Different techniques for visualisation of combinatorial maps

Tingyu Huang
1899855
MSc in Advanced Computer Science

Supervisor:Noam Zeilberger

School of Computer Science
University of Birmingham

September 10, 2019

Contents

Abstract

The combinatorial map is a combinatorial topological structure. It represents a graph embedded into the surface. This project aims to search for different techniques for visualisation of combinatorial maps. The combinatorial map has many features, and the methods implemented in the project come from them. In order to attract more people learning the structure, the methods are collected as a tool in JAVASCRIPT. The methods and the visualiser are evaluated successfully though there are some problems. Some of the methods still can be improved in the future.

Keywords: Combinatorial maps, visualisation, topological structure

Note

Some content in section 3 has been based on the content written in my Mini-project report[Hua19].

Source code

The source code for this project can be found at https://git-teaching.cs.bham.ac.uk/mod-msc-proj-2018/txh755.

Acknowledgments

Thanks to my supervisor, Noam Zeilberger, under his guidance, my project went smoothly. He always provides me with sufficient background knowledge, gives me some key tips in ideas, and highlights key issues to help me improve my project. I also thank to my families and friends who support me to go throughout the year.

CHAPTER 1

Introduction

1.1 Motivation

Combinatorial map is a combinatorial topological structure which describe a graph embedded in a surface. It includes three components vertices, faces, and edges, which are represent by the permutations of darts (half-edges). There are three notations of the permutations two-notation, one-line notation and cycle notation. All of the notations are used to represent the each component of the map, while, cycle notation and one-line are mentioned in the calculation frequently and two-line notation is utilised into visualisation generally. There are some strict constrains of a map, so that only if two maps are isomorphic by a homeomorphism of the underlying surfaces, they could be conjugation equivalent.

The drawing a topological structure is heated topic [Tam13], while, the visualisation of combinatorial maps is rarely mentioned.This project aims to build a tool for visualising the combinatorial maps by using different techniques. Firstly, the tool could make the concept more accessible and interesting, which is quite important for the beginners who are new to the relevant knowledge. As the mean time, for those researchers who conduct related research, this tool could help them improve their efficiency and productivity effectively.

Due to the final goal of the project is drawing the map automatically, the project will pay more attention on the layout of the map. The spanning tree and dual maps are discussed as the based sketch of the structure when visualising the maps.

1.2 Tool structure

The Tool has two pages introduction and visualiser. The first page is to introduce the combinatorial maps and relevant knowledges involved in the visualiser.

The second page has three units. The head unit is also an introduction that offer the details of the page, namely, what the pages consist of, how to operate each unit and the representation of each terminology. The input unit includes three types of input. In the first section some examples are supported for users to experience. For the second section, users can try or validate their own specific sample. Users can also play with the last section for generation a random map by only entering the number of darts. The last unit is the output part which contains the visualisation of each component of combinatorial maps with cycle and one-line notations, the other information of the maps and the final results of the methods. Each results can be clicked to view more details.

1.3 Report structure

The report structured as follow. Section 2 provides the related knowledge of combinatorial maps, as well as its purpose and significance. Section 3 describes the achievement in this

area, the former method was proposed by others and the later method was what I have done in the last semester mini-project. Section 4 gives more specifications of the tool. The ideas of visualisation of maps and the construct of the tool are involved in section 5. Section 6 evaluates the methods and the system. Section7 details the achievements, issues and improvement of the system. Appendix A shows the file structure of the project and Appendix B records the weekly project log.

CHAPTER 2

Background

Combinatorial map is a topological structure describing the rotational graph embedded in the surface. It includes three components: vertices, edges and faces. Each of them is composed of permutations of darts. This section will introduce these concepts and terminology.

2.1 Permutations

Permutation is arranging all members of a set or a sequence with a certain order, and each order called a permutation. For example, the set {1,2} have two permutations (1 2) and (2 1). Each permutation is written as tuple. Three notations can be used to represent this concept, two-line notation, one-line notation and cycle notation.

Two-line notation

As the name implies, this notation uses two rows two indicate the permutations. The first row is the list of elements in set , for instance, the first row of set {1,2,3} would be (1 2 3). Meanwhile, the second line is its permutations, hence the second line for the example is (3 2 1). The permutation (3 2 1) can be written as $p = \left(\begin{smallmatrix}1&2&3\\3&2&1\end{smallmatrix}\right)$. From the example we could see that that this notation shows permutations quite clear and intuitive. The permutation which is represented by this notation can be drew as the fig. 2.1. In the fig. 2.1, it is obvious that the permutation is a mapping for set to itself, so that it can be defined as a function p. Take the example (3 2 1) again, the second line of it can be written as ($p(1)$,$p(2)$,$p(3)$) where $p(1) = 3$, $p(2) = 2$ and $p(3) = 1$, as well as, the whole formula for the example can be rewritten as $p = \left(\begin{smallmatrix}1&2&3\\p(1)&p(2)&p(3)\end{smallmatrix}\right)$.

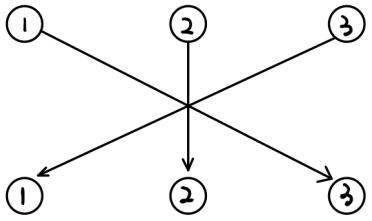


Figure 2.1: Two-line notation for permutation $p = (3\,2\,1)$,in which $p(1) = 3$, $p(2) = 2$ and $p(3) = 1$.

One-line notation

Generally, the first row in two-line notation is a list of set with ascending order, thus, the first line can be omitted and only retaining the second row, which called one-line notation.

According to the second line in two-line notation, this notation is $(p(1)\,p(2)\,P(3)\,...)$. This format has occurred more than once in the example mentioned above as (3 2 1).

Cycle notation

The permutation can be divided into disjoint cycles with the help of applying the permutation to members in set S again and again. The formula is described as $(x1\,p(x1)\,p(p(x1))\,...)$ which means that the arbitrary element $x1$ in S is chose as the start of a cycle and the result $p(x1)$ would be the next member in the cycle. The third one is applying p to $p(x1)$ again. Repeating steps until return the first element . We still using the permutation $p = (3\,2\,1)$ as example, it has two cycles (1 3) and (2). In the former cycle, the element 1 is applied to the and we get $p(1) = 3$, after that, the consequence of applying p to 3 is 1 again, thus, (1,3) is the final result. The same can be proved that (2) is the second cycle. The equation between one-line notation and cycle notation is $p = (3\,2\,1) = (1\,3)(2)$. The fig. 2.2 shows the cycle notation. This notation make a transparent structure of permutation.

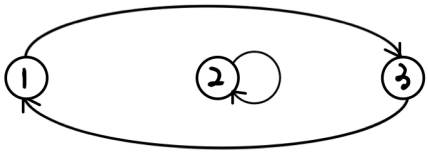


Figure 2.2: Cycle notation for permutation $p = (3\,2\,1) = (1\,3)(2)$.

2.2 Combinatorial maps

Combinatorial map using oriented subdivided objects to represent the graph embedded in the surface. At the beginning, the concept appeared for the planar graphs of polyhedral surfaces by J. Edmonds [Edm60] and named as “Constellations” [Jac69] and “rotation” [Rin12] by A. Jacques and Gerhard Ringel, respectively. The concept is extended from 2-dimensional objects to higher-dimensional objects and called “combinatorial maps” formally.

Sometimes, the computer needs to use data structure to indicate the subdivision of an object and the relation between each of them. The embedded graphs into the surfaces which are assumed to be connected, oriented and compact[nLa19], have three typical subdivision, vertices, edges and faces. Each of them are represented by a set of permutations of darts.

2.2.1 Darts

The combinatorial map is an edges-central model. All edges are divided into two parallel lines with arrow to indicate inverse directions, which called darts or half-edges. The main difference between darts and half-edges is the later one only segment the edges directly without using arrow and parallel lines. The fig. 2.2 explain this concept and two formats for it.

2.2.2 Edges

It is obvious that an edges can be represented by a pair of darts. Assume that there is a set of darts {0,1} and from the picture we found that dart “0” toward to “1”, on

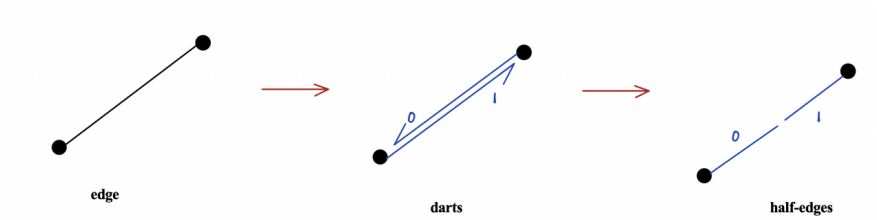


Figure 2.3: Two formats of the darts (half-edges) structure.

the contrary, dart "1" return to "0". This means the edges have the orientation for anti-clockwise, it shows a permutation of darts can be used to indicate an edges, as well. One-line notation for the example is (1 0), and convert it to the cycle notation as (0 1). The permutation of edges is denoted as α.

2.2.3 Vertices

A vertex consists of incidence darts surrounding it, which also follows the anti-clockwise and using σ denote this component. As the example in the fig. 2.4, the vertex can be indicated as (0 1 2) under cycle notation.

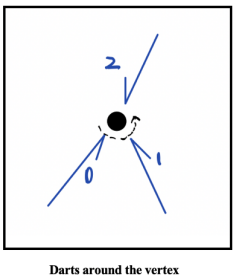


Figure 2.4: The darts surrounding a vertex.

2.2.4 Faces

There are two types of faces, the internal faces and the external faces. Both of them are the area surrounded by a series of darts, in other words, both of them have directions. The orientation of the internal faces is clockwise, while for the external ones is anti-clockwise. The signal ϕ is used to indicate the permutation of faces. There are two kinds of faces in the fig. 2.5. (a) has normal faces whose internal face is (0 4 2) and external face is (1 3 5). (b) is a loop which also has the internal face (0) and external one (1). All of the permutations that are mentioned in this section use cycle notation.

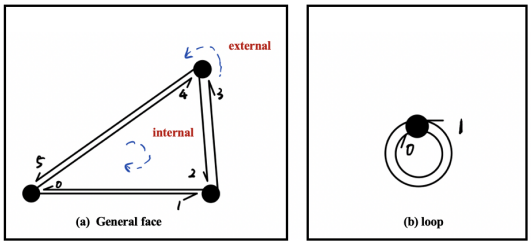


Figure 2.5: Faces

2.2.5 Planar maps

Primely, with the requirement of expressing the submission of planar graphs, a compactness topological model combinatorial map had been utilized. All of the three component vertices, edges and faces can be tidy and structural represented under an infinite set of darts. fig. 2.6 draws a typical planar map with format of combinatorial map $M = (D,\sigma,\alpha)$ in which D is the infinite set of darts. The length of D is 14.

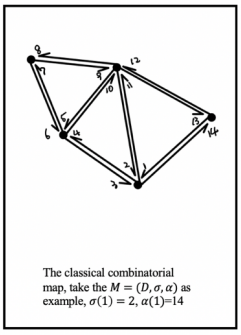


Figure 2.6: A classical combinatorial map for a planar map.

Firstly, the permutation of darts for vertices is $\sigma = (1\,2\,3)(4\,5\,6)(7\,8)(9\,10\,11\,12)(13\,14) = (2\,3\,1\,5\,6\,4\,8\,7\,10\,11\,12\,9\,14\,13)$, hence, $\sigma(1) = 2$, $\sigma(2) = 3$,etc. The edges permutation is $\alpha = (1\,14)(2\,11)(3\,4)(5\,10)(6\,7)(8\,9)(12\,13) = (14\,11\,4\,3\,10\,7\,6\,9\,8\,5\,2\,13\,12\,1)$ where $\alpha(1) = 14$, $\alpha(2) = 11$,etc. Actually, we have a triple of permutations $< \sigma,\alpha,\phi >$ on D, and the composition of them is identity, which means $\phi\alpha\sigma = id$. Accordingly, the permutation of faces is calculated as $\phi = \sigma o\alpha = (1\,11\,13)(2\,4\,10)(3\,14\,12\,8\,6)(5\,7\,9)$.

The orientation of each component is a quite strict constrain to define a combinatorial maps. The definition for discussing whether two maps are conjugation equivalent is only if they are isomorphic by a homeomorphism of the underlying surfaces. That is to say, judging the equivalent of two maps relies on if corresponding parts of those are obey the same rotation, rather than they have the same number of elements for the each component. In the fig. 2.7, although the three combinatorial maps have the same number of vertices, edges and faces, the first two a equivalent, and the last one is not the same with them.

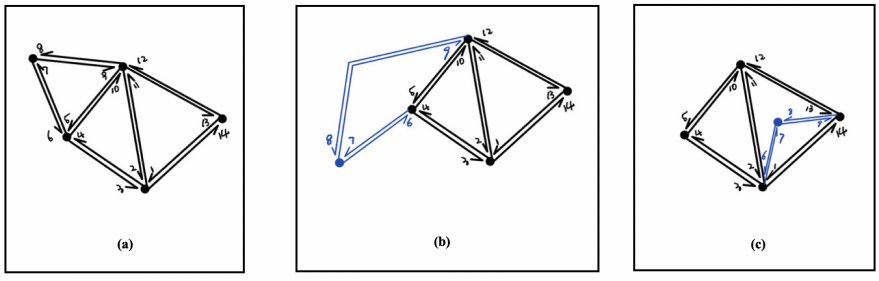


Figure 2.7: The equivalent of the combinatorial maps.

2.2.6 Genus

An orientable surface might have “holes” , and the number of “holes” it has called genus. For example, sphere as (a) in fig. 2.8 has 0 genus and torus as (b) which looks like a donut has 1genus. The genus g of a surface which the graph embedded in can be computed by

the Euler-Poincaré formula [Ric08]:$V - E + F = 2 - 2g$. The V, E and F are the number of vertices, edges and faces in the embedded graph, respectively. Substituting , V, E and F in the equation for features of combinatorial map as $c(\sigma) - c(\alpha) + c(\phi) = 2 - 2g$. The c is the function to calculate the count of the cycles for each component.

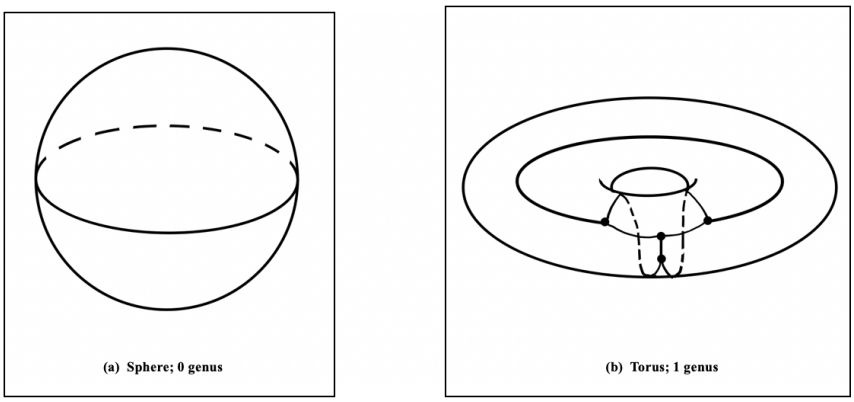


Figure 2.8: Two oriented surfaces.

2.2.7 Generalised maps

Planar graph is the graph embedded in a plane surface which has 0 genus. It is a crossing free model and the combinatorial map for it is also named 2-dimensional map. Gradually, the combinatorial map is discovered not only to represent the planar maps, but to indict more complicate objects with higher-dimension, even the non-orientation surfaces. The triple of permutations $< \sigma,\alpha,\phi >$ on set D is still used to describe the subdivision of the objects. Take the torus as the example, in fig. 2.9, the embedded graph on this oriented surface drew like (b), and the labels are half-edges. The permutation for edges is $\alpha = (1\,8)(2\,11)(3\,4)(5\,12)(6\,7)(9\,10)$ and for vertices is $\sigma = (1\,2\,3)(4\,5\,6)(7\,8\,9)(10\,11\,12)$. (c) is the isomorphic map for the graph.

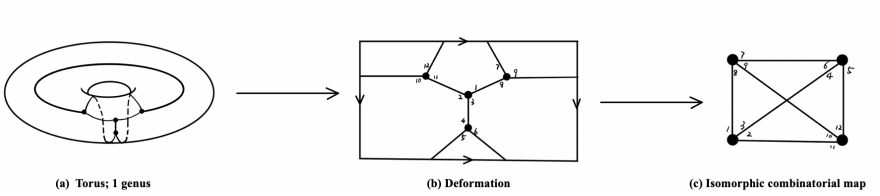


Figure 2.9: Combinatorial map for torus.

As far, the combinatorial maps can be used to indict a lot of objects with tidy and direct structure. It also involve into the field of graph drawing and hierarchical image partitioning [Hax05]. Some graph algorithm library like Pigale [Fra13] use the structure as the basic data structure.

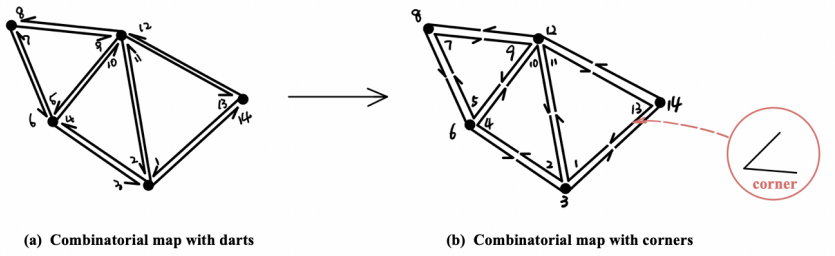
CHAPTER 3

Related work

The main work of the project is to find different approaches for visualisation of combinatorial maps, and using these techniques to build a tool for helping more people learn about the structure. From the previous content we knew the details and application of the combinatorial maps. This section is focus on the methods for drawing the structure.

3.1 Corners based method

Dainis Zeps and Paulis Kikusts [Zep12] came up with the idea in 2012. They introduced a new concept called corners replacing the half-edges in combinatorial maps. Corners are formed by two neighboring oriented edges around vertices, namely, incoming head and outgoing tale. The combinatorial map consists of a pair of permutations (R_V,R_F) with corners, in which R_V and R_F are vertices rotations and faces rotations. See fig. 3.1 the faces rotations and vertices rotations fixed from right to left. Consequently, we get $R_V = (1\,2\,3)(4\,5\,6)(7\,8)(9\,10\,11\,12)(13\,14)$, $R_F = (1\,11\,13)(2\,4\,10)(3\,14\,12\,8\,6)(5\,7\,9)$, and two edges rotations $R_{E1} = (1\,14)(2\,11)(3\,4)(5\,10)(6\,7)(8\,9)(12\,13)$ or
$R_{E2} = (1\,10)(2\,6)(3\,13)(4\,9)(5\,8)(7\,12)(11\,14)$.



(a) Combinatorial map with darts (b) Combinatorial map with corners

Figure 3.1: Combinatorial maps with corners.

Knot in maps shows the connection of corners which responses to the edges structure.Combinatorial maps involve in the partially normalised knots whose restriction is that all edges follow the same order, i.e., the less valued corners should come first, or reversely. There are three types of rotations in the combinatorial maps, so that they can be abstracted as tripe of adjacencies of corners as the image shows. The green label is vertex's vertex, the red labeled face vertex and the blue means edge face. Uniting each labeled vertex the corresponding adjacencies can be achieved. The knot occurs into the edge adjacency or r-adjacency. At the end, through fix knots of maps and drawing the rotations via corners, the whole maps could be settled.

In a word, this method is using the concept of corner to represent the rotations of each components. On the one hand, it produces a new angle to describe the combinatorial maps and drawing them. On the other hand, it makes the orientations constrain more clearly and directly which saves a lot of time to fix the ordered edges.

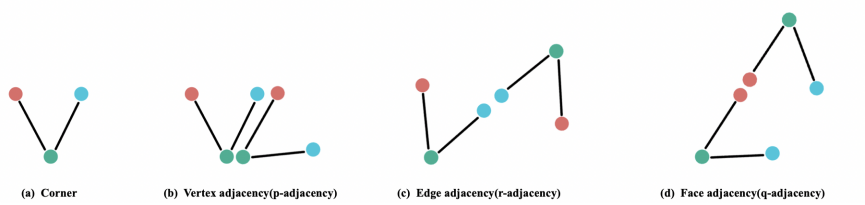


Figure 3.2: Three adjacencies

3.2 Vertices based method

For the previous method, the authors did not discuss the layout of knots or the corners, and they did not switch the process to a useful algorithm, which confuses people who want to use the method. To explore deeply for drawing the whole combinatorial map automatically on a certain canvas, we need to pay more attention on the layout of it. There are many algorithms concern about drawing planar graphs, for example, force-directed drawing algorithms summarised by Stephen G. Kobourov [Kob13]. This algorithms provide the details of vertices layout to avoid the cross. The algorithm simulates the celestial mechanics, each vertex is seen as a star which has repulsive force and attractive force with others. Therefore, the distance between vertices depends on those forces.

An idea[Hua19] for visualisation had been came up following the classical structure of combinatorial maps and the force-directed drawing algorithm. The steps as below:

a) Using force-directed algorithm sets the vertices.

b) Since the pair of darts or edges is the connection between two vertices, some darts could be settled after knowing the position of the vertices.

c) For placing the remaining darts according to the order around the vertices, we need find a base line and calculate the existed angles between base line and other darts for each vertex, and then put the rest darts by dividing the angles. Here are the details:

   – The base line is the first dart appeared in the vertex.

   – The equation $angle = atan2(v1.y,v1.x) - atan2(v2.y,v2.x)$ is to calculate the angle, if the result less than 0, it needs to return $angle = angle + 2 * \pi$. The $v1$ and $v2$ in equation mean the direction vectors about darts.

   – After knowing the rotating angle for a new dart, it can be places with the formula $d.x = v1.x * cos(angle_d) + v1.y * sin(angle_d) + v.x$ and $d.y = -v1.x * sin(angle_d) + v1.y * cos(angle_d) + v.y$ where d is the new dart and v is the vertex.

   – Note that the orientation of darts around the vertices is anti-clockwise.

d) Finally, connecting darts goes by the permutation of darts.

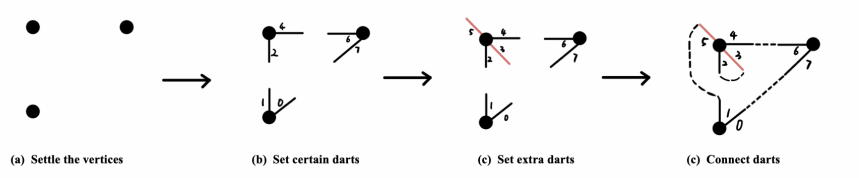


Figure 3.3: Process of vertices based method.

This method considers the whole process of drawing a map, however, it is not quite stable and have high time complexity. Its results are variant each time for the same maps, and the force-directed algorithm is the reason why this situation occurs. The fig. 3.4 shows the possible vertices positions for a map produced by the algorithm. This will produce different images for the same structure, which might confuse the users.

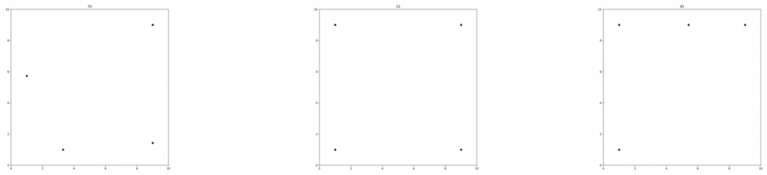


Figure 3.4: The different consequence of force-directed algorithm for a map.

CHAPTER 4

Specification

4.1 Calculating of combinatorial maps

Parsing the user input which is limited to the cycle notation.

Correctness The input can be parsed into the valid cycle notation of each permutation.

Reliability The tool can check the inputs at any time. If the input is illegal, there will be a space to show the warning information and the correct format of the input. All others operations must after checking and parsing the input.

Performance The correct result of parsing will display in the form of text and image.

Transforming cycle notation to one-line notation.

Correctness These two notations can be converted to each other.

Performance The correct result will display in the form of text and image.

Computation of faces by given the permutations of vertices and edges. This particularly depends on the equation $\phi = \sigma o \alpha$.

Correctness Any of three component can be educes by others, i.e.,$\phi\alpha\sigma = id$.

Performance The cycle notation and one-line of face will display in the form of text and image.

Generating a random combinatorial map by only given the number of darts (and vertices).

Correctness The graph should be connected, and the composition of permutations must be identity.

Performance Showing each component with text and image.

4.2 Visualising combinatorial maps

The tool using different methods to visualise the combinatorial maps.

Correctness The maps must be followed the constraints of the maps, that is, it must obey the orientation of each part. For the planar map, it must avoid the cross as much as possible. The whole structure must be connected and filled in the specific area without overflow.

Clarity User can have a clear mind when they look at the images. Each element must be discovered directly rather than distinguished for a long time. Especially the edges should not be cramped together and have an explicit identity.

Performance Showing the results in the right area and have an effective introduction. The result must show in the limited time.

Consistency The result of the certain method for a map must be the same at any time. This requires the methods must be stable and robust. Otherwise, the results might confuse the users and they will misunderstand about the combinatorial maps.

Aesthetics The whole tool must have a clear structure and comfort colour matching. The layout of the visualisation must be reliable. A good interface and drawing will attract user exploring more about the combinatorial maps.

4.3 Knowledge of combinatorial maps

The tool aims to pass more information about combinatorial maps for whether they are beginners or experts in the relative area. A introduction of the maps and tool is necessary.

Correctness The background of the combinatorial maps must be correct which cannot be misleading.

Performance A introduction page to describe the tool and the structure. Other information like genus number can be shown in the visualisation part.

CHAPTER 5

Implementation

This section covers the two parts, the implement for visualisation of combinatorial maps and package them as a tool.

5.1 Language

For designing the application as a tool, the JAVASCRIPT is used to the implement all the function. Therefore this is a website tool which is convenient for people to search no matter when and where they are. A library called D3.js [Bos19] is used to draw image as format of SVG.

5.2 Visualisation Methods

The methods are still concentrate on the layout for drawing a combinatorial maps. The features of the maps are discovered as the base sketch for the completely graphical model. The base data structure of the permutation is array, that is, all types of the permutations are kept as array and note the cycle notation as $*C$ and one-line notation as $*P$. For example, a permutation of vertices is (1 0 3 2), VC = [[0,1],[2,3]] and VP = [1,0,2,3] are cycle notation and one-line notation of it , respectively.

5.2.1 Tree Based Method

Spanning trees

The spanning tree is general subgraph of connected graphs. There are two kinds of spanning tree, one contains the maximal edges of the maps without cycles, and another one connects all the vertices through a single path. It is obvious that there is not only one spanning tree of a map.

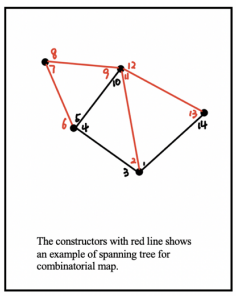


Figure 5.1: The spanning tree of a combinatorial maps.

Depth First Searching Algorithm

Depth first searching algorithm(DFS for short) is a route from an arbitrary vertex to another one. In this problem the DFS is used to find the spanning trees of the maps and validate whether the structure is connected or not. The start point is the vertex who has the least valued dart. There are some steps for DFS in this problem:

a) Using five list for storing the unsearched vertices V and darts D , as well as recording the possible searching path P including the first element of V (removing the element from , as well) and valid path via all vertices PV or some edges PE. An assistant array DTV is used to record corresponding vertices of darts.

b) Pop out the last element s in V as the start point of the current segment.

c) If all darts of the have been searched, continue from step b).

d) Otherwise, push s in P again. Find the first unsearched dart d_s in $VC[s]$ (cycle notation for vertices permutation), and get the next dart d_t via (one-line notation for edges permutation), so that the target vertex t is $DTV[d_t]$. Delete d_s and d_t in D.

e) If t is in V, remove it and push the new path in PE and PV, as the same time, add t to the tale of P.

f) Repeat the steps b) to e) until any one of V, D and P is empty.

g) If V is empty, return the truth result with PE and PV, else, return false and other results are empty.

```
function isConnected(VC, EP){
    // The input of the function is the cycle notation of permutation of vertices VC
    // and the one-line permutaion of darts EP.

    var n = VC.length, m = EP.length;
    var  vertices = new Array(n)*;// The set records the unsearched vertices
    var  darts = new Array(m)*;//The set records the unsearched darts
    var  route = [vertices.pop(0)]*;//The possible searching path
    var  path_v = []; // The valid path through vertices
    var  path_d = [];// The valid path through darts
    var  darts_to_vertices= DartsToVertices(VC, D)*;
     // Recording the darts to corresponding vertices

    while(vertices != null && darts != null && route!=null)*{
        var start = route.pop();//Find the beginning vertex

        // All darts of the vertex have been searched
        if (VC[start] ==null) *{
            continue;
        }
        // Search for end vertex
        else{
            route.push(start);
            start_dart=VC[start].pop(0)*;
            target_dart =EP[start_dart];
            target = darts_to_vertices(target_dart );
            darts.delete(start_dart,target_dart)*;
            VC[target].delete(target_dart )*;
            if (target is in vertices*){
                route.push(target);
                path_v.push([start,target]);
                path_d.push([start_dart, target_dart]);
                vertices.delete(target)*;
            }
         }
    }
    if (nodes==null*) { return {result: true,vpath:path_v,dpath:path_d};}
    else { return {result:false, vpath:null_v,dpath:null};}}
}
```

Figure 5.2: The rough implementation of DFS by JAVASCRIPT, the lines with * are not the standard JAVASCRIPT code.

Tree drawing algorithm

Recent paper by Christoph Buchheim et al [Buc02] introducing a method for drawing a n-ary trees with $O(n)$ time complexity. It is an improvement of the method for drawing binary trees proved by Edward Reingold and John Tilford [Rei81]. Bill Mill's article [MIL] described the evolution of the algorithm. There are six principles should be considered when draw a tree:

a) The edges cannot be cross with each other.

b) The nodes in the same depth should be kept in the same height.

c) The whole tree should be as narrow as possible.

d) The vertical position of parent nodes is the middle of their children.

e) Wherever the subtree lies in a tree, it should move the same distance with its parents.

f) For the n-ary tree, the children should divide the space evenly for their parents.

For obeying the first four principles, the algorithm searches the tree from bottom to up for settle the nodes, namely, post-order traversal of the tree. When maintain the fourth principle, the parent nodes move to right and the right child of it should move toward right also, which leads to a bad result that right children in a node will cover the left children of the later nodes, in another word, some of the nodes will be disappeared. The principle e) aims to solve the problem. In order to achieve the principle, the concepts of mod, contours and threads had been proposed. The mod records the distance that the current tree should be moved, and then, moving each tree with the accumulated mod value from its ancestors to itself. This concept can solve the problem mentioned above, but it cost the running time as it needs two times to search the tree. The contours and threads make the algorithm less complexity, which looks each subtree as a block.The contour is the leftmost side of trees (minimum x-coordinate) called *left contours* or the rightmost side of trees(maximum x-coordinate) called *right contours* as the example in the fig. 5.3. The threads are the link in the contours if the two nodes have no relationship.

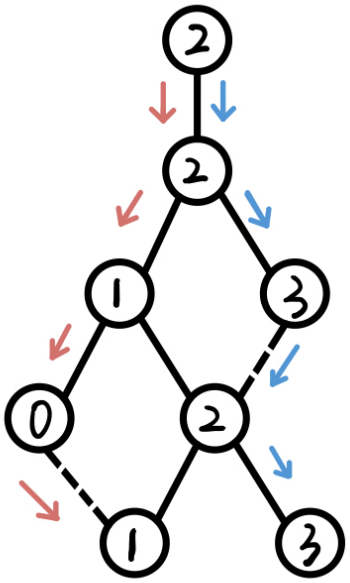


Figure 5.3: The red lines show the left contour [2,2,1,0,1], and blue lines are right contour [2,2,3,2,3]. The dash lines are the threads.

The thread can reduce the complexity and avoid the conflict between the subtrees, which helps to calculate accumulated mod value during the first walk of the tree. The last principle is specifically set for the n-ary tree. Since each parent might has more than two children, the left contour and right contour cannot totally deal with the conflict the problem. The concept of the ancestor is used to decide which left siblings conflicts with current subtree.

```
function drawTree(path_v,path_d,VC, EC,width,height){
    // The inputs of the function are path through vertices and darts,
    //and permutation of vertices and edges(cycle notation)

    var basic_tree = genBasicTree(path_v); //Convert the path to the tree format
    var tree =  new Tree(basic_tree); //Generate the tree structure for the algorithm
    var h = height/max_height, w=width/max_width;

    // Implementation of tree drawing algorithm
    setPrimCoordination(tree,w); // The tree drawing algorithm for set x-coordinate of nodes
    var range = setTargetCoordination(tree,h); // Check whether the structure out of range
    if (out of range || not in the middle){
       //offset is the variable range or the distance between x-coordination of root and middle of screen
       setTargetCoordination(tree,offset);
    }

    // Drawing tree on canvas
    drawTreeSVG(tree, path_v,path_d,VC, EC,width,height,h);
}

function setPrimCoordination(tree, distance=1){
    if (tree.children==null){
       // If  leaves, the x-coordination would be distance/2(for first children)
       //or add x-coordination of its left sibling  with distance
       tree.x = tree.lb.x+distance;
    }
    else {
       var def_ancestor = firt child of tree;
       for each child in tree do{
          child=setPrimCoordination(child,distance);
          // Find the ancestor of the current subtree
          //The setAncestor function involves moveSubtree function
          def_ancestor = setAncestor(child,def_ancestor, distance);
       }
       exe_shift(tree);
       var midpoint = middle position of tree's children
       if  (tree has left brother) { tree.x=tree.lb.x+distance; tree.mod=tree.x-midpoint;}
       else { tree.x=midpoint; }
    }
}
```

Figure 5.4: The constructor of tree drawing function and tree drawing algorithm in the JAVASCRIPT implementation.

The tree drawing algorithm just concerns on how to calculate the x-coordination, rather than fill the structure into a certain canvas. It is necessary that computing the vertical and horizon gap between each nodes. The horizon gap can be calculate directly, just using height of canvas divided by max depth of the tree. The vertical gap can be conducted by the same way, while, it is more difficult to compute the max width for the tree. The weighted average method is used to solve the problem. Firstly, generating a list *wide* which records the width for each level of trees. Sorted *wide* with ascending order as the next step. Finally, the weight of each item in *wide* is the serial number of the item divided by the length of *wide*, and the computing the weighted average of items.

The function *drawTreeSVG()* in fig. 5.4 is to do the drawing work on the canvas with D3.js. The coordination of the nodes and the path of links should be collected as a list, after that, drawing the model in a specific area. The fig. 5.5 shows the result of the implementation of tree drawing algorithm.

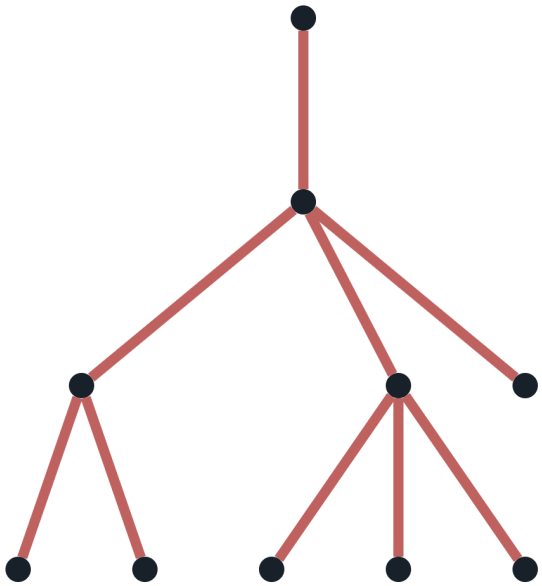


Figure 5.5: Result of the algorithm.

We have already got the tree sketch of the combinatorial maps. Some of darts are fixed within the tree structure, while the rest should be settled according to the existed darts. It uses the same approach to calculate the coordination of the endpoints of remaining darts in *vertices based method*.

a) Find two darts around the same vertices.

b) If there is no darts between them, return a).

c) Otherwise, calculate the angle between them using $angle = atan2(v1.y,v1.x) - atan2(v2.y,v2.x)$ (and $angle = angle + 2 * \pi$, if $angle < 0$).

d) N is the number of darts between them, and the radian rad between each nodes is the $angle/(N+1)$. The less valued darts is seen as the base line, and $v1$ is the direction vector for base line.

e) For $n = 1\,to\,N$, $d_n.x = v1.x * cos(rad * n) + v1.y * sin(rad * n) + v.x$ and $d_n.y = -v1.x * sin(rad * n) + v1.y * cos(rad * n) + v.y$ where v is the vertex which the darts surrounding.

f) Repeat the above steps until all the darts are placed.

g) Connect the new darts rely on the permutation of edges.

Bezier curves

For the purpose of receiving a sleek and smooth edge, the connection using curve line rather the straight. Bezier curve is an algorithm to calculate a curve between to endpoints. Unlike other splines, the Bezier curve must through the start and target points. The degree of curvature depends on the control point. Second-order Bezier curve has one control point and third-order Bezier curve has two. In this method, the third-order Bezier curve is being used. The control points are the extending along the direction of corresponding darts.

Figure 5.6: Result of the method for visualisation of the combinatorial map about Planar Tetrahedron(planar map).

Problem

The result in fig. 5.6 is a planar maps. It has been mentioned that the planar maps is a cross-free model, while in the fig. 5.6, the crossing occurs. As the same time, for the Petersen graph in the fig. 5.7 has 2 genus, which means that it is not necessary to avoid the cross as much as possible.

Figure 5.7: Result of the method for visualisation of the combinatorial map about Petersen graph.

However, the edges look so mass and entangled that every edges cannot be read directly. The traditional way is label the corresponding number in the end of each half-edges but this makes the image more disordered and unclear. The tips and highlight are added to

indicate each vertices and edges like fig. 5.8.

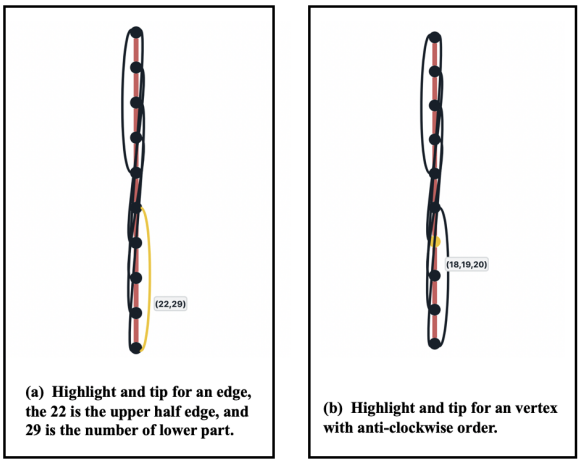


Figure 5.8: The tips and highlights example.

Improve version

The main issue about hardly avoiding the cross for planar map is the position or direction of darts around a vertex. The spanning trees (as the result of DFS algorithm) of most combinatorial maps are a straight line, like the red part in the examples. When settle the extra darts , in the previous method, it is just divide the angles between excited darts. This leads to absolutely opposite directions for a two darts which should be connected. Therefore, the result of the connection using Bezier curve will through the middle line rather than cross the line.

On behalf of improving the darts angle problem, an idea is to involve the darts as virtual nodes into the tree structure. That is to say, each vertex has darts essentially and it might also have children in the spanning tree structure, so as to drawing the darts as the virtual children for a vertex in the tree structure. After that, one or some of the darts are connect to the other darts who are belongs to the children vertices of the current one in the original tree model. The fig. 5.9 shows the transformation. The circles filled white are types of virtual nodes, one is the joint of two darts in the tree structure, and another one is the darts of the vertices.

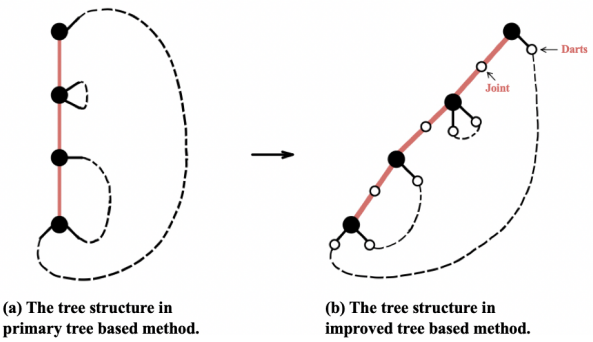


Figure 5.9: The compare between tree structures in two methods.

Redefining the tree structure is not enough to avoid the cross, some situation of the connection have to discuss too. (Note: Reverse direction of a dart in all situations means doing the mirror inversion according to the $x = x_d$ where x_d is the x-coordination of the

dart.)

a) If the less x-coordination node towards left and larger one to the right, the curve should through the bottom.

b) The extension of two darts are cross. This station still arrives the bottom but the direction of both are reversed.

c) If any of the two darts is the middle children of its father, the direction of the middle dart is horizon. This connection also go via bottom.

d) The last situation is both darts are toward the same direction. For the right side, if the less y-coordination node has larger x-coordination, reverse the lower node and connect them through the bottom, otherwise, the connecting line will go through the middle node of two darts. It is the same for the right side.

e) Last but not least, counting the numbers of connections through bottom, left and right, respectively. Using corresponding lines to divide the each space. All are using 3-order Bezier curves.

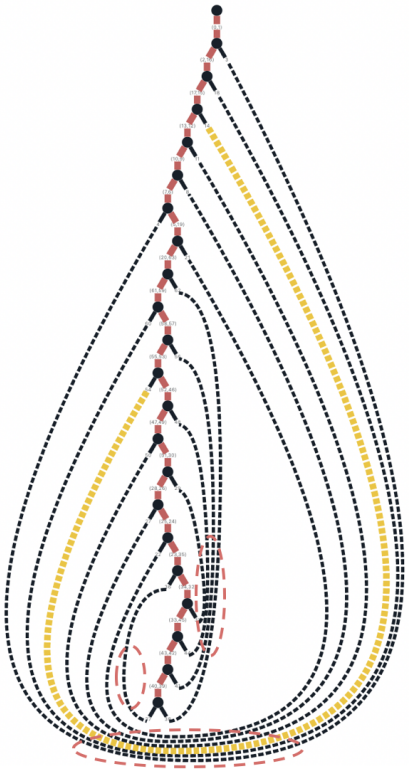


Figure 5.10: The result of improved tree based method for rooted dodecahedron graph.

This method has greatly reducing the cross, and the oval with dash line demonstrate the essence of the ideas. As the mean time, it do not need extra time to calculate the position of darts which out of the tree structure. Due to dividing the fix size of the area, the gap between lines are narrow. Highlight the selected line can produce a simple and convenient recognition of it.

5.2.2 Face Based algorithm

This approach comes from the dual of map. If it succeed, it might recover the surfaces which the combinatorial maps embedded in.

Dual

Any embedded graph has its dual graph in the same surface. The vertices of the dual structure is the centre of the corresponding faces in the combinatorial map. The darts around the vertices in the dual map are the darts surrounding the primary faces.

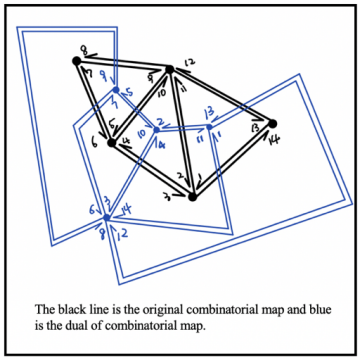


Figure 5.11: The dual map of a combinatorial map.

The core of the idea which comes from this structure is generating every faces of the combinatorial maps. Gluing the faces together by connecting the corresponding darts under the edges permutation. With the help of folding algorithm, the drawing results are improving from 2-dimensional to 3-dimensional which shows the underlying surfaces that the maps embedded in.

Visualisation

The fundamental task is to drawing the faces without gluing. It is necessary that separating the screen into N_f parts where N_f is the number of faces. Firstly, judging whether the N_f can be squared. If possible, the result is denoted as n_l, and the screen splits into n_l rows and n_l columns. If it is not , dividing the canvas into 3 lines and $abs(N_f/3) + 1$ columns. The next step is drawing each faces, it is easy to find that each face is a polygon, bilateral(with two vertices) or loop(with one vertex).

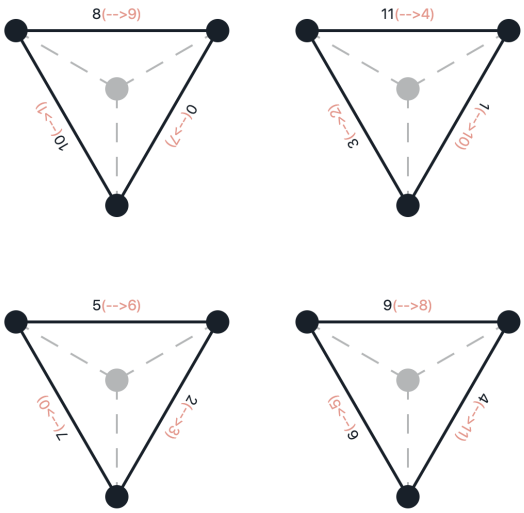


Figure 5.12: The visualising of a combinatorial map for planar tetrahedron by faces based method.

The labels of the darts are the darts number (black words) and the red numbers indicate

which darts to connect with. The grey nodes and dash lines are the corresponding parts in dual maps.

5.3 Tool Structure

A tool aims to package the methods for visualisation of combinatorial maps as a website application. This section covers the structure of the tool including two pages: introduction and visualiser, as well as introducing how does each section work. The style for the layout of pages uses Bootstrap.css [MIT19a], and the icon in the pages also from the bootstrap opening icon set [MIT19b].

5.3.1 Introduction

The purposes of the tool are not only displaying the combinatorial maps, but also helping the researchers or beginners reward the knowledge of it. Therefore, an introduction page is necessary to deliver the basic background of maps, the features which the visualiser involved, and the other informations for example what algorithm are implemented. This is also seen as the starting page.

Head

The head of the page is to describe why the tool should be created and provide the links. The right button 'Get start' goes to the visualiser page if people want to experience the tool directly, while the left one stays at the local page to let people learn more about the theory knowledge.

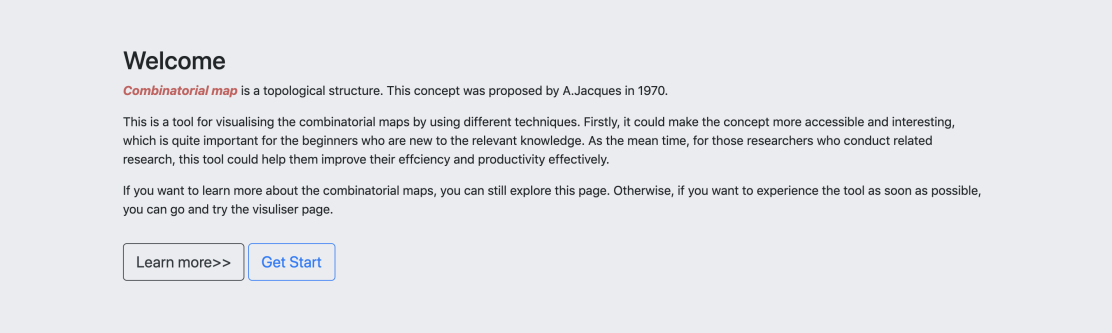


Figure 5.13: The head of the introduction page.

Main content

The side bar of the main content is a table of content plugin [Fel19]. From the guideline of the content, we can see that the there are three sections about combinatorial maps, visualiser and relevant algorithms. If people want to know more informations, just click the item and it will link to the specific area in the page. It can jump to the visualiser to another page when click icon beside the subtitle named visualiser.

5.3.2 Visualiser

This page plays an essential role in the whole project. It is divided into three parts, the introduction, input and output.

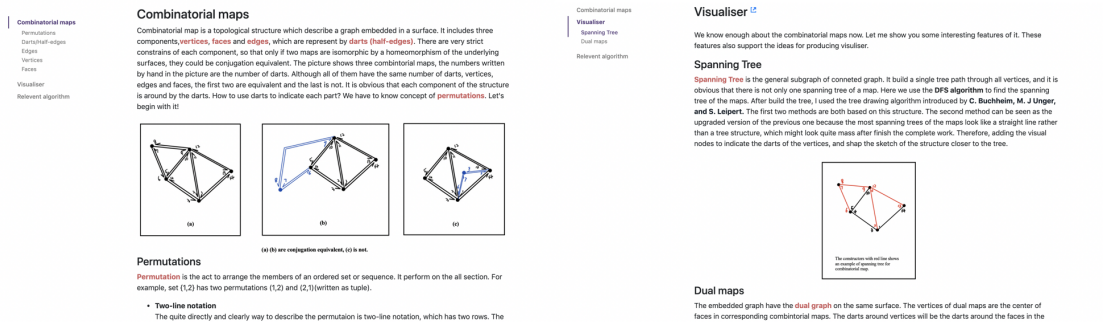


Figure 5.14: The content of the introduction page.

Introduction

The page has the introduction part, as well. The introduction can help the users have a clear mind about the structure of page and know the operations and limitation of each parts. The button 'learn more' means return to the introduction page to learn the knowledges of combinatorial maps and the tool. The 'get start' is starting to try the visualiser.

At the beginning

The whole page is split into two parts. The input area and the ouput area.

- **input**
  There are three types of input for users to experience. You can try the **examples** first, which can help you to understand how it works more directly and easily. Then, if you have some specific idea want to try, you can use **specific** tab. This part has some special requirements:
  - You need to input the **darts number** first, and it must the multiple of 2,like 4.
  - The vertices and edges are also required, and indexes of they need a series of natural numbers without duplicate, which start at 0. The input **format** is strict as {(_,_),(_,_),..}. For the **vertices** you can give more than one numbers in one parenthesis, like{(0),(1,2,3)}. However, you must add a pair of numbers for the **edges**.

  If you want to explore more you can use the **random** section. The only required input is darts number, which also should be the multiple of 2.
- **output**
  After clicking the create button, the tool begins to work. The **permutations** of each component will be generated first. The cycle notation and one-line notation will output as array, and drawing them as cycles and two-line notation. Some useful information will be indicated after that, take the **genus** number as is example. The genus meanings the "holes" of an (orientable) surface, for instance, the sphere has 0 genus and donut has 1 genus. Finally, you will see the images based on the three visulaising ideas and you can view more by click "more details" button. By the way, you can try to interactive with them, for example zoom the space or click the nodes to zoome one certain area.(If you do not have the related background of combinatorial maps, you can go to the home page to learn about it.) Now, you can start your journey and enjoy yourself!

Get Start>> Learn more

Figure 5.15: The introduction of visualiser page.

Input

There are three type of input methods for users to choose, examples, specific and random.

a) Users can select an arbitrary example as input to experience how tools work. All the examples are classical combinatorial maps including planar and non-planar maps. Click the 'create' button, the tool start to do the visualisation.

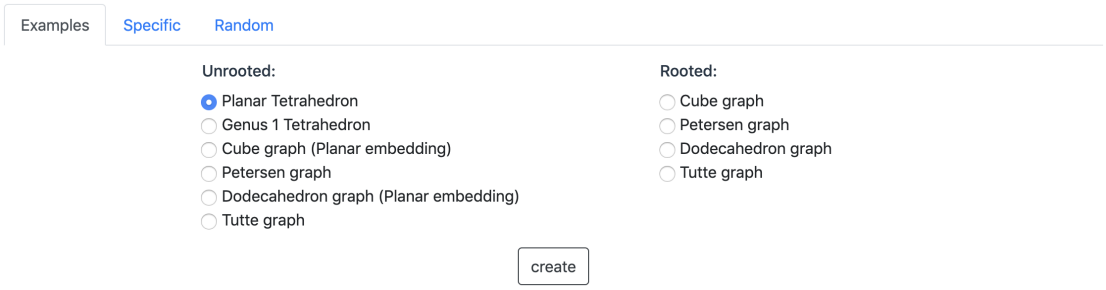


Figure 5.16: The examples tab

b) If users have a certain sample which is not included in the given examples, he or she can choose the 'specific' tab. Three pieces of information are required, the darts number and the permutation of vertices and edges with cycle notation. The darts number should be filled at first and correct,otherwise, users cannot continue doing other operations.

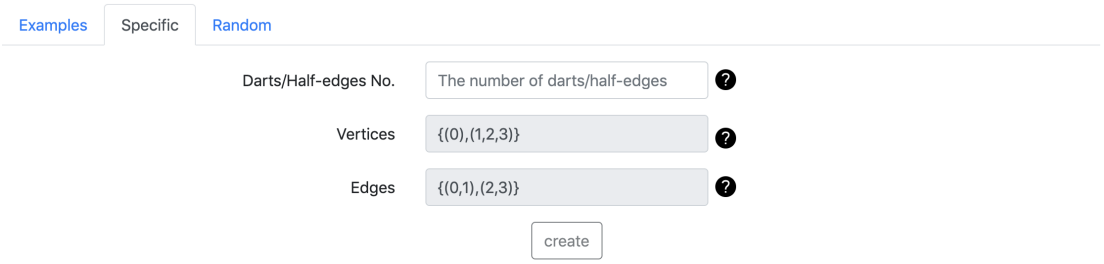


Figure 5.17: The specific tab

Some conditions are used to validate whether the input is correct or not. A JAVASCRIPT library called Vue.js[You18] is used to do the real-time validation and data communication. Firstly, for the darts number, it must be a number that is multiple of 2. Secondly, the permutation of vertices and edges have their own format the users should obey, for example, the numbers of each permutation must be a series of natural number start from 0 and the largest number must be the darts number minus 1. For the vertices, the numbers of elements in the parentheses must more than 1, while, in edges permutation, it can only have two numbers. These constraints provide a structural clue to parse the inputs whose data type is string. For parsing the string to a readable and operational data type, a regular expression is used, for instances, the expression '/^\{(\([0-9]+(,[0-9]+)\))(,\([0-9]+(,[0-9]+)\))*}$/' matches the format of inputted edges. The real-time validation will check the input at every time when users change the content. If the wrong input, the system will promote the users with an error massage and provide a right examples. The users can also click the '?' button to check the requirement of the inputs.

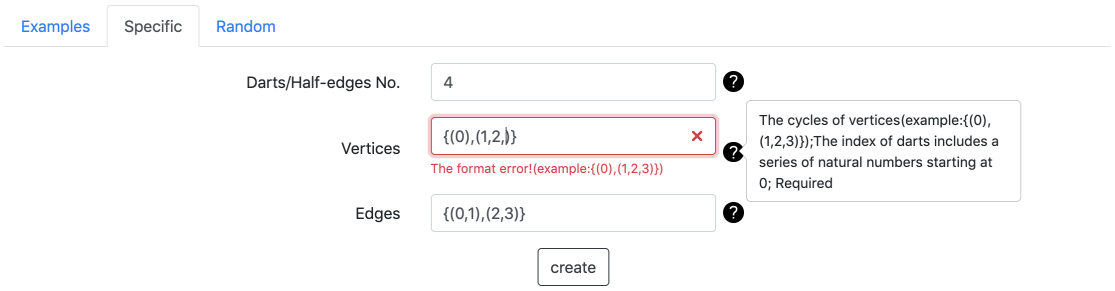


Figure 5.18: The error massage and notes.

c) The last tab is for users who do not have a specific example of combinatorial maps or those who just want to seek various results of maps with the same darts number, hence the darts number is required. Meanwhile, users can decide whether needs to limit the number of vertices for the map and the number must not more than darts number.

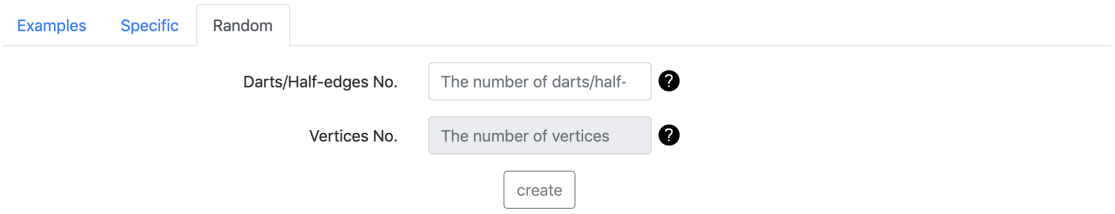


Figure 5.19: The random tab

Regardless of whether the user has entered the number of vertices, the entire process starts with generating a random permutation of edges *genRandomEdges()*: 1.Defining a set of darts D; 2. Choosing arbitrary two members of it and delete them; 3. Pushing selections as array into the edges array EC; 4.Repeating 2 and 3 steps until D is empty.

Creating the permutation of vertices is the next step, which depends on whether the users enter the number of vertices. If the input is empty, using Fisher-Yates shuffle algorithm [Fis43] to generate a vertices permutation with one-line notation *genRandomVerticesPerm()*. The logic of the function is as below: 1. Declaring a set VP whose length is the number of darts n_d and the elements follow the natural order; 2.For each member whose index is i, selecting a random number j from 0 to $n_d - 1$; 3. Swapping the elements $VP[i]$ and $VP[j]$.

Another function *genRandomVerticesByNum()* is used when the input is not empty as n_c. At the beginning, defining a set VC which has n_c members of empty array. The core of the function is a loop with period n_d. For each time n, randomly choosing a number r_d between 0 and $n_c - 1$. Adding n to the array $VC[r_d]$. Redefining VC if any of the element is still empty. Actually the random function the JAVASCRIPT defined is useless and unfair, thus, a new random generator is defined. Even though, it still need to verify that whether the VC contains the empty elements.

The function *convertToPermutations()* aims to convert the cycle notation to one-line notation and *convertToCycles()* inverses the process. Finally, using the function

isConnection() mentioned above validates the whether the map is connected. If it is not connected, restarting from the first step or continuing to the next step when it is connected.

```
var  dnum = dartsNmber, vnum= verticesNumber
Do {
    EC = genRandomEdges(dnum);
    EP = convertToPermutations(EC);
    if (!vmun){
       VP = genRandomVerticesPerm(dnum);
       VC =  convertToCycles(VP);
    }
    else{
       VC = genRandomVerticesByNum(dnum,vnum);
       VP =  convertToCycles(VC);
    }
    result = isConnected(VC,EP);
}while(!result.res)
…
```

Figure 5.20: The construct of generating a random combinatorial map with the JAVASCRIPT implementation.

Output

The output includes describing and visualising each component, displaying other information and showing the results of the approaches.

a) Components of a combinatorial map. It is known that the permutation is used to indicate the permutations. In order to supporting more specific details, every notations of permutations have been visualised. The fig. 5.21 shows the results of each components, the nodes are the darts. The different colours can distinguish the components easily. It is difficult to identify the minutiae of the nodes and edges and zooming the space by click the nodes or other ways can solve the problem. The icon at the end of permutations is to reset the canvas.

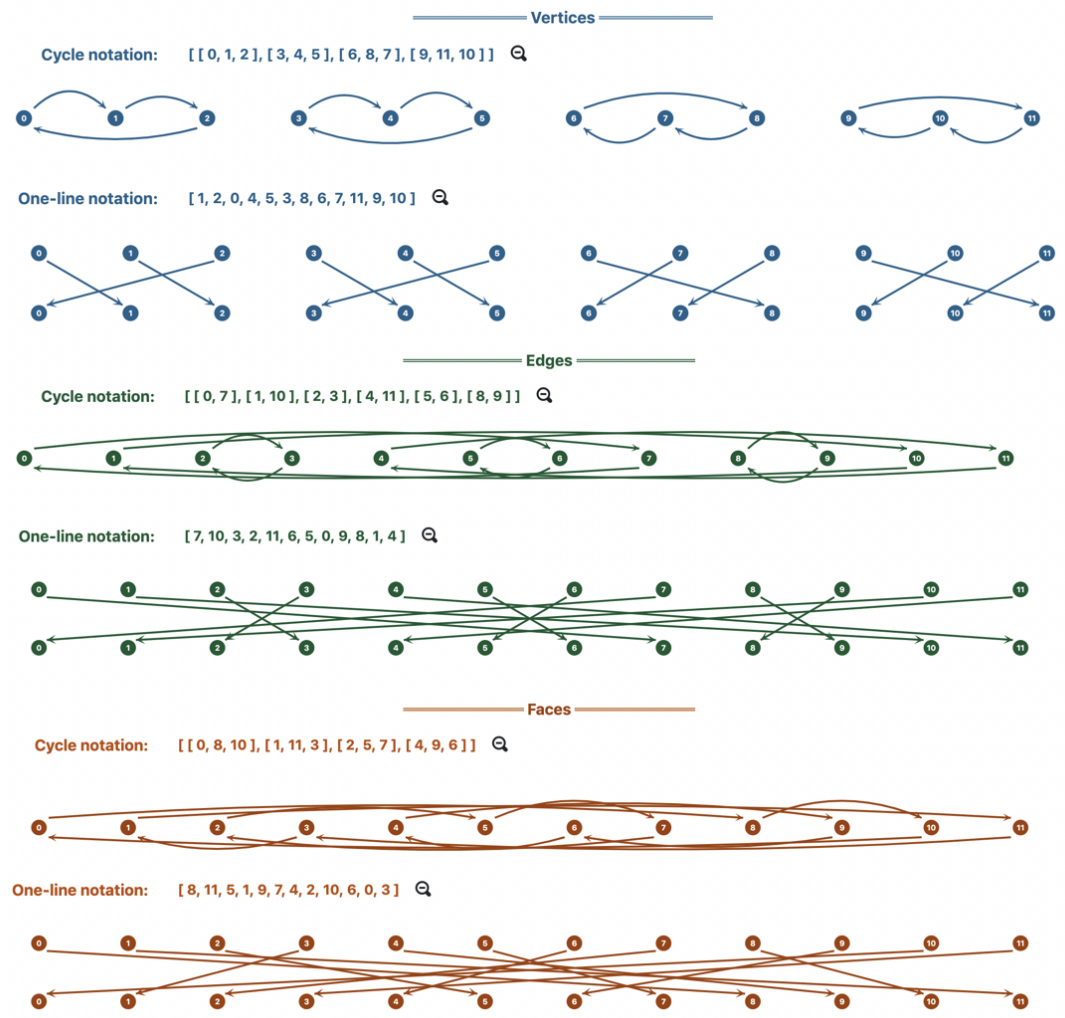


Figure 5.21: Visualisation of each component

b) Four characters faces number, vertices number, darts number and genus number are supplied in the 'other characters' area.

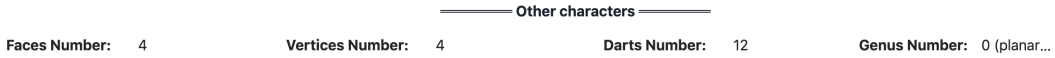


Figure 5.22: Other information

c) The bottom of the page is the results of approaches. Every results show as PNG format, and hover effect cover on the images tell the idea of the methods. If users want to see the details, clicking the 'MORE DETAILS' button and an interface will pop out. A real SVG image displays on the interface and the necessary information is wrote at the side of it.

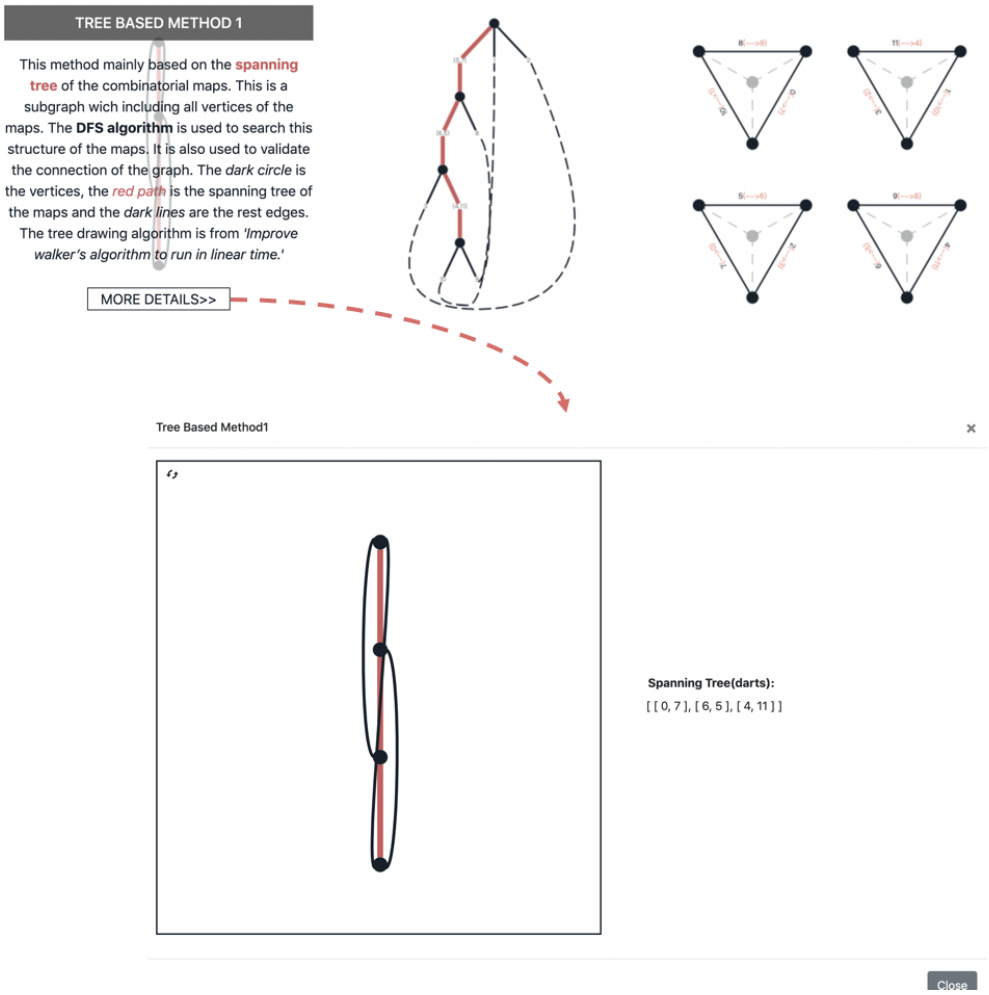


Figure 5.23: Results of methods

CHAPTER 6

Evaluation

6.1 Testing the visualisation method

The visualising methods is the main work of the project, thus, comparing each method and discovering the problems of the methods are important.

6.1.1 Cross in planar map

The most essential feature of planar graph is no cross inside of the graph. Therefore, this is the primary assessment criterion for the methods when generate a planar map.

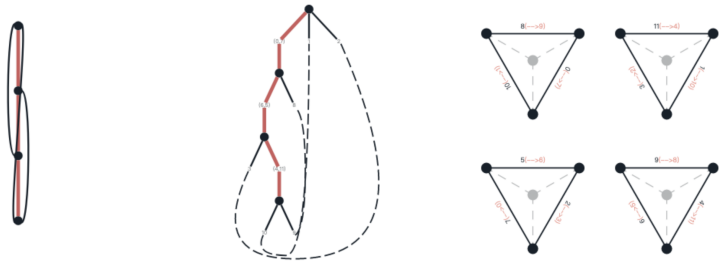


Figure 6.1: Results of planar Tetrahedron.

The fig. 6.1 shows a simple planar map called planar Tetrahedron, it is obvious that the last two methods work well and avoiding the cross successfully, though the gap between lines in the second method is too narrow. However, the first approach cannot handle the problem completely.

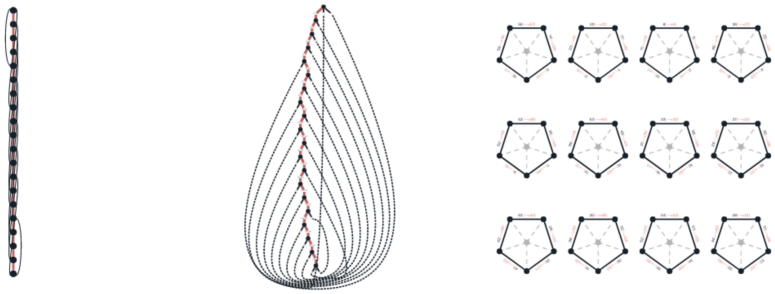


Figure 6.2: Results of dodecahedron graph.

The solutions of a larger map for dodecahedron graph display in the fig. 6.2. For this map, the first map becomes mass and unclear, and the last method is still tidy without cross. The second method, however, is not up to standard because the cross appear in the two of curves.

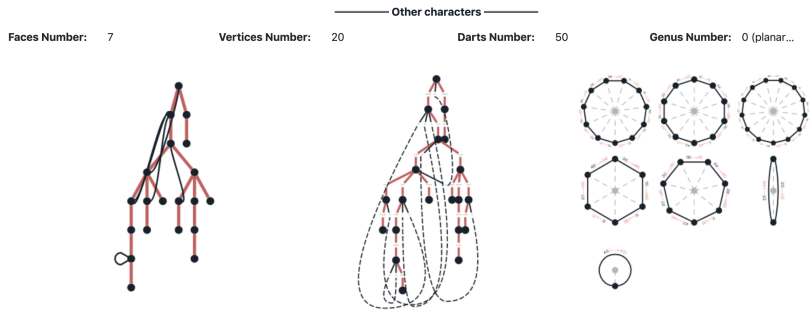


Figure 6.3: Results of a random planar map.

Besides the example, the fig. 6.3 gives results of a random planar maps. The darts number of the map is 50. The vertices number is fixed to produce a planar map with high probability. The genus number is 0, thus this is a planar map. After observing the results, the result is the same with the last example that the cross occurs in the first two results but not exists in last one.

6.1.2 Time consumption

The time consumption is a condition to estimate whether a method or an algorithm is better for a certain problem.

	Time span(ms)	Tree based method #1	Tree based method #2	Face based method
Unrooted	Planar Tetrahedron	4	5	5
	Genus 1 Tetrahedron	5	6	6
	Cube graph (Planar embedding)	5	8	8
	Petersen graph	6	8	8.5
Rooted	Cube graph	5.5	8	9
	Petersen graph	6.5	9	9.5
	Dodecahedron graph	9	13	15
	Tutte graph	11	22	29

Figure 6.4: The time consuming of the methods for different maps.

We just test this features of each approach on the examples, cause the time for the same method might be a little different for each running. In order to maintain the accuracy of the results, the consequence comes from the average of 10 times testing with the same map. From the result, first tree based method use the least time. The time span of the improved tree based method and the faces based method almost the same, but the later method spends more time.

6.1.3 Clarity

The requirement of the clarity for a visualising result is that users can analysis the output directly and the edges should not be cramped together and have an explicit identity. Firstly, trying a simple combinatorial map whose darts number is 8. Every drawing results of the map is tidy and transparent.

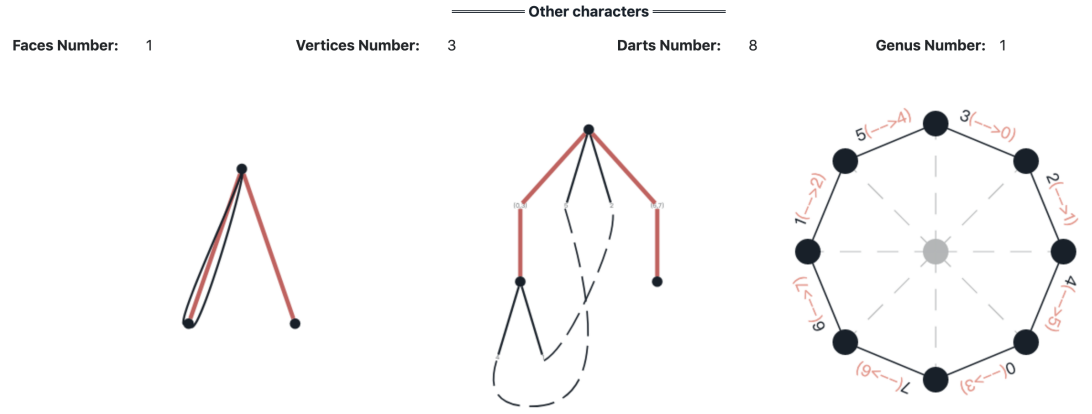


Figure 6.5: A random map with 8 darts.

Drawing a larger combinatorial maps based on the 100 darts. The edges cramped together and hardly to recognise the structure of the map for the first method. The second method still retain a beautiful and complete structure. The most neat result is the last one.

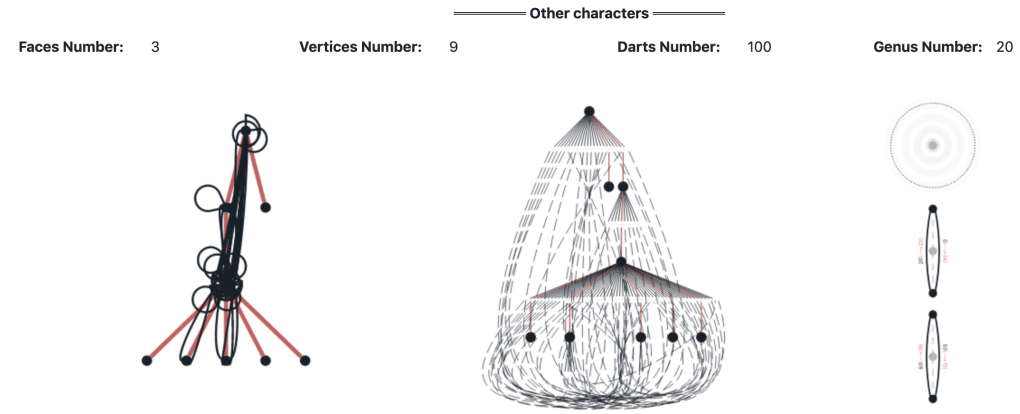


Figure 6.6: A random map with 100 darts.

The fig. 6.7 shows a combinatorial map having 1000 darts and 50 vertices. It cannot be recognised any plot of structure, even the tree structure in the first result. The last two still maintain a featly and legible structure. The only one problem for the last method is that it just shows each face and indicates the darts number and the connection information at the darts around the faces. The number is too small to read for a larger map, and readers cannot joint every faces together.

6.1.4 Usability

All of the methods can generate a combinatorial maps successfully and the results are correct. For improving the aesthetics problem in first image, the highlight and tips are used to indicate the edges and vertices, though it just an assistant operation without changing the result. The highlight also occurs in the second method, since the edges are

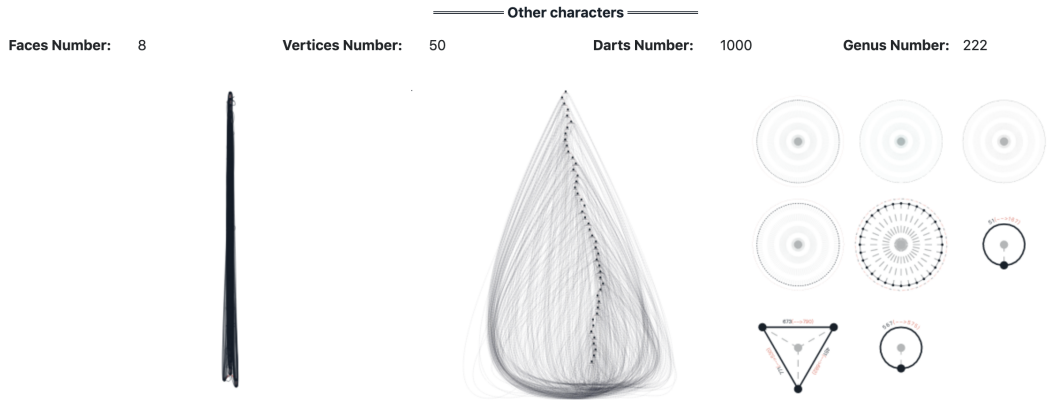


Figure 6.7: A random map with 1000 darts.

so slim that it difficult to spot the specific path of it in a large maps. The SVG images can be zoomed through clicking the nodes in the them or some other operations with mouse like scrolling the mouse in the space. A reset button is used to recover the whole image.

6.2 Testing the system

After estimate the visualisation approaches, the system should be tested according to whether the specification in chapter 4 is satisfied. This also help to point out the bugs in the current system.

6.2.1 Calculating of combinatorial maps

Parsing the user input which is limited to the cycle notation. All the specifications in this item can be test under the 'specific' tab in the input area.

Correctness Checked by entering a variety of valid strings, and making sure that the results match the cycle notation for each component. All tests are **passed**.

Reliability Checked by inputing different types of invalid strings. The error massage and simple examples occurs real time for the illegal string. As the mean time, if click 'create' button when the inputs are illegal, the output will give the corresponding error message rather than continue to generation of the maps. All tests are **passed**.

Performance Checked the corresponding text and images for a parsing in the output area are correct by hand. The test is **passed**.

Transforming cycle notation to one-line notation.All the specifications in this item can be test under the 'specific' tab and 'random' tab in the input area.

Correctness Checked by entering different strings of these two notations. The output should match the format of another notation. The test is **passed**.

Performance TChecked the corresponding text and images for a transformation in the output area are correct by hand. The test is **passed**.

Computation of faces by given the permutations of vertices and edges. This particularly depends on the equation $\phi = \sigma o \alpha$. All the specifications in this item can be test under the 'specific' tab in the input area.

Correctness Checked by entering variety vertices and edges permutations. The faces can be conducted in both notations. The test is **passed**.

Performance Checked the corresponding text and images for a face in the output area are correct by hand. The test is **passed**.

Generating a random combinatorial map by only given the number of darts (and vertices). All the specifications in this item can be test under the 'random' tab in the input area.

Correctness Checked by entering different numbers of darts and generating the graph which is connected successfully. The test is **passed**.

Performance Checked the text and images for each component in a map in the output area are correct by hand. The test is **passed**.

6.2.2 Visualising combinatorial maps

The tool using different methods to visualise the combinatorial maps. All the specifications in this item can be test at the bottom of the output area.

Correctness Checked by judging whether the results of methods followed the constraints of the maps, that is, it must obey the orientation of each part.The whole structure must be connected and filled in the specific area without overflow. For the planar map, it must avoid the crossing as much as possible. All the tests are checked by hand, and the first two tests are **passed**. For the last test, the last method **passed**, while, for the first two, it **failed**.

Clarity Checked by making sure that users can have a clear mind when they look at the images immediately. This test **passed** for the last two methods and **failed** for the first approach.

Performance Checked by making sure that it shows the results in the right area and has an effective introduction. These test **passed**. Checked by creating a map in the limited time. This test **failed** when generating a large map with the huge number of vertices.

Consistency Checking by making sure the result of the certain method for a map must be the same at any time. These test **passed**.

Aesthetics Checked by showing tools to they others, and the receiving the feedback of the users. The most feedback is beautiful, colourful and clearly structural for the whole tool, but the result for the first approach looks a little terrible. Therefore, this test **passed** for the whole structure but **failed** for a specific result.

6.2.3 Knowledge of combinatorial maps

The tool aims to pass more information about combinatorial maps for whether they are beginners or experts in the relative area. A introduction of the maps and tool is necessary.

Correctness Checked by making sure the background of the combinatorial maps is correct and not misleading. The test **passed**.

Performance Checked whether each section of the introduction displays successfully and is there a introduction at the beginning of the visualiser page. The test **passed**.

CHAPTER 7

Further work

7.1 Achievements

After testing the visualisation methods and the system in the section 6,most of the specifications are satisfied. The tool has the functions to pursue the drawing the combinatorial maps and visualising the vital features of the maps.

- The introduction page and the head of the visualiser page help the users learn the combinatorial maps and know the basic structure of the tool.
- The tools support three kinds of input to satisfy different requirement of users. The existed examples which are the typical combinatorial maps help the user experience the whole process of the tool; people can also try and test their own sample using the specific input; if users have no much idea about the combinatorial maps, they can receive a random map from the random section by just enter the number of darts.
- The tool can parsing the users input and convert the input into an accessible format. It can also convert the one-line notation permutation to the cycle-notation, or reversely.
- The tool can calculate the faces with the help of edges permutations and vertices permutations according to the equation $\phi = \sigma o \alpha$. It visualises all the permutations under both notations, as well.
- The visualiser can draw the tree structure using tree drawing algorithm. Under the tree model (spanning tree), the whole structure of combinatorial maps can be drew.
- A improved version of tree based methods also be implemented. This algorithm makes the result more transparent and tidy.
- Another method based on the dual of combinatorial maps is implemented, as well. The results of the method is clearer, but losing the connectivity between each face.
- To improve legibility of the visualisation, some interactive functions are added, for example, highlight the edges and vertices, indicating them for a tips and zooming the each canvas.

7.2 Issues

However, some issues occur in the testing results, which mainly around the visualisation approaches.

- The methods under the spanning tree structure cannot avoid the crossing, especially for the first method which never avoid crossing at all. It even cannot generate a clear structure.

- The faces based method cannot shows the structure directly, that is to say, it needs an extra work for users to glue the faces together.
- When generating a large combinatorial maps with a lot of number of darts like 100 and the number of vertices is more than half of darts, the system might be blocked or delay generating the maps. The reason why this problem appears is the function that generates the permutation of vertices. Since, the probability of generating a VC contains empty members, as it was mentioned before, the empty elements cannot appear in the final result.

7.3 Future improvement

7.3.1 Drawing methods

Tree based method

Though there is an improved version of the first tree based method, but it can be better by itself. After the darts fixed, it can simulate the connecting approach in the second method by discussing different connection situations. Through checking the directions of the two pre-connected darts to determine which side of space the curves might go through.

Faces based method

Gluing the faces is the next work of the method. The core idea is that if two darts of two faces are tied together, rotating one of the faces clockwise so that the darts can fit together. Following the steps until all the faces are gluing together. In order to reestablish the surface which the maps embedded in, the folding method could be used. However, the research of the folding algorithms stays in theoretical stage without the actual algorithmic process[Dem05] so that there is no effective measure for building a 3-dimensional or higher-dimensional surface from the 2-dimensional map.

7.3.2 Interactions

Additional interactions make the tool more entirely and practically.

Save images

The result of drawing is the SVG format. A saving functions to save the image into different format such as PNG, JPG and SVG. There is a method for convert the SVG format into the PNG in the JAVASCRIPT as the *showInCanvas()* in the file *type1.js*. After that, the image can be download as the PNG.

Animated operations

The zooming behaviour has been added into each result. One more operation, dragging is necessary too. Under the behaviour, people can drag and turn the faces in the last method to connect the them manually to make up for the lack of gluing faces. Meanwhile, it also enhances users expression of knowledges of equivalent for two maps.

CHAPTER 8

Project management

8.1 Project structure

This project is a continuation of the previous research. When I first came into contact with the concept of combinatorial maps, it was very vague to me. The only information I know about it is the basic structure of the maps, so that I just implement a method to visualise the map with the fundamental steps. With the successful implementation of the first visualisation method, I learned more about the maps and drawing graph. According to prior knowledges accumulated in the previous research, I was quite clear about the content and purpose of the project. The essential purpose is still using different methods to visualise the structure, but a tool produced to enhance the project integrity and practicality. It also used to validate the result of methods. After discussing the ideas with supervisor and writing the proposal, the structure of project is more clearly. The first step is generating the frame of the tool. Filled the tool with different visualisation methods.

Validating whether each unit of the tool works well and testing if results of the approaches perform well according to the specification writing in the proposal. All tests through the strict constrains. At the beginning, the visualiser page was separated into two pages, one for trying the specific examples of users and a randomly generation as the another page. Some fixed examples were used to test the methods, so that the early work is to creating the format page. For reducing the time to input the same string every time, a typical examples item was supported.Due to the same process for generating maps, the page of producing the random maps was merged into the same page.

Though the project has been finished successfully, there are still some problems can be improved. So many time is using to check the related knowledges that it is no sufficient time to implement every ideas who might be pursued later.

8.2 Project log

The project log was submitted weekly on the canvas, which also can be seem in the Appendix B. The log help me to summarise the knowledges learned and recording the problems appeared in the project. It gives me enough time to consider what should do next week, as well.

8.3 Supervisor meeting

Meeting with the Project supervisor held once a week, to know what I have done in the previous week and discuss the problems I met. A clearer mind and explicit direction have been achieved after the discussion.

8.4 Version control

The GitLa is used to control the version of the project. The log of the Git helps us know what changes in each push operations. It will store the latest version of project after implementing a new function or fixing the problems of the project.

CHAPTER 9

Conclusion

Summarily, the project is to find different methods to visualise the combinatorial maps which is a topological structure. According to the previous research and work, the project set out to explore the layout of the map through searching the properties of combinatorial map. Firstly, it has the spanning tree and the structure can be produced based on the tree model. As the same time, many tree drawing algorithm can be used to build the tree structure. After implementing the first method, we found that the additional information is needed for the tree structure. The improved version of tree based method aims to strengthen the tree structure and makes the whole layout more transparent. Other than the spanning tree, another feature called dual of maps provide a new idea about the visualisation. Drawing faces and then gluing them. Unfortunately, the gluing method did not be implemented. From the whole project, the tool has already achieved the goals, though it still have some issues need to be fixed.

Bibliography

[Bos19] Bostock, Mike et al.: *D3. js-data-driven documents*. Sept. 2019 (cit. on p. 14).

[Buc02] Buchheim, Christoph, Michael Jünger, and Sebastian Leipert: 'Improving Walker's algorithm to run in linear time'. *International Symposium on Graph Drawing*. Springer. 2002: pp. 344–353 (cit. on p. 17).

[Dem05] Demaine, Erik D and Joseph O'Rourke: 'A survey of folding and unfolding in computational geometry'. *Combinatorial and computational geometry* (2005), vol. 52: pp. 167–211 (cit. on p. 38).

[Edm60] Edmonds, J.R.: *A Combinatorial Representation for Oriented Polyhedral Surfaces*. University of Maryland, 1960 (cit. on p. 5).

[Fel19] Feldman, Aidan: 'Table of Contents plugin for Bootstrap'. (Sept. 2019), vol. (cit. on p. 25).

[Fis43] Fisher, Ronald A and Frank Yates: *Statistical tables for biological, agricultural and medical research*. Oliver and Boyd Ltd, London, 1943 (cit. on p. 28).

[Fra13] Fraysseix, Hubert de and P Ossona de Mendez: 'PIGALE-Public Implementation of a Graph Algorithm Library and Editor'. *Handbook of Graph Drawing and Visualization* (2013), vol. (cit. on p. 8).

[Hax05] Haxhimusa, Yll, Adrian Ion, Walter Kropatsch, and Luc Brun: 'Hierarchical image partitioning using combinatorial maps'. *Joint Hungarian-Austrian Conference on Image Processing and Pattern Recognition*. 2005: pp. 179–186 (cit. on p. 8).

[Hua19] Huang, Tingyu: 'Different techniques for visualization of combinatorial map'. (2019), vol. (cit. on pp. d, 10).

[Jac69] Jacques, Alain: 'Constellations et propriétés algébriques des graphes topologiques'. PhD thesis. Faculté des Sciences, Université de Paris, 1969 (cit. on p. 5).

[Kob13] Kobourov, Stephen: 'Force-Directed Drawing Algorithms'. *Handbook of Graph Drawing and Visualization* (2013), vol. (cit. on p. 10).

[MIL] MILL, Bill: *Drawing Presentable Trees* (cit. on p. 17).

[MIT19a] MIT: *Bootstrap*. Sept. 2019 (cit. on p. 25).

[MIT19b] MIT: *Open Iconic*. Sept. 2019 (cit. on p. 25).

[nLa19] nLab authors: *combinatorial map*. Sept. 2019 (cit. on p. 5).

[Rei81] Reingold, Edward M. and John S. Tilford: 'Tidier drawings of trees'. *IEEE Transactions on software Engineering* (1981), vol. (2): pp. 223–228 (cit. on p. 17).

[Ric08] Richeson, David S: *Euler's gem: the polyhedron formula and the birth of topology*. Vol. 6. Springer, 2008 (cit. on p. 8).

[Rin12] Ringel, Gerhard: *Map color theorem*. Vol. 209. Springer Science & Business Media, 2012 (cit. on p. 5).

[Tam13] Tamassia, Roberto: *Handbook of graph drawing and visualization*. Chapman and Hall/CRC, 2013 (cit. on p. 2).

[You18] You, Evan: 'Vue. js'. *Diakses dari https://vuejs. org/, pada tanggal* (2018), vol. 17 (cit. on p. 27).

[Zep12] Zeps, Dainis and Paulis Kikusts: 'How to draw combinatorial map? From graphs and edges to corner rotations and permutations'. *arXiv preprint arXiv:1207.5673* (2012), vol. (cit. on p. 9).

List of Figures

A File Structure

The appendix shows the structure of the project directory. The program can be started at any .html file in the html directory.

A.1 root - **Main files**

- README.md: A readme file

A.2 document **directory** - **Documents**

- 1899855-Tingyu Huang-proposal.pdf: The proposal of the project.
- dissertation draft.pages: The draft of the dissertation.
- dissertation draft_complete.pages: The content of dissertation.
- Drawer.pdf: The prorotype of the tool.
- latexthesistemplate directory: The Latex templete of the dissertaion.
- Tingyu Huang-1899855.pdf: The dissertation of project.

A.3 css **directory** - **Style of pages**

- index.css: The general style of the tool.

A.4 html **directory** - **HTML files**

- index.html: The introduction page.
- type1.html: The visualiser page.

A.5 javascript **directory** - **Source code**

- data.json: The file stores all the typical examples. (If the file cannot loading succesfully, the examples tab in the visualiser cannot be used. Since the limitation of the browser, a local servise is needed to resolve cross-domain issues, otherwise, the files cannot be read.)
- Drawing.js: All the implementation of visualisation.
- index.js: The initialization of introduction page.
- type1.js: The layout and data monitoring on visualiser page.

A.6 image **directory** - **Images**

The images used in the dissertaion and project.

A.7 other **directories** - **Others**

The library introduced in the project.

B Project log

week 1 This week mainly talk about the topics which were related to the combinatorial map. The first one is using different techniques to do the visualization of combinatorial maps and another one is to convert the pixels of images into combinatorial maps. For the first topic, there were two more methods can be used to visualize, using a spinning tree or using the third component faces of combinatorial maps. As for the second topic, it needs more knowledge about it. All of these topics and approaches need me to do further research

The second task of the week was discussing which types of topics could I choose and how to perform the final work for the types. Two types were suitable to me, theory and software engineering, if I want to search the more methods to show the combinatorial map the first type might a good choice, however, if I can produce an interface to help the people know a lot about the combinatorial map, the second type is better. There is no doubt that this task is base on the first one. Hence, more information about the topics is needed to help make decisions and find the most appropriate project for me to do.

week 2 The meeting this week had two parts, solving the problems in the proposal and identifying the details of the project. After reading the proposal template, I found I was confusing about some items, for example, whether the project objective should be more specific or not. Therefore, the supervisor helped me to tackle the barrier. During this period, the approaches for dealing with the problem of the topic were settled, and we discussed if the ideas were feasible. Supervisor gave me more professional theoretical support and clearer research direction after listening to my description of the thoughts.

week 3 Discussing the details of the project through the proposal. First, a clear direction, since the project mainly cares about the approaches of visualization of combinatorial maps, it needs to involve all general combinatorial maps, rather than focus on the quite narrow type of maps like the maps based on the planar maps. As well, certain the primary frame of the system via prototypes.

week 4 This week introduce the library and frame used to implement the designing of the sketch.

week 5 Implement the sketch of the web page, and discuss the content of the report. The library for drawing the maps.

week 6 The validating and parsing of the inputs and the supervise support some other idea for drawing the graph.

week 7 Changing the structure of the pages, computing spanning tree and visualizing the permutation and cycles notations. Changing some terminology on pages.

week 8 Implement drawing tree algorithm and visualizing tree-based combinatorial maps. Discussing faces-based algorithm and the methods to make the result clearer. Adding the useful introduction information to help freshman to understand the structure.

week 9 Another tree-based method to avoid the cross problem. The problems occur in the project.

week 10 Solving the problems occurred in the second tree-based method, and complete the method. Some suggestions about the layout of the page to collect all the type of inputs into one page.

week 11 Finish the whole pages, fix the problem.

week 12 Discussing details about the dissertation.

week 13 Displaying the pages and discussing each section of the dissertation. Finishing the writing work.