Journal of Computer Science and Cybernetics, V.37, N.3 (2021), 185–200
DOI 10.15625/1813-9663/37/3/16253

USING COMBINATORIAL MAPS FOR ALGORITHMS ON GRAPHS

ROBERT CORI

Université de Bordeaux, Bordeaux, France

Abstract. In this paper the representation of the embedding of a graph on a surface by a pair of permutations is considered. This representation is used to implement efficiently some algorithms for graphs. The more important ones are the generation of all the spanning trees and the computation of the Tutte polynomial. A detailed presentation of a recursive formula due to I. Pak for the Tutte polynomial of the complete graphs is also given.

Keywords. Graph algorithms, Topological embeddings, spanning trees Tutte polynomials.

1. INTRODUCTION

The aim of this paper is to come back to a data structure representation of graph by permutations. This originated in the years 1960-1970 by contributions due to J. Edmonds [1], A. Jacques [2], W. Tutte [3] in order to consider the embedding of a graph in a surface as a combinatorial object. Some algebraic developments where suggested in [4] and [5]. It was also used for implementation in different situation, like planarity testing by H. de Fraysseix and P. Rosenstiehl [6], computer vision by G. Damiand and A. Dupas [7] or formal proofs by G. Gonthier [8].

There are two main reasons to come back to this old combinatorial data structure. The first one is the publication of the Volume 4 (Fasc. 4) of the book of D. Knuth "The Art of Computer Programming". This booklet contains an algorithm for the generation of all spanning trees of a graph. It is interesting to compare this algorithm using the *dancing links* data structure introduced by this author in [9] to the one using combinatorial maps for deletion and contraction of an edge we present here.

The second reason is the renewal of interest to Tutte Polynomials in many recent papers like those O. Bernardi in [10] and with T. Kalman and A. Postnikov in [11]. The use of combinatorial maps to compute the Tutte polynomials we present here may be compared to other recent proposals for computing these polynomials (see [12, 13, 14]).

This paper contains many algorithms written like computer programs but also some combinatorial results, it is organized as follows. In the next section we introduce combinatorial

Dedicated to Professor Phan Dinh Dieu on the occasion of his 85th birth anniversary.

Corresponding author.
E-mail addresses: robert.cori@labri.fr.

maps as a mathematical object and an algorithmic data structure to represent graphs. In Section 3, we recall some facts on the representation of topological graphs by this mathematical object, the readers interested by this aspect should consult the books: [15] and [16]. In Section 4 we show how the combinatorial maps can be used as a data structure for classical algorithms like Depth First Search and the deletion and contraction of an edge which are central in graph theory. Section 5 is devoted to an algorithm for generating all spanning trees of a graph which is inspired by that given in [17]. The last two sections are concerned with Tutte polynomials associated to a graph. In Section 6 the definition of these polynomials are recalled and a way to compute them is proposed. In Section 7 we come back to a result given by I. Pak on the computation of the Tutte polynomials of the complete graph which contains a nice bijection.

Notice that when speaking of a graph in this paper we mean an undirected graph $G = (V, E)$ where V is the set of vertices and E a set of edges, each edge e having two end vertices as end points v_i, v_j which may have loops (that means that v_i may be equal to v_j) and multiple edges (that allows two edges to have the same endpoints).

2. THE TWO PERMUTATIONS DEFINING A COMBINATORIAL MAP

Permutations

We will use permutations as a tool for the representation of a graph. A permutation α is usually defined as a one to one function of the set $\{1, 2, \ldots, n\}$ on itself. It is represented by the sequence of the n integers $\alpha(1), \alpha(2), \ldots, \alpha(n)$, $\alpha(i)$ satisfies $1 \leq \alpha(i) \leq n$ and all the $\alpha(i)$ are all different since α is a one to one function. We will write algorithms for permutations which will be implemented by programs where an array will contain this sequence, hence we define a permutation of size n as an array of length n containing all the integers $\{0, 1, \ldots, n-1\}$. This will allow a quick transformation of our algorithms into programs in languages like C, Java or Python, for which $\alpha[0]$ is the first element of the array α.

The composition of permutations α, β will be used denoting $\alpha\beta(i) = \alpha(\beta(i))$.

Example 1. For instance considering the two permutations $\alpha = 4, 0, 1, 7, 2, 8, 6, 5, 3, 9$ and $\beta = 1, 0, 3, 5, 9, 2, 7, 6, 4, 8$ we have $\alpha\beta = 0, 4, 7, 8, 9, 1, 5, 6, 2, 3$ while $\beta\alpha = 9, 1, 0, 6, 3, 4, 7, 5, 8$.

A *cycle* of a permutation α is a sequence $(i_1, i_2, \ldots, i_p)$ such that $i_{j+1} = \alpha(i_j)$ for $j < p$ and $\alpha(i_p) = i_1$. The permutation α in the example above has four cycles

$$C_1 = (0, 4, 2, 1), \quad C_2 = (3, 7, 5, 8), \quad C_3 = (6), \quad C_4 = (9).$$

One can write the representation of α by its cycles like

$$\alpha = (0, 4, 2, 1)(3, 7, 5, 8)(6)(9).$$

As a training in order to be familiar with our description of algorithms we give the simple algorithm finding the cycles of a permutation and computing their number.

Transpositions

A transposition τ on $\{0, 1, 2, \ldots, n-1\}$ is a permutation with one cycle of length 2 and $n-2$ cycles of length 1. Such a permutation can be written as a sequence, as follows

$$1, 2, \ldots i-1, j, i+1 \ldots j-1, i, j+1, \ldots n,$$

```
1  Function nbCycles(α): /* Count the number of cycles of a permutation α     */
   Data: The permutation α on n elements and a boolean array visited initialized to false.
   Result: The number of cycles of α
2  nbC ⟵ 0;
3  for j ← 0 to n − 1:
4  |  if (not visited[j])then
5  |  |  visited[j] ⟵ true;
6  |  |  nbC ⟵ nbC + 1;
7  |  |  k ⟵ j;
8  |  |  while σ[k] ≠ j do
9  |  |  |  visited[σ[k]] ⟵ true;
10 |  |  |  k ⟵ σ[k] ;
11 |  |  end
12 |  end
13 return nbC;
```

where $i < j$ are the elements in the cycle of length 2.

Or as the decomposition into cycles

$$(0)(1)\cdots(i,j)\cdots(n-1).$$

We will prefer here the notation $\tau = (i, j)$, hence ignoring the fixed points in the decomposition of τ into cycles, which is often the case while writing permutations.

Lemma 1. *Let α be a permutation and $\tau = (i, j)$ a transposition. Then for the permutation $\alpha\tau$ obtained by the composition of α and τ, satisfies one of the two following conditions:*

- *If i and j are in different cycles of α then these two cycles are merged into one cycle of $\alpha\tau$.*
- *If i and j are in the same cycle of α then this cycle splits into two cycles of $\alpha\tau$, one containing i and the other containing j.*

Proof. In the two cases we have

$$\alpha\tau(k) = \alpha(k) \text{ if } k \notin \{i, j\}, \text{ and } \alpha\tau(i) = \alpha(j),\ \alpha\tau(j) = \alpha(i).$$

In the first case the two cycles $(i, a_1, \ldots, a_p)$ and $(j, b_1, \ldots, b_q)$ of α, give the cycle

$$(i, b_1, \ldots, b_q, j, a_1, \ldots, a_p) \text{ in } \alpha\tau.$$

In the second case the cycle $(i, a_1, \ldots, a_p, j, a_{p+1}, \ldots, a_q)$ of α gives the two following cycles in $\alpha\tau$: $(i, a_{p+1}, \ldots, a_q)$ and $(j, a_1, \ldots, a_p)$.

Remark 1. The operation of transforming the permutation α into $\alpha\tau$ where $\tau = (i, j)$ may be done by exchanging in the array representing it the values of $\alpha[i]$ and $\alpha[j]$. This is done in three elementary operations

$$temp \longleftarrow \alpha[i];\ \ \alpha[i] \longleftarrow \alpha[j];\ \ \alpha[j] \longleftarrow temp.$$

Combinatorial maps

Definition 1.

A combinatorial map consists of two permutations σ and α on the set $\{0, 1, \ldots, 2m-1\}$ such that all the cycles of α have length 2.

Example 2. Consider the combinatorial map consisting of the two following permutations defined by their cycles

$$\sigma = (0,2,4)(1,6,8,)(3,9,10)(5,11,7), \quad \alpha = (0,1)(2,3)(4,5)(6,7)(8,9)(10,11).$$

To a combinatorial map one can associate a graph in the following way. The vertices of this graph are associated to the cycles of σ and the edges are associated to the cycles of α. More precisely if σ has k cycles denoted $C_1, C_2, \ldots, C_k$ the graph $G(\sigma, \alpha)$ has k vertices $v_1, v_2, \ldots, v_k$, to each cycle (i,j) of α is associated an edge with end points v_p, v_q, such that C_p is the cycle of σ containing i and C_q is the cycle of σ containing j. Notice that this graph may have multiple edges if the cycles (i,j) and (i',j') of α are such that i, i' are in the same cycle of σ and so as j, j'. It may have loops if a cycle (i,j) of α is such that i, j are in the same cycle of σ. The graph associated to the combinatorial map of Example 2 has 4 vertices and 6 edges, one for each pair of vertices, it is represented in the figure below.

The elements $1, 2, \ldots, 2m$ defining the permutations σ and α are called here *half-edges*, they are also called points or arcs or corners in other papers.

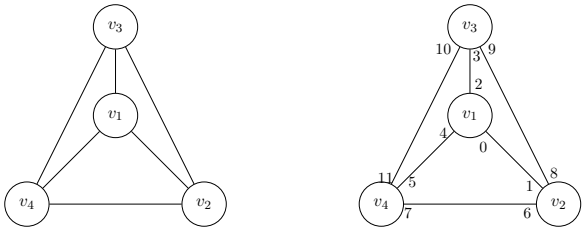

Figure 1: The graph associated to the combinatorial map of Example 2

Remark 2. For some algorithms modifying a graph it will be useful to allow α to have cycles of length 1. These cycles should be ignored in the graph represented by such a combinatorial map.

Proposition 1. *The graph associated to a combinatorial map (σ, α) is connected if and only if the group generated by the permutations σ and α acts transitively on the set of half-edges. This means that any half-edge i may be reached from the half-edge 0 by a combination of actions of σ and α. When it is not connected the orbits of the action of the group correspond to connected components in the graph.*

In many algorithms of this paper it will be assumed that the permutation α is such that $\alpha(i) = i + 1$ if i is even and $\alpha(i) = i - 1$ if i is odd, this α will be called *canonical*.

3. TOPOLOGICAL EMBEDDINGS OF GRAPHS

The embedding of a graph on an orientable surface may be represented by a combinatorial map. In this embedding the order in which the edges are met around a vertex is important. This order is taken into account in the representation by asking that the half edges around a vertex should be met in a trigonometrical positive (or counterclockwise) rotation as they appear in each cycle of σ. Taking as example the graph of Figure 1, when representing the embedding the cycles of σ are

$$\sigma = (0,2,4)(1,6,8)(3,9,10)(5,11,7).$$

Notice that graph above may be embedded in the torus in two different ways, these are described in Figure 2, where the torus is represented as a square where the top and bottom boarders are identified as the left and right ones. For these representations the permutations are

$$\sigma_1 = (0,2,4)(1,8,6)(3,10,9)(5,7,11) \quad \text{and} \quad \sigma_2 = (0,4,2)(1,6,8)(3,9,10)(5,7,11).$$

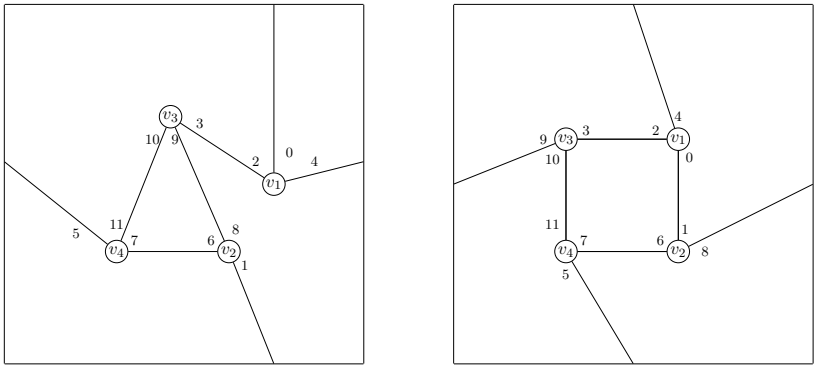

Figure 2: Two embeddings of the complete graph K_4 in the torus

The main interest of this representation is given by the determination of the faces of the embedding using the permutations.

Proposition 2. *In the representation of the embedding of a graph by a combinatorial map* (σ, α) *the faces correspond to the cycles of the permutation obtained by the composition of* α *and* σ*.*

Proof. When going around one face of the embedding beginning by the half-edge i one goes around the vertex of this half-edge to the half-edge $\sigma(i)$ then along the edges of it to the half-edge $\alpha(\sigma(i))$ and so on until one reaches i again. This consists of a visit of all the elements of the cycle of $\alpha\sigma$ containing i. ∎

The examples of these cycles for the different representations of K_4 give 4 cycles for the first embedding

$$\alpha\sigma = (0,3,8)(1,7,4)(2,5,10)(6,9,11).$$

For the others there are 2 cycles

$$\alpha\sigma_1 = (0,3,11,4,1,9,2,5,6)(10,8,7),$$

$$\alpha\sigma_2 = (0,5,6,9,11,4,3,8)(1,7,10,2).$$

As a consequence the genus g of the surface of the embedding satisfies equation (1) below, where m is the number of cycles of α, n the number of those of σ and f the number of those of $\alpha\sigma$.

$$m + 2 - 2g = n + f. \tag{1}$$

4. CLASSICAL GRAPH ALGORITHMS

Many algorithms on graphs use a *searching* procedure, that means to follow the edges in order to visit all their vertices. One of the most popular searching is *Depth First Search* that uses adjacency lists for visiting all the neighbors of a vertex, we can write this algorithm

using combinatorial maps. We illustrate this method with an algorithm giving the number of vertices that can be reached by a path starting in a vertex v_i. This algorithm may be used to test the connectivity of a graph, since the graph is connected if and only if the number of vertices reached by a path from v_i is equal to the total number of vertices. Depth first search was also used to test planarity and building spanning trees with a special property see: [6, 18, 19].

The algorithm $nbCountDfs(i)$ counts the number of vertices in the connected component of the vertex incident to the half edge i. First this vertex is visited by turning around the vertex using the cycle of σ such that all the half-edges incident to that vertex are marked. Then for all these half-edges j, the half edge $k = \alpha(j)$ is tested, if it was not already visited, a recursive call to the algorithm is done by $nbCountDfs(k)$.

```
1  Function nbCountDfs(i, visited): /*                                  */
   Data: i is a half edge of the start vertex, visited is a boolean array initialized to false. The
         combinatorial map is given by the two permutations σ, α represented as arrays of
         integers.
   Result: The number of vertices in the component of the start vertex
2  nbF ⟵ 0;
3  visited[i] ⟵ true;
4  k ⟵ i;
5  while σ[k] ≠ i do
6  |  visited[σ[k]] ⟵ true;
7  |  k ⟵ σ[k] ;
8  end
9  k ⟵ i;
10 repeat
11 |  j ⟵ α[k];
12 |  if (not visited[j])then
13 |  |  nbF ⟵ nbF + 1 + nbCountDfs(j, visited);
14 |  end
15 |  k ⟵ σ[k];
16 until (k = i);
17 return nbF;
```

We now consider two fundamental operations on a graph which are often used in graph theory, these are the deletion and the contraction of an edge. With these two operations one builds the *minors* of a graph.

The deletion operation is very simple it has effect to obtain a graph with one edge less than itself. When programming this operation one can keep the same half-edges and consider that a half-edge i such that $\alpha[i] = i$ does not belong to an edge of the graph. Hence the deletion in the combinatorial map σ, α of the edge composed by the two half-edges i and j consists in the operation of replacing α by $\alpha\tau$, τ being the transposition (i, j). Doing this transformation costs 3 elementary operations and after that one can do a reinsertion of it by the same elementary operations. Hence the function $un - delete(i, j)$ will be the same but applied to a combinatorial map such that $\alpha(i) = i$ and $\alpha(j) = j$.

1 **Function** `delete(`i, j`): /* Delete the edge consisting of the two half edges` i, j `*/`

Data: i, j are half edges such that $\alpha[i] = j$ and $\alpha[j] = i$
Result: The edge is deleted, then i and j are fixed points of α
2 $\alpha \longleftarrow \alpha\tau$;

The operation of contraction of two vertices v, v' connected by an edge e with the two half-edges i in v and j in v' consists in replacing these two vertices by one vertex which will be incident to all the edges incident with v or v' except e itself. This operation merges the two cycles of σ corresponding to the vertices v and v' and deletes e. Lemma 1 expresses that this merging can be done by replacing σ by $\sigma\tau$ where $\tau = (i, j)$. Notice that if one wants to perform the inverse operation, returning to the initial map, the multiplication by τ can be also used since τ^2 is equal to the identity. This fact will be useful in the algorithms considered in the next sections.

1 **Function** `contract(`i, j`): /* Contract the edge consisting of the two half edges` i, j `*/`

Data: i, j are half edges such that $\alpha[i] = j$, $\alpha[j] = i$ and i and j are in different cycles of σ
Result: The edge is contracted, then i and j are in the same cycle of σ
2 $\alpha \longleftarrow \alpha\tau$;
3 $\sigma \longleftarrow \sigma\tau$

A bridge in a connected graph is an edge which deletion disconnects the graph. Using the algorithms already considered we can determine if an edge e is a bridge by deleting this edge then count the number of vertices which can be reached by a path starting from a vertex incident to e. The edge e is a bridge if and only if this number is less than the total number of vertices (which is equal to the number of cycles of σ).

1 **Function** `isBridge(`i, j`): /* Determine if the edge consisting of the two half edges` i, j `is a bridge */`

Data: i, j are half edges such that $\alpha[i] = j$ and $\alpha[j] = i$
Result: The result is *true* if after the deletion of this edges the graph is not connected, and $false$ otherwise
2 $n \longleftarrow nbCycles(\sigma)$;
3 $delete(i, j)$;
4 $k \longleftarrow nbCountDfs(i)$;
5 **return** $(k < n)$;

5. ALL SPANNING TREES

Recall that a spanning tree of a connected graph $G = (V, E)$ with n vertices is a subset T of E containing $n - 1$ edges that does not contain a cycle. Hence for any pair of vertices v, v' of G there exists a unique path using edges in T. In a relatively recently written chapter of The Art of Computer Programming [17] D. E. Knuth proposed an algorithm to generate all the spanning trees of a graph he uses the dancing links with doubly linked lists. We introduce here an algorithm using the data structure of combinatorial maps.

In this Section and the nexts we will consider that G with n vertices and m edges is represented by a combinatorial map (σ, α) such that

$$\alpha = (0,1)(2,3)\cdots(2m-2,2m-1).$$

We will represent the spanning tree by an array containing $n-1$ even integers $i_1, i_2, \dots i_{n-1}$ such that the tree T consists of the edges

$$\{i_1, i_1+1\}, \{i_2, i_2+1\}, \cdots, \{i_{n-1}, i_{n-1}+1\}.$$

For the graph represented in Figure 1 the arrays $[0,2,4]$ and $[4,6,8]$ represent spanning trees.

In order to build all spanning trees we will use un array T' of even numbers $i_1, i_2, \dots, i_k$ satsifying $i_1 < i_2, \dots, < i_k$ such that the set of edges $\{i_1, i_1+1\}, \{i_2, i_2+1\}, \cdots, \{i_k, i_k+1\}$ contains no cycle and we will try to add an edges $e = \{2j, 2j+1\}$ satisfying $2j > i_k$ and such that e does not add a cycle to T'. Repeating this until $k = n-1$ will give a spanning tree. A recursive procedure repeating this construction will provide our full algorithm.

The key point for the efficiency of the algorithm is the following.

Lemma 2. *In a connected graph $G = (V, E)$ let T' be a subset of edges that do not contain a cycle, and let e be an edge not in T'. Then $T' \cup \{e\}$ contains a cycle if and only if e is a loop in the graph obtained from G by contracting all the edges in T'.*

Proof. In a cycle $e_1, e_2, \dots e_k, e$ of a graph G, the contraction of the edges $e_1, e_2, \dots e_k$ will give a unique vertex obtained by merging the half edges adjacent to both endpoints of all the e_i. In this graph the edge e has both endpoints adjacent to this vertex, hence it is a loop. ∎

We also have.

Proposition 3. *In a connected graph $G = (V, E)$ let E' be a subset of E that does not contain a cycle. Then all the spanning trees of G containing E' are obtained by adding to E' each of the spanning trees of the graph $G_{E'}$ obtained from G by contracting all the edges in E'.*

Proof. We proceed by induction on the number k of edges in E'.

If $k = 1$ then $E' = \{e\}$ let v_i and v_j be such that $e = \{v_i, v_j\}$, let T' be a spanning tree of the graph G_e obtained by contracting e in G. Then each edge in T' has at most one among v_i, v_j as endpoint, adding the edge e to T' and returning will consist in separating the vertex obtained by the contraction of e into two vertices v_i and v_j and adding an edge between them. This transforms the spanning tree T' of G_e into a spanning tree of G. Going from E' having k elements to E' with $k+1$ elements may be done by similar arguments. ∎

The whole algorithm can now be written. To check that $k, k+1$ is a loop in the graph obtained by the contraction of the edges in E' one checks if k and $k+1$ are not in the same cycle of the permutation σ obtained after performing the contractions. The recursive procedure needs to un-contract after contracting but we observe that the three basic operations for contraction performed twice rebuild the initial combinatorial map representing the graph.

6. TUTTE POLYNOMIALS

In two seminal papers ([20, 21]) W. T. Tutte introduced a polynomial associated to a graph which contains many informations on it. The definition of this polynomial uses the

```
1  Function allTrees(tempTree,i): /* Display all the spanning trees of the map
     (σ, α), where α is canonical.                                          */
   Data: i is an integer and tempTree is an array of integers containing the label of the edges
         determining a partial tree of i − 1 edges.
   Result: Display all the spanning trees which contain the edges in tempTree and edges
           with labels greater than those.
2  n is the number of cycles of σ;
3  m is the number of cycles of α;
4  if i= n-1 then
5  |  display(tempTree) ;
6  else
7  |  if i = 0 then
8  |  |  k ⟵ 0;
9  |  else
10 |  |  k ⟵ tempTree[i − 1] + 2;
11 |  end
12 |  while k < 2 ∗ m do
13 |  |  if (not inSameCycle(σ, k, k + 1)) then
14 |  |  |  tempTree[i] ⟵ k;
15 |  |  |  contract(k,k+1);
16 |  |  |  allTrees(i+1,tempTree);
17 |  |  |  (un)Contract(k,k+1);
18 |  |  end
19 |  |  k ⟵ k + 2;
20 |  end
21 end
```

association of a monomial in two variables to any spanning tree of this graph, the Tutte polynomial is the the sum of all these monomials.

We consider a connected graph (G, E) where the edges are labelled by non negative integers such that no two different edges have the same label, this gives a total order on its edges.

Definition 2. Let T be spanning tree of G and e an edge of T, the deletion of the edge e in T divides this tree into two subtrees T' and T''. Denote $Cut(T, e)$ the subset of E consisting of all the edges $e' = \{v_i, v_j\}$ such that v_i is in T' and v_j is in T'', clearly $e \in Cut(T, e)$. The edge e is internally active with respect to T if it has the minimal label among all the elements of $Cut(T, e)$.

Definition 3. Let T be spanning tree of G and $e = \{v_i, v_j\}$ an edge not in T. Denote $Cyc(T, e)$ the subset of E consisting the edge and all the edges in the path in T with end points v_i and v_j, $Cyc(T, e)$ is then an elementary cycle of G. The edge e is externally active with respect to T if it has the minimal label among all the elements of $Cyc(T, e)$.

The monomial $mon(T)$ associated to the spanning tree T is $x^a y^b$ where a is the number of edges internally active with respect to T and b the number of edges externally active with respect to T. The Tutte polynomial $T_G(x, y)$ of the graph G is the sum of monomial $Mon(T)$ for all the spanning trees T of G.

Example 3. Consider the cycle graph C_n having n vertices $v_1, v_2, \cdots, v_n$ and n edges labelled

$e_1 = \{v_1, v_2\}$, $e_i = \{v_i, v_{i+1}\}$ for $i < n$ and $e_n = \{v_1, v_n\}$. Let T_i be the spanning tree of G obtained by deleting e_i in E. Then we have that T_1 has one externally active edge (namely e_1) and no internally active edge), T_2 has one internally active edge (namely e_1) and no externally active edge, and for $i > 1$ T_i has $i-1$ internally active edge (namely $e_1, e_2, \ldots, e_{i-1}$) and no externally active edge. Hence the Tutte polynomial of C_n is $y + x + x^2 + \cdots + x^{n-1}$.

The main results concerning these polynomials are summarized in the Theorem below.

Theorem 1. *The Tutte polynomial of the graph $G = (V, E)$ does not depend of the order of the labels of the edges in E. Moreover for any edge e the Tutte polynomial T_G of G may be determined using the Tutte polynomial $T_{G'}$ of the graph G' obtained from G by deleting e and the polynomial $T_{G''}$ of the graph G'' obtained by contracting e as follows:*

1. *If the graph has only one edge e then $T_G = y$ if e is a loop and n $T_G = x$ if e is a bridge.*
2. *If the edge e is a loop then $T_G = yT_{G'}$.*
3. *If the edge e is a bridge then $T_G = xT_{G''}$.*
4. *If e is neither a loop nor a bridge then $T_G = T_{G'} + T_{G''}$.*

The following Theorem allows to compute the monomial associated to a spanning tree of a connected graph.

Theorem 2. *Let $G = (V, E)$ be a connected graph where the edges are labelled $e_1, e_2, \ldots, e_m$ and let T be a spanning tree of G. For $1 \leq i \leq m$ let G_k be the obtained from G by contracting all the edges e_i such that $i > k$ and $e_i \in T$ and deleting all the edges such that $i > k$ and $e_i \notin T$ then we have for $1 \leq k \leq m$*

- *e_k is internally active with respect to T if and only if $e_k \in T$ and e_k is a bridge of G_k.*
- *e_k is externally active with respect to T if and only if $e_k \notin T$ and e_k is a loop of G_k.*

Proof.

If $e_k \in T$ the e_k is internally active if $Cut(T, e_k)$ does not contain an edge e_j with $j < k$, we notice that this is equivalent to the fact that e_k is a bridge of G_k.

If $e_k \notin T$ the e_k is externally active the cycle $Cyc(T, e_k)$ contains only edges e_j such that $j > e_k$ this is equivalent to the fact that e_k is a loop of G_k. ∎

Using this proposition we obtain the following algorithm determining the monomial $Mon(T)$. This algorithm examines all the edges of the graph in reverse order of their label.

- If the edge $\{j, j+1\}$ is an element of the tree T then it is candidate to be an internally active edge, this happens if it is a bridge in the graph obtained by contracting the edges in T with label greater than j. To check this fact we use the procedure described at the end of Section 4, since the current number of vertices of the graph is given by the variable $nbVertices$ this procedure does not need to determine it. Then this edge is contracted and the number of vertices decreased by 1.

- If the edge $\{j, j+1\}$ is not an element of the tree T then it is candidate to be an externally active edge, this happens if it is a loop in the graph obtained by contracting the edges in T with label greater than j. To check this fact we determine if j and $j+1$ are in the same cycle of the permutation σ of the current graph obtained by performing the successive contractions.

The full Tutte polynomial is obtained by adding all the monomials obtained by using a new procedure which is obtained from the procedure *allTrees* in replacing *display*(*tempTree*) by add *Monomial*(*tempTree*) to the current polynomial.

```
1  Function monomialOf(thisTree): /* Find the monomial x^a y^b associated to the
     spanning tree thisTree for the Tutte polynomial                          */
2  n ⟵ numberOfCycles(σ);
3  m ⟵ numberOfCycles(α);
4  j ⟵ 2m − 2;
5  nbVertices ⟵ n;
6  a ⟵ 0;
7  b ⟵ 0;
8  while j ≥ 0 do
9      if (nbVertices ≥ 2) and (j = thisTree[nbVertices − 2])then
10         if isBridge(j,j+1)then
11             a ⟵ a + 1 ;
12         end
13         contract(j, j + 1);
14         nbVertices ⟵ nbVertices − 1 ;
15     else
16         if inSameCycle(σ, j, j + 1)then
17             b ⟵ b + 1 ;
18             delete(j, j + 1);
19         end
20     end
21     j ⟵ j − 2 ;
22 end
23 return (x^a y^b);
```

7. THE TUTTE POLYNOMIAL OF COMPLETE GRAPHS

The complete graph with n vertices denoted K_n has n vertices $v_1, v_2, \ldots, v_n$ and all pairs of vertices v_i, v_j determine an edge. The number of its edges is then the binomial coefficient $\binom{n}{2}$.

The Tutte polynomial of this graph interested many researchers and a formula for the exponential generating function of these polynomials is given in [18]. Less known is a simple recursive formula proposed by I. Pak in an unpublished manuscript available in a web page (see [22]) where the proof is outlined. it seems interesting to give here this proof in more details since it relies on a nice bijection.

This allows to compute this polynomial thanks to this recursive formula which helps to obtain it. Let T_{K_n} denote the Tutte polynomial of K_n, by convention we denote $T_{K_1} = 1$, we have $T_{K_2} = x$. The Tutte Polynomial of K_3 which is also the cycle C_3 is obtained in Example 3 and is equal to $y + x + x^2$, $T_{K_3} = x^2 + x + y$.

Theorem 3. *For $n > 2$ the Tutte polynomial T_{K_n} may be computed from the polynomials T_{K_i} for $1 \leq i < n$ using the equation*

$$T_{K_n}(x, y) = \sum_{p=1}^{n-1} \binom{n-2}{p-1} (x + y + y^2 + \cdots y^{p-1}) T_{K_p}(1, y) T_{K_{n-p}}(x, y), \tag{2}$$

with the convention that $\binom{n}{0} = 1$.

Let us first illustrate this theorem by showing how to compute T_{K_4}. Applying equation (2) with $n = 4$ we have

$$T_{K_4} = \binom{2}{0} x T_{K_3} + \binom{2}{1}(x+y) T_{K_2}(1,y) T_{K_2} + \binom{2}{2}(x+y+y^2) T_{K_3}(1,y).$$

Which gives

$$T_{K_4} = x(y+x+x^2) + 2(x+y)(1)(x) + (x+y+y^2)(2+y),$$

and

$$T_{K_4} = x^3 + y^3 + 3x^2 + 3y^2 + 4xy + 2x + 2y.$$

Proof of Theorem 3.

We now give a proof of this Theorem by using a bijection. This proof was given by I. Pak [22] in a preprint, we propose to give it here since it sheds light on labelled trees and the determination of the number of active internal and external edges.

This proof proceeds in three steps, in the first one we give a canonical labelling of the edges of the complete graph and a canonical combinatorial map representing it, in the second one we give a bijection between the spanning trees T of K_n and quadruples composed of two spanning trees T' and T'' one of K_p (for some p such that $1 \leq p < n$) the other of K_{n-p} , an integer i such that $1 \leq i \leq p$ and a subset of $\{3, \cdots n\}$ containing $p-1$ elements. The third step consists in computing the Tutte polynomial of K_n using that of K_p and K_{n-p}. ∎

Canonical labelling for the edges of K_n.

The vertices of the complete graph K_n will be denoted here by $v_1, v_2, \cdots, v_n$ it has $m = \frac{n(n-1)}{2}$ edges hence $n(n-1)$ half edges. We define a canonical combinatorial map representing it, it has as half edges the integers $0, 1, \cdots, 2m-2, 2m-1$, the edges are represented by the permutation α with cycles $(0,1)\cdots(2i, 2i+1)\cdots(2m-2, 2m-1)$. The half edges are labelled such that the edges are ordered lexicographically

$$\{v_1, v_2\}, \{v_1, v_3\}, \cdots \{v_1, v_n\}, \cdots \{v_2, v_3\}, \cdots \{v_2, v_n\}, \cdots \{v_{n-1}, v_n\},$$

an edge $\{v_i, v_j\}$ represented by $(2a, 2a+1)$ and an edge $\{v_k, v_l\}$ represented by $(2b, 2b+1)$ one has $a < b$ if and only if $i < k$ or $i = k$ and $j < l$.

Taking as example the complete graph K_4 the edges are such that

$$\alpha = (0,1)(2,3)(4,5)(6,7)(8,9)(10,11),$$

corresponding in that order with

$$\{v_1, v_2\}, \{v_1, v_3\}, \{v_1, v_4\}, \{v_2, v_3\}, \{v_2, v_4\}, \{v_3, v_4\}.$$

Hence giving the permutation which cycles corresponding to the vertices v_1, v_2, v_3, v_4

$$\sigma = (0,2,4)(1,6,8)(3,7,10)(5,9,11).$$

This labelling of the edges defining the permutation α determines an order on them using the usual order on integers.

Lemma 3. *Let T be a spanning tree of K_n where the edges are labelled as above. Then $e \in T$ is an active internal edge if and only if $e = (v_1, v_i)$ and i is such that for any v_j which is in the subtree of T with root v_i one has $j > i$.*

Proof. Cutting $e = \{v_i, v_j\}$ in the tree T decomposes T into two subtrees T' and T'', suppose that $i, j \neq 1$ let T' be the subtree containing v_1 then one of the external edge $\{v_1, v_i\}$ or $\{v_1, v_j\}$ connects the two subtrees and has a label smaller than that of $\{v_i, v_j\}$, hence $\{v_i, v_j\}$ is not active. Considering an edge $\{v_1, v_i\}$ of T , this edge is internally active if and only for any the vertex v_j in the subtree with root v_i one has $j > i$, since $\{v_1, v_j\}$ is an edge connecting the two subtrees obtained by deleting $\{v_1, v_i\}$ in T. ∎

Lemma 4. *Let T be a spanning tree of K_n where the edges are numbered as above. Then $e \notin T$ is an active external edge if and only if one of the two following conditions holds*

1. *The edge e connects two vertices belonging to a subtree T' with root a neighbor v_k of v_1 and is externally active in T'.*
2. *$e = \{v_1, v_i\}$ and i is such that in the path going from v_1 to v_i in T the first step $\{v_1, v_k\}$ satisfies $k > i$.*

Proof. If $e = \{v_i, v_j\}$, where v_i and v_j are in two different subtrees which roots are neighbors of v_1, then the cycle of $T \cup \{e\}$ contains an edge $e = \{v_1, v_k\}$ which has a label less that that of $e = \{v_i, v_j\}$. If they are in the same subtree then e is clearly active externally active in that subtree. If $e = \{v_1, v_i\}$ then the cycle of $T \cup \{e\}$ contains an edge $e' = \{v_1, v_k\}$ and e is externally active if and only if $k > i$. ∎

Example 4. For the spanning tree of K_{10} drawn in Figure 3 containing the edges

$$\{v_1, v_3\}, \{v_1, v_6\}, \{v_1, v_7\}, \{v_2, v_5\}, \{v_2, v_7\}, \{v_3, v_8\}, \{v_4, v_6\}, \{v_7, v_{10}\}, \{v_9, v_{10}\}.$$

Notice that their labels determine the cycles of α given by

$$(2, 3), (8, 9), (10, 11), (22, 23), (26, 27), (42, 43), (50, 51), (82, 83), (88, 89).$$

This tree has one internally active edges namely $\{v_1, v_3\}$ and 4 externally active edges

$$\{v_1, v_2\}, \{v_1, v_4\}, \{v_1, v_5\}, \{v_7, v_9\}.$$

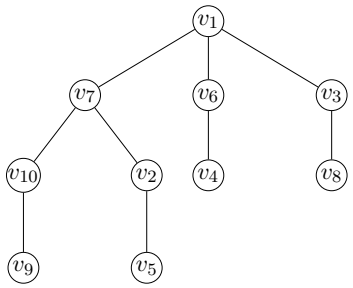

Figure 3: A spanning tree T of K_{10}

Decomposing the spanning trees of K_n

A spanning tree of K_n is represented by the label of $n-1$ edges. We now build a decomposition of the spanning tree T of K_n into two trees. The vertices of K_n are $v_1, v_2, \ldots, v_n$.

Consider the path in the tree T going from v_1 to v_2 and cut the first edge $\{v_1, v_k\}$ of this path, this gives two trees T' and T''. The tree T' contains the vertex v_2 and T'' the vertex v_1. Denote p the number of vertices of T'. Let X be the subset of the indices of the vertices in T' except v_2 and i be the number of vertices in T' which index is less or equal to k. Let T_1 be the tree obtained from T' by renumbering the vertices from v_1 to v_p keeping the order of these indexes. Let T_2 be the result of the same operation on T''. Then T_1 is a spanning tree of K_p while T_2 is a spanning tree of K_{n-p}.

Definition 4. Let $\phi(T)$ be the quadruple (T_1, T_2, X, i).

Clearly one can build T knowing $\phi(T)$. First label the vertices of T_1 in such a way that v_1 becomes v_2 and the other vertices take as indexes the values in X according to the order they have in T_1. Then the vertices in T_2 different from v_1 have to be receive new indexes taken in the complement of X in $\{3, 4, \ldots, n\}$ as this was done for T_1. The last step consists in adding an edge from v_1 in T_2 to the i-th vertex in T_1 using the usual order on integers.

The construction is illustrated below by the spanning tree of K_{10} in Example 4.

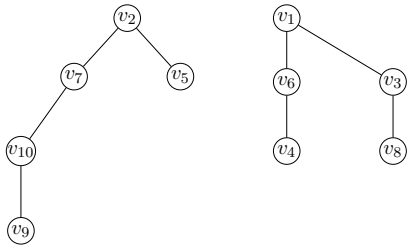

Figure 4: The decomposition of the spanning tree T into T' and T'', the value of i is 3

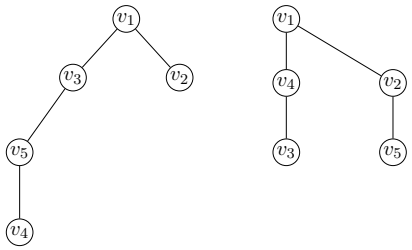

Figure 5: The spanning tree T decomposed into T_1 and T_2

The Tutte polynomial of K_n from those of K_p and K_{n-p}

We consider now how to compute the monomial $mon(T)$ associated to the tree T, knowing $\phi(T) = (T_1, T_2, X, i)$ and the monomials $mon(T_1) = x^a y^b$, $mon(T_2) = x^c y^d$. This monomial does not depend of X since for two subsets X and X' with $p-1$ elements order of the labels of the edges are the same. The vertices of T are denoted $v_1, v_2, \ldots v_n$ let us denote $u_1, u_2, \ldots, u_p$ those of T_1 and $w_1, w_2, \ldots, w_{n-p}$ those of T_2.

By Lemma 3 we have that the internal active edges of T are those coming from T_2 and $\{v_1, v_2\}$ if $i = 1$, hence there are $c+1$ internally active edges in T if $i = 1$ and c ones if $i \neq 1$.

Concerning the externally active edges of T we use Lemma 4. These are of two types some corresponding to those of T_1 and of T_2 and those added by the reconstruction of T.

These of the second type are the edges $\{v_1, v_j\}$ of K_n where the v_j's correspond to the vertices u_k of T_1 such that $k < i$.

Hence if $Mon(T_1) = x^a y^b$ and $Mon(T_2) = x^c y^d$ then $Mon(T) = x^{c+1} y^{b+d}$ if $i = 1$ and $Mon(T) = x^c y^{b+d+i-1}$ if $i > 1$. Given T_1, T_2, X where T_1 is spanning tree of K_p and T_2 is spanning tree of K_{n-p} there are p different spanning trees of T that can be built, one for each value of i, hence the number of externally active edges of T is $b + d + i - 1$.

The sum of the $Mon(T)$ for these trees is given by

$$x^{c+1} y^{b+d} + x^c y^{b+d+1} + \cdots x^c y^{b+d+p-1}. \tag{3}$$

Returning to Example 4 and the trees T_1, T_2 in Figure 5, we notice that T_1 has two internally active edges $\{v_1, v_2\}$ and $\{v_1, v_3\}$, the corresponding edges in T namely $\{v_2, v_5\}$ and $\{v_2, v_7\}$ are not internally active in T. There is one edge of K_p which is externally active for T_1 namely $\{v_3, v_4\}$, this corresponds to the edge $\{v_7, v_9\}$ externally active for T. The tree T_2 has one internally active edge $\{v_1, v_2\}$ which gives the internally active edge $\{v_1, v_3\}$ in T. For this tree T_2 there is an externally active edge $\{v_1, v_3\}$ this corresponds to the edge $\{v_1, v_4\}$ externally active for T. Since $i = 3$ there are two more externally active edges for T namely $\{v_1, v_2\}$ and $\{v_1, v_5\}$.

To end the proof of Theorem 3 we notice that there are $\binom{n-2}{p-1}$ possibilities for the values of X giving the same polynomial for $i = 1, \ldots, p$. The sum of these polynomials give Equation (2). The polynomial $T_{K_p}(1, y)$ translates the fact that the internally active edges of T_1 are not internally active in T while the externally actives are. The polynomial $T_{K_{n-p}}(x, y)$ translates the fact that both internally and externally active edges of T_2 correspond to active edges in T.

8. CONCLUSION

The description given here of the algorithms using combinatorial maps is hoped to be used for Master's courses in Discrete Mathematics and Data Structures. The use of this data structure for implementation was already very frequent. It sheds a new light on the beautiful theory of Tutte polynomials and the different results presented are also hoped to be used to enrich this domain.

REFERENCES

[1] J. R. Edmonds, "A combinatorial representation for polyhedral surfaces," *Notices Amer. Math. Soc*, vol. 7, 646, 1960.

[2] A. Jacques, "Sur le genre d'une paire de susbstitutions," *C. R. Acad. Sci. Paris*, vol. 267, pp. 625–627, 1968.

[3] W. T. Tutte, "What is a map?" in *New Directions in The Theory of Graphs.* Academic Press, New York, 1973, pp. 309–325.

[4] R. Cori and A. Machí, "Maps, hypermaps and their automorphisms, a survey," *Expo. Math.*, vol. 10, pp. 403–467, 1992.

[5] G. Jones and D. Singerman, "Theory of maps on orientable surfaces," *Proc. London Math. Soc.*, vol. 31, pp. 211–256, 1978.

[6] H. de Fraysseix and P. Rosenstiehl, "A depth-first-search characterization of planarity," in *Graph Theory (Cambridge 1981)*, A. North Holland, Ed., 1981, pp. 75–80.

[7] G. Damiand and A. Dupas, "Combinatorial maps for 2d and 3d image segmentation," in *Digital Geometry Algorithms*, Springer, Ed., 2012, pp. 359–393.

[8] G. Gonthier, "A computer checked proof of the four colour theorem," *Notices of the Amer. Math. Soc.*, vol. 55, no. 11, pp. 1382–1393, 2008.

[9] D. E. Knuth, "Dancing links," *arXiv:cs/0011047v1*, 2020.

[10] O. Bernardi, "A characterization of the Tutte polynomial via combinatorial embeddings," *Annals of Combinatorics*, vol. 10, pp. 139–153, 2008.

[11] O. Bernardi, T. Kalman, and A. Postnikov, "Universal Tutte polynomial," *arXiv:math/2004.00683*, 2020.

[12] A. Björklund, T. Husfeldt, P. Kaski, and M. Koivisto, "Computing the Tutte polynomial in vertex-exponential time," in *2008 49th Annual IEEE Symposium on Foundations of Computer Science*, 2008, pp. 677–686.

[13] M. Monagan, "A new edge selection heuristic for computing the tutte polynomial of an undirected graph," *DMTCS*, vol. Proc AR, pp. 839–850, 2012.

[14] K. Sekine, H. Imai, and S. Tani, "Computing the Tutte polynomial of a graph of moderate size," in *Algorithms and Computation, 6th International Symposium.* Lecture notes in Computer Science Springer, 1995, pp. 224–233.

[15] S. K. Lando and A. K. Zvonkin, *Graphs on Surfaces and Their Applications*, S. Verlag, Ed., 2004.

[16] B. Mohar and C. Thomassen, *Graphs on Surfaces.* John Hopkins University Press, 2001.

[17] D. E. Knuth, *The Art of Computer Programming, Vol 4, Fasc.4 : Generating all Trees*, Addison-Wesley, Ed., 2006.

[18] I. Gessel and B. Sagan, "The Tutte polynomial of a graph, depth-first search, and simplicial complex partitions," *Electronic Journal of Combinatorics*, vol. 7, no. 2, 1996.

[19] J. Hopcroft and R. E. Tarjan, "Efficient planarity testing," *Journal of the ACM*, vol. 21, pp. 549–568, 1974.

[20] W. T. Tutte, "A contribution to the theory of chromatic polynomials," *Canad. J. Math.*, vol. 6, pp. 80–91, 1954.

[21] ——, "On dichromatic polynomials," *J. of Comb. Theory*, vol. 2, pp. 301–320, 1967.

[22] I. Pak, "Computation of the Tutte polynomials of complete graphs," *https://www.math.ucla.edu/ pak/.*

Received on July 05, 2021
Accepted on August 27, 2021