The Hamiltonicity of cubic planar maps is also of great interest, see for example [19]. Our Python implementation automatically checks for the existence of Hamiltonian cycles within the CCGs produced, and will raise an exception if no Hamiltonian cycles are found. We note that throughout our test runs of the Python implementation, we have not encountered a case in which no Hamiltonian cycle is found for any configuration.

We intend that this algorithm and its associated implementation may prove a useful tool for the study of the properties of planar cubic maps, and that the conjectures presented here may lead to further progress in the understanding of the implications of the four colour theorem.

**Acknowledgements**

I would like to thank Georgina Liversidge, whose extensive knowledge of graph theory was an invaluable help. I would also like to thank Richard Easther, who patiently allowed me to spend time on graph theory when I was supposed to be doing cosmology.

**References**

1. Encyclopædia of Mathematics. Four-colour problem. [http://www.encyclopediaofmath.org/index.php?title=Four-colour_problem&oldid=31833](http://www.encyclopediaofmath.org/index.php?title=Four-colour_problem&oldid=31833). Accessed: 2019-08-22.

2. W. G. Brown and W. T. Tutte. On the enumeration of rooted non-separable planar maps. *Canadian Journal of Mathematics*, 16:572–577, 1964.

3. Gilles Schaeffer. Bijective census and random generation of eulerian planar maps with prescribed vertex degrees. *The Electronic Journal of Combinatorics [electronic only]*, 4(1):Research paper R20, 14 p.–Research paper R20, 14 p., 1997.

4. Mireille Bousquet-Mélou. Counting planar maps, coloured or uncoloured. In *23rd British Combinatorial Conference*, volume 392 of *London Math. Soc. Lecture Note Ser*, pages 1–50, Exeter. United Kingdom, July 2011. Surveys in combinatorics 2011, London Math. Soc. Lecture Note Ser. 392, pp. 1-50.

5. D. Bessis, C. Itzykson, and J. B. Zuber. Quantum field theory techniques in graphical enumeration. *Adv. Appl. Math.*, 1:109–157, 1980.

6. J. Bouttier, P. Di Francesco, and E. Guitter. Counting colored random triangulations. *Nuclear Physics B*, 641(3):519–532, October 2002.

Note that the Python implementation will also alert the user if none of the Hamiltonian cycles is suitable for the next edge addition, but this is not to say that no Hamiltonian cycles have been found.
Appendix A  Construction of an arbitrary cubic planar map from a trivial starting configuration

All 2-connected planar cubic maps consist of a set of edges which terminate at order-3 vertices only. Hence, if we take an arbitrary cubic planar map and remove any one edge, along with its two terminal vertices, the result is again a 2-connected planar cubic map. Because a valid 2-connected map must have a continuous outer boundary, we can consider successively removing only internal edges (along with their terminal vertices) until we end up with the most trivial planar cubic map, demonstrated in Figure 12. Hence, if every planar cubic map may be reduced to this trivial configuration through successive removal of internal edges, we may conversely construct any arbitrary planar cubic map by successive insertion of edges. A new edge may be inserted such that each of its terminal vertices are placed along the same existing edge, or such that each terminal vertex is placed upon a different existing edge (see Figure 13). We note, however, that due to planarity the existing edges to which the new edge is attached must always lie upon the same face.

Figure 12: The most trivial planar cubic map, from which any arbitrary planar cubic map may be generated through successive insertion of internal edges. Note that this trivial configuration is easily properly edge-labelled, as shown in the figure.
Appendix B  Conversion of arbitrary planar maps into cubic planar maps

Let us first consider the arrangement of order-3 vertices shown in the left side of Figure 14, here we have a collection of order 3 vertices, arranged around a circle. Let us now imagine constricting the diameter of this circle, as shown in the central diagram. If we continue constricting the circle, we arrive at an infinitesimal point. At this point, the collection of order 3 vertices becomes a single vertex of higher order (in this case order 4). If we consider this process in reverse, we see that we can always transform a vertex of arbitrary order $\geq 3$ into a collection of order 3 vertices.

Hence, if we take an arbitrary planar map, which contains any number of vertices of order $\geq 3$, we can imagine blowing up each of the vertices in the manner described above, such that we generate a new planar map, containing only order 3 vertices. That is, a planar cubic (3-regular) map (Note that the original map must be bridgeless, as all edges must constitute borders separating distinct regions. By consequence, no vertices of order $< 3$ exist. Furthermore, any such map must be at least 2-connected, except for ‘enclaves’, which themselves constitute complete 2-connected maps, and can be labelled independently).

This type of map conversion has the following important consequence: If we can prove that it is always possible to obtain a four-colouring of any 2-connected (bridgeless) planar cubic map, then the full Four Colour Theorem is automatically proved. This is because if we ‘blow up’ an arbitrary planar map in the manner described above, and find that the resulting cubic map possesses a four-colouring, then if we re-collapse the vertices to recover the original map we do not introduce any new neighbours for any region of the original map. Indeed, each region actually loses a neighbour as the circular buffer region collapses into an infinitesimal point, and thus no new restrictions on the colouring of any given region are introduced. Thus, if the expanded (cubic) map is four-colourable, the original map with vertices of arbitrary order must also be four-colourable.
Appendix C  Colouring of a properly edge-labelled 3-connected planar cubic map

Let us imagine that we have a properly edge-labelled cubic map, as demonstrated in Figure 15. Here it can be seen that around each vertex, each edge is labelled differently, namely a, b, or c.

Let us now consider the relationship between the proper edge-labelling and the colouring of the map regions. Assuming we have an ensemble of four different colours to assign to the map regions, for convenience let us denote each colour by a two-index identifier, namely one of the set \{(+, +), (−, −), (+, −), (−, +)\}. To satisfy the colouring criterion, no two neighbouring map regions may possess the same index. Hence, when traversing an edge, one of three possible operations must occur:

1. The first index swaps sign
2. The second index swaps sign
3. Both indices swap sign

Let us therefore assign to each of the edge labels one of the three above operations. e.g. \(a =\) swap first index, \(b =\) swap second index, \(c =\) swap both indices. Now, if we begin from a properly edge-labelled map such as that shown in Figure 15, we can randomly choose a starting region, and assign to it one of the four two-index identifiers. From here, we can assign identifiers to the rest of the map regions, according to the label of the edge traversed to get to that region. We demonstrate this labelling process in Figure 16.

![Figure 15: Proper edge labelling of the tetrahedral 3-connected planar cubic map.](image)

![Figure 16: Colour index assignment according to edge labelling.](image)
background is in its entirety will be four-colourable. This means that ‘enclaves’, or submaps completely embedded in one larger map region may be treated separately, and inserted without compromising the colouring of the entire map.