Theoretical Computer Science 411 (2010) 2945–2967

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

An optimal algorithm to generate rooted trivalent diagrams and rooted triangular maps

Samuel Alexandre Vidal *

Lille 1 University of Sciences and Technology, Paul Painlevé Laboratory of Mathematics, 59 655 Villeneuve d'Ascq Cédex, France

ARTICLE INFO

Article history:
Received 10 July 2007
Received in revised form 30 March 2010
Accepted 12 April 2010
Communicated by E. Pergola

Keywords:
Triangular maps
Trivalent diagrams
Exhaustive genretator
CAT generator

ABSTRACT

A *trivalent diagram* is a connected, two-colored bipartite graph (parallel edges allowed but not loops) such that every black vertex is of degree 1 or 3 and every white vertex is of degree 1 or 2, with a cyclic order imposed on every set of edges incident to the same vertex. A *rooted* trivalent diagram is a trivalent diagram with a distinguished edge, its *root*. We shall describe and analyze an algorithm giving an exhaustive list of rooted trivalent diagrams of a given size (number of edges), the list being non-redundant in that no two diagrams of the list are isomorphic. The algorithm will be shown to have optimal performance in that the time necessary to generate a diagram will be seen to be bounded in the amortized sense, the bound being independent of the size of the diagrams. We call this the CAT property. One objective of the paper is to provide a reusable theoretical framework for algorithms generating exhaustive lists of complex combinatorial structures with attention paid to the case of unlabeled structures and to those generators having the CAT property.

0. Introduction

Roughly speaking, a trivalent diagram is a connected graph with degree conditions imposed on its vertices and a cyclic orientation imposed on the edges adjacent to each vertex. It is the combinatorial description of an unembedded trivalent ribbon graph [23,14] (cf. Definitions 1.1 and 1.2 for a precise definition). We shall see (cf. Theorem 1.1) that it can be described by a pair of permutations $(\sigma_\bullet, \sigma_\circ)$ satisfying the conditions of *involutivity* $\sigma_\circ^2 = \mathrm{id}$ and *triangularity* $\sigma_\bullet^3 = \mathrm{id}$. The notion of rooted trivalent diagrams is also very useful, both to our study and to the target applications; so we take a special care to study them in detail.

0.1. Problem statement

We shall describe and analyze two algorithms, the first giving an exhaustive list of rooted trivalent diagrams of a given size (cf. Definitions 1.3 and 1.4 below) and the second giving an exhaustive list of unrooted trivalent diagrams (Definitions 1.1 and 1.2), those lists being non-redundant in that no two diagrams in the same list are isomorphic. The algorithm for rooted diagrams will be shown to have optimal performance meaning that the time necessary to generate a diagram is bounded in the amortized sense. What is striking is that the bound is independent of the size of the diagrams being generated. One objective of the paper is to provide a reusable theoretical framework for algorithms generating exhaustive lists of complex combinatorial structures with attention paid to the case of unlabeled structures and to those generators having the CAT property.

* Tel.: +33 06 85 16 83 55.
E-mail address: samuel.vidal@yahoo.fr.

0304-3975/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2010.04.026

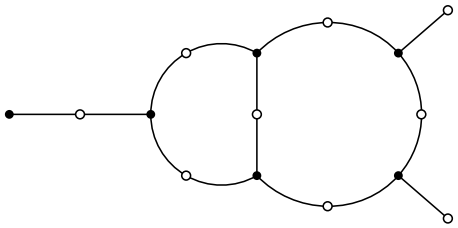


Fig. 1. A trivalent diagram is conveniently described by a diagram like the one above, hence the name. The actual cyclic orientation of the vertices are conveniently rendered implicit by adopting the counter-clockwise orientation of the figure.

0.2. Motivation

In a recent paper [26] we gave a complete classification of the subgroups of the modular group $\mathrm{PSL}_2(\mathbb{Z})$ and their conjugacy classes by rooted trivalent diagrams and trivalent diagrams. A question one may ask is how to generate a complete list of such trivalent diagrams. Such a question is unavoidable: for a classification to be fully satisfactory there should be a systematic way to enumerate all the particular instances of the objects being classified. Moreover, it was soon realized that there is a connection with combinatorial maps. In this paper we clarify that point and give as an application a way to generate exhaustive lists of triangular combinatorial maps.

The other sources of motivations to generate trivalent diagrams come mainly from mathematical physics in connection with two-dimensionnal quantum gravity and the Witten–Kontsevich model [14]. Algebraic topology is also a source of motivation through triangular subdivisions of surfaces, knots, braids, links and tangles theory [23,2]. It is also connected to the deformation theory of quantized Hopf algebras [6,7]. The problem we solve is also relevant to the study of combinatorial maps as explained in Section 5 and to the vast Galoisian program of Grothendieck [9] as explained in hundreds of papers and books such as [21,26,16,5]. As an application, we give in Section 4 a way to generate a complete list describing all the subgroups of a given finite index in the modular group $\mathrm{PSL}_2(\mathbb{Z})$ and a way to decide conjugacy relations among those subgroups. We show also in Section 5, as a second application, how to generate an exhaustive list of triangular maps satisfying various criteria.

0.3. Definition of a CAT generator

The expression "CAT generator" is an acronym for *constant amortized time generator* meaning a generator of combinatorial structures that on the average spend only a constant time generating each of the structures. The usual idea in such a generator is that passing from one structure to the next requires only a few modifications to be made. Sometimes, though, it could take more modifications than usual and we usually have no upper bound on the number of actual modifications that could be needed to pass from a structure to the next. While the need for a large number of modifications tends to be significantly rare in comparison to a small number, we can sometimes prove that an amortization effect is going on. Technically, one can summarize that amortization effect by saying that the total amount of time needed to generate n distinct structures is asymptotically bounded by a constant multiple of the number n of structures being generated, the word *constant* meaning that the bound is independent of the size of the structures being generated.

0.4. Comment on terminology

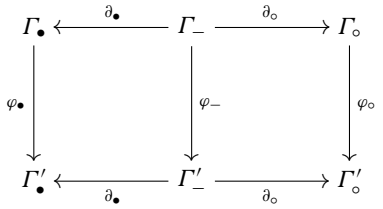
We chose to borrow some notation and terminology from category theory in this exposition. The related concepts: categories, functors, natural transformations, equivalence of categories, are fully covered in the first four sections of the first chapter of the textbook [17, p. 7–18] by Mac Lane.

1. Trivalent diagrams

Definition 1.1. A *trivalent diagram* is a connected, two-colored bipartite graph (parallel edges allowed but not loops) such that every black vertex is of degree 1 or 3 and every white vertex is of degree 1 or 2, with a cyclic order imposed on the edges incident to each vertex. The *size* of a trivalent diagram is the number of its edges.

Given a trivalent diagram Γ, we denote by Γ_-, $\Gamma_\bullet$ and $\Gamma_\circ$ the sets of its edges, black vertices and white vertices, respectively. Given an edge $a \in \Gamma_-$, we denote by $\partial_\bullet(a) \in \Gamma_\bullet$ and $\partial_\circ(a) \in \Gamma_\circ$ the black vertex and the white vertex to which it is incident. Given an edge $a \in \Gamma_-$, we denote by $\sigma_\bullet(a)$ and $\sigma_\circ(a)$ the next edge incident to $\partial_\bullet(a)$ and $\partial_\circ(a)$, respectively, in the cyclic order. According to the degree conditions of the definition we have $\sigma_\bullet^3 = \sigma_\circ^2 = \mathrm{id}$ which implies that both $\sigma_\bullet$ and $\sigma_\circ$ are bijections; so they are permutations on the set of edges Γ_- of the diagram. The connectivity condition of the definition is equivalent to the transitivity of the permutation group generated by $\sigma_\bullet$ and $\sigma_\circ$ (Fig. 1).

Definition 1.2. A *morphism* φ between two trivalent diagrams Γ and Γ' is a triple of mappings $\varphi_\bullet$, $\varphi_\circ$ and φ_- from the three sets $\Gamma_\bullet$, $\Gamma_\circ$ and Γ_- to the three sets $\Gamma'_\bullet$, $\Gamma'_\circ$ and Γ'_-, respectively, compatible with the three structure mappings and the two permutations in that φ_- is equivariant to each of the permutations $\sigma_\bullet$ and $\sigma_\circ$ and the following diagram is commutative.



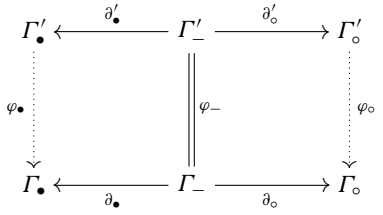
When those three mappings are bijections the morphism is an *isomorphism*.

An important fact, well known to the experts, is recalled in the following theorem. It is used throughout the article to formulate the algorithms.

Theorem 1.1. *The set Γ_- and the two permutions $\sigma_\bullet$ and $\sigma_\circ$ entirely suffice to describe the isomorphism class of the digram Γ. Moreover, the cycles of the permutations $\sigma_\bullet$ and $\sigma_\circ$ are in natural bijection with the black and white vertices of Γ, respectively.*

Proof. Given a trivalent diagram Γ, an isomorphic trivalent diagram Γ' can be reconstructed from the set Γ_- and the two permutations $\sigma_\bullet$ and $\sigma_\circ$ of Γ_-. Let $\Gamma_-/\sigma_\bullet$, the cyles of the permutation $\sigma_\bullet$, be its set of black vertices and let $\Gamma_-/\sigma_\circ$, the cycles of the permutation $\sigma_\circ$, be its set of white vertices, and define its boundary mappings $\partial'_\bullet$ and $\partial'_\circ$ to be the natural projection of the quotients.

We now construct the isomorphism the following way. Since $\partial_\bullet$ and $\partial_\circ$ are equivariant to the permutations $\sigma_\bullet$ and $\sigma_\circ$, respectively, they induce natural mappings $\varphi_\bullet$ and $\varphi_\circ$ completing the following commutative diagram.



Taking φ_- to be the identity mapping, one has a morphism from the diagram Γ' to the diagram Γ. To show it is an isomorphism, one has to show the bijectivity of the three mappings $\varphi_\bullet$, $\varphi_\circ$ and φ_-. The mapping φ_-, being the identity, is necessarily bijective. The bijectivity of the mappings $\varphi_\bullet$ and $\varphi_\circ$ means that two edges are in the same cycles of the respective permutations $\sigma_\bullet$ and $\sigma_\circ$ if and only if they are incident to the same black and white vertex, respectively, which is guaranteed by the definition. □

1.1. Rooted trivalent diagrams

The following concept plays an important rôle in this article and in the applications.

Definition 1.3. A trivalent diagram is said to be *rooted* if one of its edges is distinguished from the others as its *root*.

A convenient way to describe the rooting of a diagram is to draw a cross on its distinguished edge.

Definition 1.4. A *morphism* φ of rooted trivalent diagrams (Γ, a) and (Γ', a') is a morphism of the underlying diagrams (ignoring the roots) whose φ_- component is further assumed to send root to root.

1.2. Labeled versus unlabeled diagrams

Historically, the dichotomy between *labeled* and *unlabeled* structures had been greatly clarified and properly emphasized by the introduction by Joyal of *combinatorial species* [11]. The subject was, and still is, a very prolific source of discovery from the Quebec school of combinatorics and from a growing community of researchers around the world. One must cite the book [3,4] by Bergeron, Labelle and Leroux, which gives an exposition of the whole subject.

On a given set of vertices X one can build different trivalent diagrams and rooted trivalent diagrams. We denote by $D_3(X)$ and $D_3^\bullet(X)$ the corresponding sets of structures, we call X the *labeling alphabet* and we refer to diagrams one can build on that set as diagrams *labeled* by X. Any bijection ϱ between two finite sets X and Y induces a bijection ϱ_* between the sets $D_3(X)$ and $D_3(Y)$ of trivalent diagrams labeled by X and Y, respectively. This induced bijection is the *relabeling operation* from $D_3(X)$ to $D_3(Y)$. It is also referred as a *transport of structure* along the relabeling bijection ϱ. The same considerations also apply to rooted trivalent diagrams and in fact to any labeled combinatorial structures.

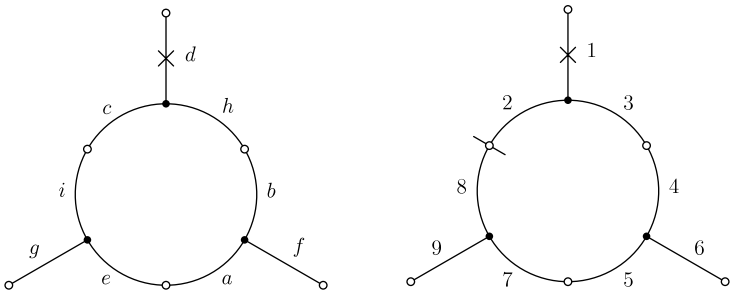


Fig. 2. If one gives as input to the relabeling algorithm (Algorithm 2.2) the rooted diagram shown on the left with an arbitrary initial labeling on the arbitrary alphabet $X = \{a, b, c, d, e, f, g, h, i\}$, it produces the characterstic relabeling shown on the right with numbers from 1 to 9 according to the depth-first traversal order of Algorithm 2.1. Note the *natural cutting* between the edges labeled by 2 and 8 that arises from the depth-first traversal.

The above discussion leads to the consideration of the *Joyal Functors* D_3 and $D_3^\bullet$ of the two combinatorial species of trivalent diagrams and rooted trivalent diagrams, respectively. In the formalism of Joyal, two labeled structures are said to be *conjugate* or *isomorphic* if they coincide modulo the relabeling operation. An unlabeled structure is then just a conjugacy class of labeled structures. We denote by $\tilde{D}_3(n)$ and $\tilde{D}_3^\bullet(n)$ the sets of unlabeled trivalent diagrams, unrooted and rooted, respectively, and by $D_3(n)$ and $D_3^\bullet(n)$ the sets of trivalent diagrams, unrooted and rooted, respectively, and labeled by the set $\{1, \ldots, n\}$. The symmetric group $\mathfrak{S}_n$ acts via relabeling on the set of structures labeled by $\{1, \ldots, n\}$ and the sets $\tilde{D}_3(n)$ and $\tilde{D}_3^\bullet(n)$ can be seen as the quotient sets of those group actions.

$$\tilde{D}_3(n) \stackrel{\text{def.}}{=} D_3(n)/\mathfrak{S}_n \quad \text{and} \quad \tilde{D}_3^\bullet(n) \stackrel{\text{def.}}{=} D_3^\bullet(n)/\mathfrak{S}_n.$$

We call the corresponding natural projections

$$\pi_n : D_3(n) \to \tilde{D}_3(n)$$
$$\pi_n : D_3^\bullet(n) \to \tilde{D}_3^\bullet(n)$$

the *condensation mappings* of the combinatorial species D_3 and $D_3^\bullet$.

2. Characteristic labeling

A *characteristic labeling* is the choice of a unique representative in every conjugacy class of structures. In other terms, a characteristic labeling can be seen as a natural section to the condensation mapping π *i.e.* a natural mapping s such that $\pi s = \mathrm{id}$. Good characteristic labelings are those which are computable. They are even better if there is an efficient way to compute them.

Rooted trivalent diagrams have the enjoyable property of possessing many characteristic labelings that are computable by means of efficient algorithms. This situation is to be contrasted with that of general graphs. No algorithm is known to decide in polynomial time whether two given graphs are isomorphic, and having an efficient algorithm computing a characteristic labeling of general graphs would render that particular problem trivial. What makes trivalent diagrams particular in that respect is not so much that they are trivalent but more in that their edges are cyclically oriented at their vertices. Indeed, general graphs with only trivalent vertices still suffer from the above problem.

What we give now is a succinct description of an algorithm producing a characteristic labeling of rooted trivalent diagrams Γ and having linear time-complexity in the number of edges of Γ. The idea is the following: build a rooted plane binary tree T by depth-first traversal (in prefix order) of the edges of the diagram (not the vertices, we insist on the edges). Given a particular edge a of Γ, the two directions that are explored from it are given by the two operations $\sigma_\bullet$ and $\sigma_\circ$ on the set of edges. We take care never to revisit a previously visited edge and we label the edges of Γ by numbers from 1 to n according to the order of their appearance in the depth-first traversal (Fig. 2).

2.1. Implementation

We need as global data an integer c and the following seven arrays:

$$\begin{aligned} \mathit{visited} &: X \to \mathrm{Bool} \\ \ell_0 &: X \to \{1, \ldots, n\} \\ \ell_1 &: \{1, \ldots, n\} \to X \\ s_0, s_1 &: X \to X \\ t_0, t_1 &: \{1, \ldots, n\} \to \{1, \ldots, n\}. \end{aligned}$$

Algorithm 2.1, which is an auxiliary recursive program, computes the transport bijections ℓ_0 and ℓ_1. Algorithm 2.2 is the main entry point of the relabeling process. It does the initialization job (line 2–4) and the actual relabeling of the input diagram (line 6–8). It takes as input a trivalent diagram labeled with the elements of the set X and rooted by the element x of X. The arrays s_0 and s_1 and the element $x \in X$ are descriptions of the input diagram via its associated two permutations $\sigma_\bullet$ and $\sigma_\circ$ (cf. Theorem 1.1). The output diagram is encoded by the two arrays t_0 and t_1 in the very same fashion. The array *visited* is used to remember the positions already visited by the relabeling process. The integer c serves as a counter to label the vertices in the order they are encountered, while ℓ_0 and ℓ_1 are internal arrays describing the mutual inverse transport bijections between the input diagram and the output diagram.

Algorithm 2.1: VISIT ($x : X$)

```
1 begin
2   if visited[x] then return
3   visited[x] ← true
4   ℓ0[x] ← c
5   ℓ1[c] ← x
6   c ← c + 1
7   VISIT (s0[x])
8   VISIT (s1[x])
9 end
```

Algorithm 2.2: RELABEL ($x : X$)

```
1 begin
2   c ← 1
3   for i ∈ X do
4     visited[i] ← false
5   VISIT (x)
6   for k ∈ {1, ..., n} do
7     t0[k] ← ℓ0[s0[ℓ1[k]]]
8     t1[k] ← ℓ0[s1[ℓ1[k]]]
9 end
```

2.2. Correctness

The idea behind that algorithm is quite simple and presents no difficulty except the actual proof of the relabeling being characteristic. There are two ways to do the proof; one is conceptual by nature and the other is more technical. The particular description of the algorithm is itself part of that former argument. We shall give both arguments because preferring one or the other is simply a mater of taste. We give the conceptual argument first.

One could have taken the input diagram to be labeled by the set $\{1, \ldots, n\}$ and then shown that the output labeled diagram remains unchanged if one conjugates the input labeled diagram according to any permutation of the labeling set. Such a proof would typically look rather technical if not difficult. Instead, one can rather *abstract* the labeling alphabet of the input diagram to be an arbitrary n-element set X, this requirement being the only assumption made on X. In particular, we make absolutely no assumption on its elements or on any structure that it may carry.

A moment's thought may convince the reader that abstracting the input label set to X and making no assumption whatsoever on its elements indeed guarantees the required invariance. As this argument is a bit subtle and may seem a hand-waving argument to most people, we now give another proof avoiding such considerations.

Theorem 2.1. *Algorithm* 2.2 *produces a characteristic relabeling of the connected rooted trivalent diagrams of size n — that is, t_0 and t_1 are invariant under any bijection from X onto another set X'.*

Proof. Any bijection ϱ between two sets of input labels X and X' induces a conjugacy of the two input permutations s_0 and s_1 of X yielding two permutations $s'_0 = \varrho \cdot s_0 \cdot \varrho^{-1}$ and $s'_1 = \varrho \cdot s_1 \cdot \varrho^{-1}$ of X'. Now, putting $\ell_0(x) = c$ and $\ell'_0(x') = c$ according to line 4 of Algorithm 2.1 with $x' = \varrho(x)$ and varying x yields $\ell'_0 = \ell_0 \cdot \varrho^{-1}$. Similarly, considering line 5 of the same algorithm, we get $\ell'_1 = \varrho \cdot \ell_1$. The permutations t_0, t'_0, t_1 and t'_1 verify the following identities (by line 6–8 of Algorithm 2.2):

$$t_0 = \ell_0 \cdot s_0 \cdot \ell_1, \qquad t'_0 = \ell'_0 \cdot s'_0 \cdot \ell'_1,$$
$$t_1 = \ell_0 \cdot s_1 \cdot \ell_1, \qquad t'_1 = \ell'_0 \cdot s'_1 \cdot \ell'_1,$$

and substituting ℓ'_0, ℓ'_1, s'_0, and s'_1 for their above values yields a cancellation of the ϱ's,

$$t'_0 = (\ell_0 \cdot \varrho^{-1}) \cdot (\varrho \cdot s_0 \cdot \varrho^{-1}) \cdot (\varrho \cdot \ell_1) = t_0,$$
$$t'_1 = (\ell_0 \cdot \varrho^{-1}) \cdot (\varrho \cdot s_1 \cdot \varrho^{-1}) \cdot (\varrho \cdot \ell_1) = t_1,$$

thus proving the required invariance of the output. □

2.3. Applications

Beside its simplicity, this algorithm has several important consequences and applications,

(1) Since, according to Theorem 2.1, the relabeling is characteristic to the isomorphism classes, one can test the isomorphism of two rooted diagrams by relabeling them using Algorithm 2.2 and simply compare the two results for equality.
(2) There is a linear order on the isomorphism classes of rooted diagrams induced by the lexicographic order on the permutations of their characteristic labeling.
(3) A notion of *characteristic rooting* for diagrams emerges from that linear order by choosing as a representative of each conjugacy class its minimal element.

That last remark can be used to implement an efficient filter procedure rejecting any rooted trivalent diagram that is not minimal in its conjugacy class. It is useful to get a generator of unrooted diagrams from a rooted diagram generator.

3. Generating algorithm

One can imagine that while exploring a particular rooted trivalent diagram using Algorithm 2.2 of Section 2, we output a sequence of events describing the particular cycles of the permutations t_0 and t_1 we encounter at each stage of the traversal. Those events could typically say for example: there we reach a new unforeseen black vertex (forward connection) and we label its adjacent edges c, $c+1$, $c+2$, or there we reach a previously visited white vertex (backward connection), or there we reach an unforeseen white vertex, etc...

One can easily convince oneself that such a sequence of events, relying only on the execution of the algorithm and not on the particular labeling of the input diagram, is in fact characteristic to the diagram. If sufficiently detailed, that sequence of events can be used to unambiguously characterize rooted trivalent diagrams. The idea now would be to consider a rooted plane tree with leaves labeled by rooted trivalent diagrams and with edges labeled by events in such a way that the sequence of events one gets along any branch from the root to a leaf is the very sequence of events that unambiguously characterizes the corresponding rooted trivalent diagram.

We now obtain a usable principle of generation if we require two further properties: exhaustivity, meaning that every isomorphism class of rooted trivalent diagram gets represented on a particular leaf of the tree and non-redundancy, meaning that every such isomorphism class gets represented just once. Assuming that we spend only a constant time on each node of that tree and that the number of those nodes is linearly bounded by the number of its leaves, this would provide a constant amortized time algorithm to generate rooted trivalent diagrams.

To ease the memory requirements of the generator, we won't actually build the generation tree in memory. It will instead be realized in the calling pattern between the procedures of the generating program. Also, the program would be more useful if it generates the diagrams in permutational form instead of a sequence of events describing it. This means that we have to carry around a partial diagram that gets built while exploring the generation tree, each generating event completing that description and each backtrack reversing the particular changes we have made.

3.1. Implementation

The generating algorithm uses as global data two integers c and n, a stack of integers and two integer arrays

$$s_0, s_1 : \{1, \ldots, n\} \to \{1, \ldots, n\}$$

representing the rooted trivalent diagram being constructed by its black and white permutations $\sigma_\bullet$ and $\sigma_\circ$, respectively. The integer n represents the maximum size of the diagram being generated while the integer c is the labeling counter used to attribute integer labels to the edges of rooted trivalent diagrams being generated. The manipulation of the stack is done through the following five primitives.

$$\begin{aligned} \textsc{Push} &: \text{Integer} \times \text{Stack} \to \text{Stack} \\ \textsc{Pop} &: \text{Stack} \to \text{Integer} \times \text{Stack} \\ \textsc{StackIsEmpty} &: \text{Stack} \to \text{Bool} \\ \textsc{Mask}, \textsc{Reveal} &: \text{Integer} \times \text{Stack} \to \text{Stack}. \end{aligned}$$

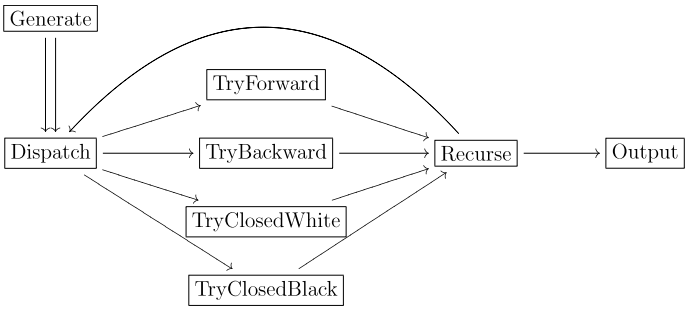


Fig. 3. Overall structure of the generating algorithm.

The stack can be implemented using a doubly linked circular list represented by two zero-based arrays of integers.

$$N, P : \{0, \ldots, n\} \to \{0, \ldots, n\}.$$

The item of index zero is just a sentinel and the stack is considered empty if the following relation holds,

$$N[0] = P[0] = 0.$$

The Mask and Reveal procedures implement removal and insertion primitives using a trick popularized by Knuth [13] under the name of "dancing link". Namely, a call to the Mask procedure with parameter s removes the item s of the stack using the following two instructions,

$$N[P[s]] \leftarrow N[s],$$
$$P[N[s]] \leftarrow P[s],$$

while a subsequent call to the Reveal procedure with parameter s restores the previous state of the stack, before the call to the Mask procedure, using the following two instructions,

$$N[P[s]] \leftarrow s,$$
$$P[N[s]] \leftarrow s.$$

The generating algorithm is composed of seven procedures (Algorithms 3.1–3.7) and a user-defined procedure called Output that serves as an outlet to the algorithm and that can be used, for instance, to do printing jobs or to collect some statistics on rooted trivalent diagrams. The overall structure of the calling pattern between those procedure is shown in Fig. 3. The algorithm works by a recursive exploration of the structure being constructed in a way that mimics the depth-first traversal of the labeling algorithm of Section 2.

The recursion has two base cases that are produced by the Generate procedure (Algorithm 3.1) which is the main entry point of the algorithm. The inductive step of the recursion corresponds to a call to the Dispatch procedure (Algorithm 3.2), whose purpose is to successively handle each of the various cases one can encounter at each stage of the construction/exploration of the diagrams. This is the branching part of the generating algorithm in the sense that it is there that the generation tree forks into subtrees eventually leading to the leaves where the produced structures reside. A call to the Dispatch procedure results in a call to the Recurse procedure (Algorithm 3.7) through each of the four Try procedure (Algorithms 3.4 and 3.5). The purpose of the Recurse procedure is to call the Output procedure if the stack is empty, meaning that the exploration/construction is finished and that we can thus output a finished structure, or to pop an edge and call the dispatch procedure if the stack is not empty, meaning that the structure being explored/constructed is not yet finished.

3.1.1. The Generate procedure

The Generate procedure (Algorithm 3.1) is the main entry point of the program. It is responsible for the two base cases of the induction, namely whether the produced structure has univalent black vertices adjacent to its root edge (handled in lines 3–5 of the procedure) or a trivalent one (handled in lines 7–13).

In the case where the vertex is univalent, the corresponding fixed point is built (line 4 of the algorithm), the labeling counter is set to 2 (the label of the next encountered edge), and then the exploration/construction continues in the direction of the white vertex by calling the Dispatch procedure with parameter 1 (line 5 of the algorithm).

In the case where the vertex is trivalent, the three edges around it are labeled 1, 2 and 3 in counterclockwise direction, the corresponding cycle in the black permutation is built by the three instruction lines 8–10 of the algorithm and the two edges 1 and 2 are pushed onto the stack for further exploration (lines 11 and 12) while the edge 3 is explored in the direction of its white vertex by calling the Dispatch procedure with parameter 3 (line 13 of the algorithm).

Algorithm 3.1: GENERATE ( )

```
1  begin
2      if n ≥ 1 then
3          c ← 2
4          s0[1] ← 1
5          DISPATCH (1)
6      if n ≥ 3 then
7          c ← 4
8          s0[1] ← 2
9          s0[2] ← 3
10         s0[3] ← 1
11         PUSH (1)
12         PUSH (2)
13         DISPATCH (3)
14 end
```

3.1.2. The Dispatch procedure

This is the start of the induction step of the generation algorithm. The hypothesis at its start is that the two arrays s_0 and s_1 reflect the structure of a partial trivalent diagram being explored according to the depth-first traversal of the labeling algorithm of Section 2. The current edge s and the labeling counter c reflect the stage of the exploration. The exploration/construction is supposed to continue from the current edge s in the direction of its white vertex and c is the label attributed to the next unlabeled edge we encounter. At the beginning of the procedure, we do not know whether that white vertex is univalent or bivalent, and if it is bivalent, we do not know which is the other edge incident to it. There are four possible cases:

Case 1. The white vertex incident to the current edge s is univalent. That case is handled by a call to the TryClosedWhite procedure in line 3 of the Dispatch procedure (Algorithm 3.2). We can then assume for the three other cases that this vertex is bivalent.

Case 2. The edge adjacent to the current edge by its bivalent white vertex has not yet been visited and the next black vertex is trivalent. This case is handled in line 4 by a call to the TryForward procedure.

Case 3. As in the previous case, the adjacent edge has not yet been visited but here the next black vertex is univalent. This case is handled in line 5 by a call to the TryClosedBlack procedure.

Case 4. The edge adjacent to the current edge by its bivalent white vertex has already been visited and thus already has a label, which we call t. The point is that those edges, already visited but still awaiting further exploration on their white side, are precisely those that are stored in the stack. Each edge stored in the stack corresponds to an admissible possibility for the white neighbour of the current edge. The exploration of those possibilities is done by the "for" statement in lines 6–9 of the procedure. The edge of the stack t matched with the current edge s is temporarily removed from the stack using the "dancing link" trick implemented by the Mask and Reveal primitives called in lines 7 and 9 of the procedure.

The production of new edges has to be compatible with the maximum allowed size n of the diagrams. That condition is checked by the two "if" statements in lines 4 and 5 of the procedure.

Important. *We claim that those four cases cover all the possibilities and that they are mutually exclusive.*

Remark. The order in which the cases are handled by the Dispatch procedure only affects the order in which the structures are produced but not the way they are labeled nor does it change the set of structures that is produced.

Algorithm 3.2: DISPATCH (s : *integer*)

```
1  local t : integer
2  begin
3      TRYCLOSEDWHITE (s)
4      if c + 3 ≤ n + 1 then TRYFORWARD (s)
5      if c + 1 ≤ n + 1 then TRYCLOSEDBLACK (s)
6      for t ∈ STACK do
7          MASK (t)
8          TRYBACKWARD (s, t)
9          REVEAL (t)
10 end
```

3.1.3. The TryClosedWhite procedure

It handles the case where the current edge s is incident to a univalent white vertex (case 1 above), as the following picture shows.

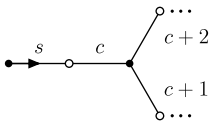
It simply builds a fixed point on s in the white permutation (line 2 of the procedure) then it calls the Recurse procedure.

Algorithm 3.3: TryClosedWhite (s : *integer*)

```
1  begin
2  |  s1[ s ] ← s
3  |  Recurse ( )
4  end
```

3.1.4. The TryForward procedure

Its purpose is to handle the case where the current edge s is incident to a bivalent white vertex followed by a trivalent black vertex (case 2 above) as the following picture shows.



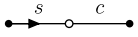
The edges incident to that trivalent black vertex are supposed never to have been encountered before and are then labeled by the values c, $c+1$ and $c+2$ (the case where the adjacent edge has already been encountered and thus has already a label is handled by the TryBackward procedure). Lines 2–6 build the corresponding black and white cycles in the permutation arrays. Among the three created edges, two need further exploration on their white side; so they are put on the stack by the Push instructions lines 7 and 8. Before the Recurse procedure is called, the labeling counter c is increased to account for the creation of the three new edges. The state of the stack and the labeling counter are both restored to their previous value by the instructions in lines 11–13, before the procedure exits.

Algorithm 3.4: TryForward (s : *integer*)

```
1  begin
2  |  s0[ c ] ← c + 1
3  |  s0[ c + 1 ] ← c + 2
4  |  s0[ c + 2 ] ← c
5  |  s1[ s ] ← c
6  |  s1[ c ] ← s
7  |  Push (c + 1)
8  |  Push (c + 2)
9  |  c ← c + 3
10 |  Recurse ( )
11 |  c ← c − 3
12 |  Pop ( )
13 |  Pop ( )
14 end
```

3.1.5. The TryClosedBlack procedure

It handles the case where the current edge s is incident to a bivalent white vertex followed by a univalent black vertex (case 3 above), as the following picture shows.



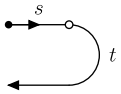
It builds the white 2-cycle and the black 1-cycle in lines 2–4 then the labeling counter c is increased in line 5 to account for the creation of the new edge labeled c. It calls the Recurse procedure line 6 and then restores the value of the labeling counter before it exits.

Algorithm 3.5: TryClosedBlack (s : *integer*)

```
1 begin
2 |  s_1[s] ← c
3 |  s_1[c] ← s
4 |  s_0[c] ← c
5 |  c ← c + 1
6 |  Recurse ( )
7 |  c ← c − 1
8 end
```

3.1.6. The TryBackward procedure

It handles the case where the current edge s is incident to a bivalent white vertex followed by an edge t that has already been visited (case 4 above), as the following picture shows.



The edge t is chosen in the Dispatch procedure and is removed from the stack before the TryBackward procedure is called and reinstalled back after the procedure terminates. The procedure simply binds together the two edges s and t by building a 2-cycle in the white permutation and then calls the Recurse procedure.

Algorithm 3.6: TryBackward (s, t : *integer*)

```
1 begin
2 |  s_1[s] ← t
3 |  s_1[t] ← s
4 |  Recurse ( )
5 end
```

3.1.7. The Recurse procedure

Its purpose is to check for the termination of the recursion. If the stack is empty, then the recursion terminates and it calls the Output procedure. In that case, the two arrays s_0 and s_1 describe a finished rooted trivalent diagram with edges labeled from 1, the root edge, to $c - 1$, the last attributed label. The size of the diagram is thus $c - 1$. Otherwise, when the stack is not empty, an edge is popped out of the stack and the recursion continues by a call to the Dispatch procedure in line 7.

Algorithm 3.7: Recurse ( )

```
1 local k : integer
2 begin
3 |  if StackIsEmpty ( ) then
4 |  |  Output ( )
5 |  else
6 |  |  k ← Pop ( )
7 |  |  Dispatch (k)
8 |  |  Push (k)
9 end
```

3.2. Correctness

In this section we show that the algorithm is correct and provide a complexity analysis of its execution time. This complexity analysis is based on a study of the structure of the execution tree of the algorithm and relies on the assumption that it is finite. We first prove that assumption.

Lemma 3.1. *The generating algorithm terminates in a finite amount of time.*

Proof. Looking at the procedures we see that each of them takes only a finite time to complete provided the procedures that are called also take in turn a finite time to complete. Therefore the proof reduces to showing that only a finite number of procedures is called, meaning that the execution tree of the algorithm is finite. Using Kőnig's infinity lemma on trees [12] one has to show that all the branches of that tree are finite. We show that there is a non-negative integer quantity

that is strictly decreasing along every branch. If we denote by n_s the size of the stack, then $\mu = 2\,n_s + n - c + 1$ is such a quantity, where n is the maximum size of the diagrams being generated and c is the labeling counter of the algorithm. It is non-negative because n_s is non-negative and because according to the "if" conditions of the Dispatch procedure, $n - c + 1$ is also non-negative, meaning that no label exceeding n is ever attributed to an edge. In the following table we summarize the changes in the value of n_s, c and μ after a cycle Dispatch → Try– → Recurse → Dispatch has been completed in each of the four cases described in Section 3.1.2 (cf. Fig. 3).

	Case 1	Case 2	Case 3	Case 4
n_s'	$n_s - 1$	$n_s + 1$	$n_s - 1$	$n_s - 2$
c'	c	$c + 3$	$c + 1$	c
μ'	$\mu - 2$	$\mu - 1$	$\mu - 3$	$\mu - 4$

In each case, $\mu' < \mu$ so the quantity μ is strictly decreasing along every branch of the execution tree. □

Lemma 3.2. *The rooted trivalent diagrams produced by the generating algorithm are labeled according to the characteristic labeling of Section 2.*

Proof. The proof is by Induction. Assuming that the stack, the labeling counter and the labels of the cycles already generated agree with the corresponding state of the labeling procedure of Section 2 at the start of a call to the Dispatch procedure (induction hypothesis) one can check that the execution of the algorithm through each of the four Try– procedures (Algorithms 3.4 and 3.5) each preserves that hypothesis, that is the new cycles introduced in the permutations s_0 and s_1 are labeled consistently with that of the labeling procedure and that the state of the stack also matches the one found in the labeling after visiting those new cycles. Finally, one has to check that the two base cases produced by the Generate procedure are also consistent with the characteristic labeling. This is immediate and completes the induction. □

Theorem 3.3. *The generating algorithm produces an exhaustive and non-redundant list of rooted trivalent diagrams.*

Proof. Exhaustivity comes primarily by induction from the local exhaustivity claim of the case analysis of Section 3.1.2. The mutual exclusion of the cases ensures that different rooted diagrams are produced, at least differing in the way they are labeled (the labeling counter is strictly increasing during the generation process of a structure), but since the structures are produced in their characteristic labeling, none of them could be isomorphic. □

3.3. Average time complexity

In this section we prove the main property of the algorithm, that it spends a constant amortized time generating each structure. One way to do the proof could be to express an estimate of the total execution time and the number of structures produced and show that the quotient of those two quantities is bounded independently of the size of the produced structures. We propose instead a proof of the majoration based on the following principle and a careful analysis of the execution tree of the generating algorithm.

Balance Principle. *In a finite tree, the number of leaves is greater than the number of its nodes having degree at least* 2.

Proof. The proof is by an easy recurrence on the number of internal nodes. Every finite tree can be constructed starting from a one-node tree by successive replacement of a leaf by an internal node having only leaves as sons. The starting tree satisfies the balance principle; so the recurrence is initialized. Assuming by recurrence that a finite tree satisfies the principle, and replacing one of its leaves by a internal node having $k \geq 1$ leaves as sons, one increases the number of internal nodes by 1 and the number of leaves by $k - 1 \geq 0$. If $k \geq 2$ the number of nodes having degree at least 2 is increased by 1, but the number of leaves is also increased by $k - 1 \geq 1$; so the resulting tree still satisfies the principle. The recurrence is complete. □

Lemma 3.4. *The total execution time of the generating algorithm is $O(C_n)$, where C_n is the number of procedures called during the execution.*

Proof. The only procedure of the algorithm that contains a loop and thus can have an arbitrarily long execution time is the Dispatch procedure. Since each iteration of the loop has a constant execution time and since the TryBackward procedure is called each time, we can transfer the cost of the iteration to the TryBackward procedure and then assume the Dispatch procedure to have a bounded execution time. In this way every procedure is assumed to have a bounded execution time so that the total execution time is proportional to C_n. □

Let D_n, R_n, O_n denote respectively, the total number of times the Dispatch, Recurse and Output procedures are called and let T_n denote the total number of times one of the four Try– procedures is called, where n is the maximum size of the structures being produced.

Lemma 3.5. *We have $D_n \leq 2\,O_n$.*

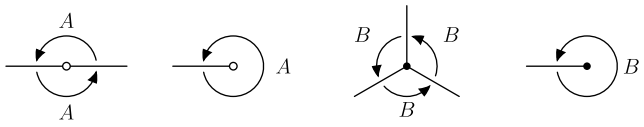


Fig. 4. A picture of the result of the action of the two elementary moves A and B on the edges incident to the various sorts of vertices.

Proof. Let D'_n denote the number of calls to the Dispatch procedure that have an out-degree at most 2. The leaves of the execution tree of the algorithm are calls to the Output procedure because the other procedures all have out-degree at least 1, hence by the balance principle above $D'_n \leq O_n$. The Dispatch procedure has an out-degree at least one because it calls the TryClosedWhite procedure unconditionally (line 3), and when its out-degree is 1 it means that the stack is empty (line 6) and that $c \leq n$ (lines 4 and 5). This means that all the n edges have been labeled and that the call to the Recurse procedure subsequent to the call to the TryClosedWhite will result in a call to the Output procedure and no further call to the Dispatch procedure. Therefore we see that the Dispatch procedure can have an out-degree of 1 but that can only happen once at the end of each branch. We thus have $D_n \leq D'_n + O_n$ and then $D_n \leq 2\,O_n$. □

Theorem 3.6 (*CAT Property*)**.** *The average time spent by the generating algorithm to produce each structure is bounded independently of its size.*

Proof. Let C_n denote the total number of procedures called during the execution of the algorithm. The number of structures produced by the algorithm is equal to O_n. Since the total execution time of the algorithm is $O(C_n)$ the average time spent producing each structure is $O(C_n/O_n)$. We have to show that the quotient C_n/O_n is bounded independently of n. We clearly have $C_n = 1 + D_n + R_n + T_n + O_n$: the $+1$ accounts for the first call to the Generate procedure. Since the Recurse procedure is only called by one of the four Try– procedures and each one calls it exactly once we have $T_n = R_n$. Since the Recurse procedure is called twice by the Generate procedure and the Recurse procedure calls one of the Recurse or Output procedures, we have $R_n + 2 = D_n + O_n$. Therefore we have,

$$\begin{aligned} C_n &= 1 + D_n + R_n + T_n + O_n \\ &= 1 + D_n + 2\,R_n + O_n && \text{as } T_n = R_n, \\ &= 3\,D_n + 3\,O_n - 3 && \text{as } R_n + 2 = D_n + O_n, \\ &\leq 9\,O_n && \text{as } D_n \leq 2\,O_n, \end{aligned}$$

and then $C_n/O_n \leq 9$. The bound on the quotient is independent of n as announced. □

4. First application: modular group and unrooted trivalent diagrams

We recall that the modular group $\mathrm{PSL}_2(\mathbb{Z})$ is the following group of 2 by 2 integer matrices with unit determinant:

$$\mathrm{PSL}_2(\mathbb{Z}) = \left\{ \pm \begin{pmatrix} a & b \\ c & d \end{pmatrix} \in \mathcal{M}_2(\mathbb{Z})/\pm \mathrm{Id} \;\middle|\; ad - bc = 1 \right\}.$$

There are many possible finite presentations for this group and we shall stick to the following:

$$\mathrm{PSL}_2(\mathbb{Z}) = \langle A, B \,|\, A^2 = B^3 = 1 \rangle$$

with A and B being the following two matrices:

$$A = \pm \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \quad \text{and} \quad B = \pm \begin{pmatrix} 0 & 1 \\ -1 & 1 \end{pmatrix},$$

for it renders explicit the following isomorphism:

$$\mathrm{PSL}_2(\mathbb{Z}) \simeq \mathbb{Z}/2\mathbb{Z} * \mathbb{Z}/3\mathbb{Z}.$$

4.1. Displacements groups

The modular group acts naturally on the set of edges of any trivalent diagram. This action is generated by the two *elementary moves* (Fig. 4),

$$a \cdot A = \sigma_\circ(a) \quad \text{and} \quad a \cdot B = \sigma_\bullet(a).$$

The elementary move A acts by exchanging positions of the two edges incident to any bivalent white vertex and by fixing the only edge incident to any univalent white vertex. Similarly, the elementary move B acts by cyclically permuting the three edges incident to any trivalent black vertex and by fixing the only edge incident to any univalent black vertex.

Given any trivalent diagram Γ, the two elementary moves just described generate a group Φ_Γ called the *displacement group* of Γ. It is easily verified that it is the quotient group of $\mathrm{PSL}_2(\mathbb{Z})$ by the kernel of the group action $\rho : \mathrm{PSL}_2(\mathbb{Z}) \to \mathfrak{S}_{\Gamma_-}$.

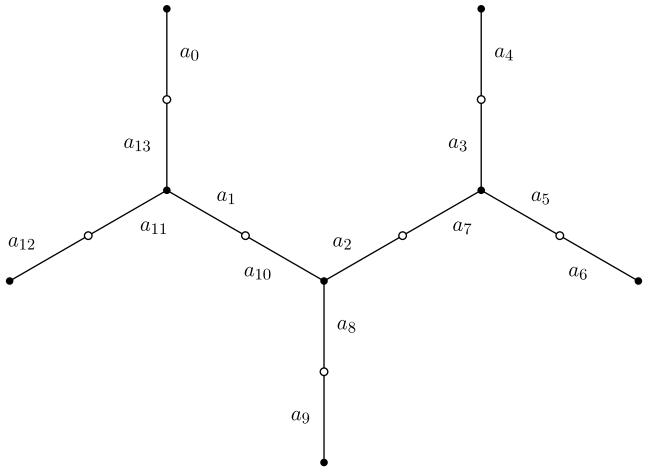


Fig. 5. We see in this example the result of iterating the elementary move T on the edges of a binary tree. The edges are labeled by a_k where $a_{k+1} = a_k \cdot T$. This can be used to implement depth-first traversals in a purely iterative way.

The modular group has therefore a universal status with respect to that construction: it can be considered as the universal group of displacements for the species of trivalent diagrams. If one restricts one's attention to finite trivalent diagrams, the profinite completion of $\mathrm{PSL}_2(\mathbb{Z})$ would be a more appropriate candidate for that purpose.

4.2. Unrooted plane binary trees

One can associate to any (unrooted) plane binary tree Θ a connected and acyclic trivalent diagram Γ, called its *enriched barycentric subdivision* $\Gamma = \Theta^{sb+}$, by putting an extra white vertex in the middle of every edge of Θ. The set of directed edges of Θ and that of undirected edges of Γ are in bijection in two natural ways.

There is another famous presentation of the modular group. It is given by two generators S and T and two relations as follows,

$$\mathrm{PSL}_2(\mathbb{Z}) = \langle S, T \mid S^2 = (ST)^3 = 1 \rangle,$$

with S and T being the following two matrices:

$$S = \pm \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \quad \text{and} \quad T = \pm \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}.$$

The conversion between the two presentations is done through the application of the following rules:

$$A \to S, \qquad S \to A,$$
$$B \to (ST)^2, \qquad T \to AB^{-1}.$$

Here are two basic criteria relating connectedness and acyclicity of finite trivalent diagrams to the transitivity of the action of some displacement groups:

(1) A finite trivalent diagram Γ is *connected* if and only if its displacement group Φ_Γ acts transitively on its set of edges.
(2) If it is a tree, then the subgroup Ψ_Γ of its displacements generated by the elementary move T acts transitively on its set of edges.

There is a natural bijection between trivalent diagrams having no univalent white vertex and those having no univalent black vertex. It works by removing every univalent black vertex and the adjacent edges in one direction and by growing every univalent white vertex with a new edge and a new univalent black vertex in the other direction. This bijection is compatible with connectedness and acyclicity and we thus recover by restriction the classical bijection between complete and incomplete plane binary trees (Fig. 5).

4.3. Classification principle

To any connected rooted trivalent diagram, one can moreover associate the subgroup of $\mathrm{PSL}_2(\mathbb{Z})$ consisting of elements that fix the distinguished edge of the diagram. We proved in [26] that this correspondence is one to one and we presented an enumeration in the form of a generating series that agrees perfectly with the number of structures generated by Algorithm 3.1.

Table 1
Trivalent diagrams of size up to five, up to isomorphism.

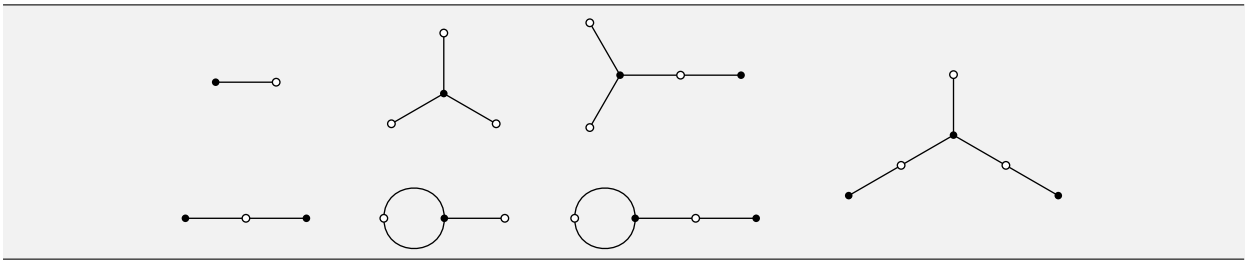


Table 2
Trivalent diagrams of size six, up to isomorphism.

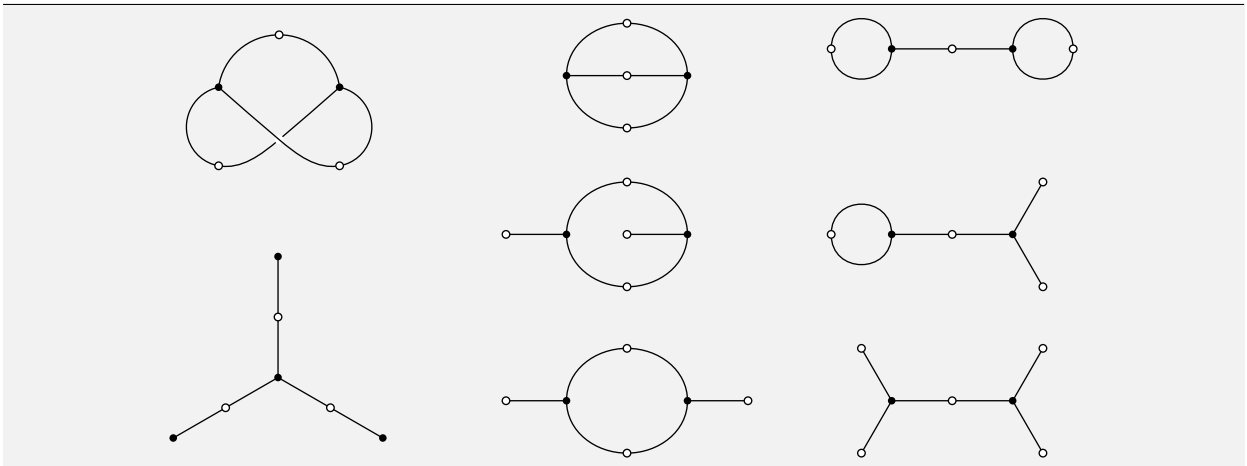
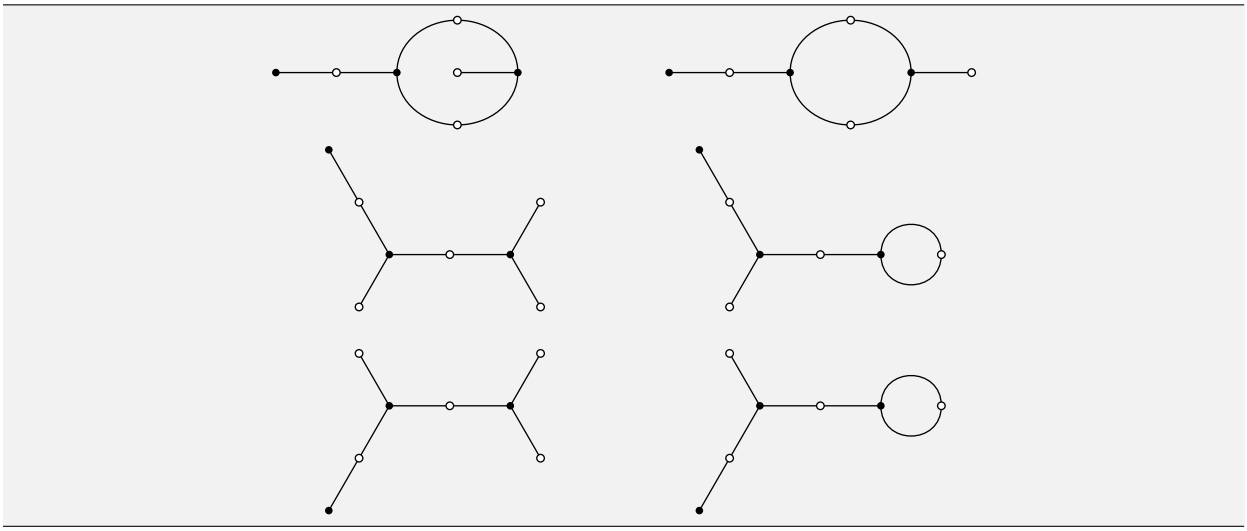


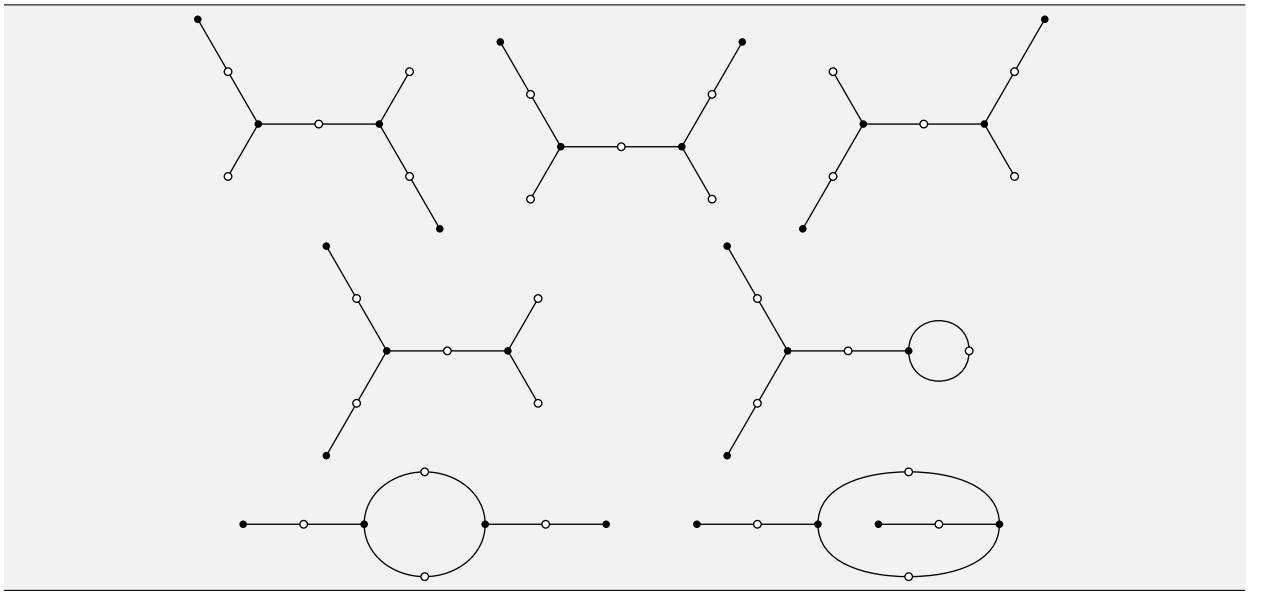
Table 3
Trivalent diagrams of size seven, up to isomorphism.



Now, if one changes the distinguished edge of a rooted trivalent diagram, the corresponding subgroups get conjugated. We have moreover proved that two subgroups in the modular group are conjugated if and only if the associated rooted trivalent diagrams differ only in the position of their distinguished edges. It follows that the unrooted trivalent diagrams correspond in a one-to-one fashion to the conjuguacy classes of subgroups of the modular group.

We gave in [26] the exhaustive list of trivalent diagrams of size up to nine. That list was computed by hand in a non-systematic fashion. One intent of Algorithm 3.1 of Section 3.1 is to permit a retroactive validation of both those tables and

Table 4
Trivalent diagrams of size eight, up to isomorphism.



the associated generating series that we reproduce here in Tables 6 and 7; they are parts of the online encyclopedia of integer sequences [22] under the references (A005133) and (A121350).

We generated unrooted trivalent diagrams using a method already described in [30]: by imposing a linear order on the set of rooted trivalent diagrams with a given number of edges, generating them all and accepting only those that are minimal, according to the linear order, among all those that differ only in the position of their root. Tables 1–5 show all the unrooted trivalent diagrams with up to 9 edges.

5. Second application: triangular maps

By an (*oriented*) *triangular map* we mean a finite polyhedral structure composed of vertices, directed edges, and oriented triangular faces with an incidence relation between them. Suggestively enough, a directed edge, also called an *arc*, is bordered by an ordered pair of vertices, which we call its *origin* and its *destination*, respectively, such that triangular faces are each bordered by a cycle of three arcs whose destination coincides with the origin of the following arc in cyclic order.

The following definition is useful in grasping the incidence relations of a combinatorial map but insufficient because it lacks some traversal information such as the cyclic orientation of the faces.

Definition 5.1. A *combinatorial pre-map* Λ is given by three sets Λ_0 Λ_1 and Λ_2 and five mappings $s, t : \Lambda_1 \to \Lambda_0$, $\ell, r : \Lambda_1 \to \Lambda_2$ and $.^{-1} : \Lambda_1 \to \Lambda_1$ satisfying the following conditions for all elements a of Λ_1,

$$s(a^{-1}) = t(a) \qquad \ell(a^{-1}) = r(a) \qquad (a^{-1})^{-1} = a$$
$$t(a^{-1}) = s(a) \qquad r(a^{-1}) = \ell(a) \qquad a^{-1} \neq a.$$

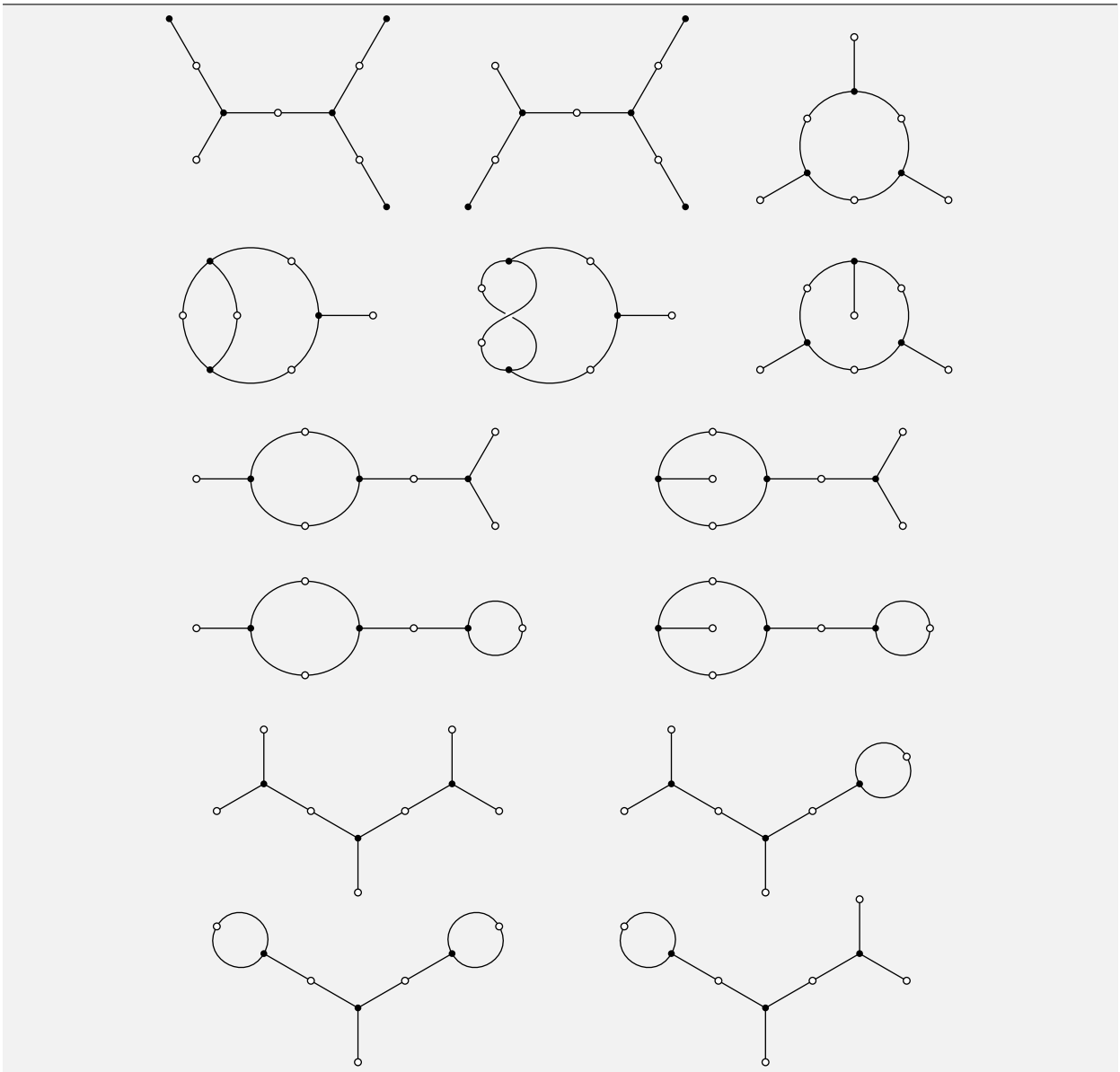
The four mappings s, t, ℓ and r are further assumed to be *surjective*.

The elements of the three sets Λ_0, Λ_1, and Λ_2 are the *vertices*, the *arcs* (directed edges), and the *faces* of the combinatorial map, respectively. The two mappings s and t map any arc a to its *origin* $s(a)$ and *destination* $t(a)$. The two mappings ℓ and r map any arc a to its *left-hand face* $\ell(a)$ and *right-hand face* $r(a)$. Finally, the mapping $.^{-1}$ maps any arc a to its *inverse* a^{-1} obtained by reversing its direction.

Definition 5.2. A *morphism* φ between two combinatorial pre-maps Λ and Λ' is a triple of mappings $\varphi_0 : \Lambda_0 \to \Lambda'_0$, $\varphi_1 : \Lambda_1 \to \Lambda'_1$ and $\varphi_2 : \Lambda_2 \to \Lambda'_2$ compatible with the five structure mappings in the sense that the following diagrams are commutative.

$$\begin{array}{ccc} \Lambda_1 & \xrightarrow{\varphi_1} & \Lambda'_1 \\ {\scriptstyle s,t}\downarrow & & \downarrow{\scriptstyle s,t} \\ \Lambda_0 & \xrightarrow[\varphi_0]{} & \Lambda'_0 \end{array} \qquad \begin{array}{ccc} \Lambda_1 & \xrightarrow{\varphi_1} & \Lambda'_1 \\ {\scriptstyle \ell,r}\downarrow & & \downarrow{\scriptstyle \ell,r} \\ \Lambda_2 & \xrightarrow[\varphi_2]{} & \Lambda'_2 \end{array} \qquad \begin{array}{ccc} \Lambda_1 & \xrightarrow{\varphi_1} & \Lambda'_1 \\ {\scriptstyle .^{-1}}\downarrow & & \downarrow{\scriptstyle .^{-1}} \\ \Lambda_1 & \xrightarrow[\varphi_1]{} & \Lambda'_1 \end{array}$$

Table 5
Trivalent diagrams of size nine, up to isomorphism.



When the three mappings are bijections the morphism is an *isomorphism*.

5.1. Cyclic orientation

In a given combinatorial pre-map Λ, the *inner border* of a face f is the set $\ell^{-1}(f) = \{ a \in \Lambda_1 \mid \ell(a) = f \}$ of arcs having f as their left-hand face. A combinatorial pre-map is said to be *strictly triangular* if each of its faces has exactly three arcs in its inner border. The following definition describes the traversal information lacking from a triangular combinatorial pre-map to fully describe a triangular map.

Definition 5.3. A (*strictly*) *triangular map* Λ is a strictly triangular combinatorial pre-map together with a cyclic order imposed on the three arcs that have the same left-hand face: $a + 1$ denotes the next arc after a in this cyclic order.

Definition 5.4. A *morphism* φ (respectively, an *isomorphism*) between two triangular maps Λ and Λ' is a morphism (respectively, an isomorphism) of the underlying combinatorial pre-maps that preserves the cyclic order of the previous definition.

Table 6
Order fifty development of the generating series $\tilde{D}_3^{\bullet}(t)$ giving as the coefficient of t^n the number of connected rooted trivalent diagrams with n edges (A005133) which is also the number of index n subgroups in the modular group $\mathrm{PSL}_2(\mathbb{Z})$.

$$\begin{aligned}\tilde{D}_3^{\bullet}(t) = {} & t + t^2 + 4\,t^3 + 8\,t^4 + 5\,t^5 + 22\,t^6 + 42\,t^7 + 40\,t^8 + 120\,t^9 + 265\,t^{10} + 286\,t^{11} \\ & + 764\,t^{12} + 1729\,t^{13} + 2198\,t^{14} + 5168\,t^{15} + 12144\,t^{16} + 17034\,t^{17} + 37702\,t^{18} \\ & + 88958\,t^{19} + 136584\,t^{20} + 288270\,t^{21} + 682572\,t^{22} + 1118996\,t^{23} \\ & + 2306464\,t^{24} + 5428800\,t^{25} + 9409517\,t^{26} + 19103988\,t^{27} + 44701696\,t^{28} \\ & + 80904113\,t^{29} + 163344502\,t^{30} + 379249288\,t^{31} + 711598944\,t^{32} \\ & + 1434840718\,t^{33} + 3308997062\,t^{34} + 6391673638\,t^{35} + 12921383032\,t^{36} \\ & + 29611074174\,t^{37} + 58602591708\,t^{38} + 119001063028\,t^{39} \\ & + 271331133136\,t^{40} + 547872065136\,t^{41} + 1119204224666\,t^{42} \\ & + 2541384297716\,t^{43} + 5219606253184\,t^{44} + 10733985041978\,t^{45} \\ & + 24300914061436\,t^{46} + 50635071045768\,t^{47} + 104875736986272\,t^{48} \\ & + 236934212877684\,t^{49} + 499877970985660\,t^{50} + o(t^{50})\end{aligned}$$

Table 7
Order fifty development of the generating series $\tilde{D}_3(t)$ giving as the coefficient of t^n the number of connected unrooted trivalent diagrams with n edges (A121350) which is also the number of conjugacy classes of index n subgroup in the modular group $\mathrm{PSL}_2(\mathbb{Z})$.

$$\begin{aligned}\tilde{D}_3(t) = {} & t + t^2 + 2\,t^3 + 2\,t^4 + t^5 + 8\,t^6 + 6\,t^7 + 7\,t^8 + 14\,t^9 + 27\,t^{10} + 26\,t^{11} \\ & + 80\,t^{12} + 133\,t^{13} + 170\,t^{14} + 348\,t^{15} + 765\,t^{16} + 1002\,t^{17} + 2176\,t^{18} \\ & + 4682\,t^{19} + 6931\,t^{20} + 13740\,t^{21} + 31085\,t^{22} + 48652\,t^{23} + 96682\,t^{24} \\ & + 217152\,t^{25} + 362779\,t^{26} + 707590\,t^{27} + 1597130\,t^{28} + 2789797\,t^{29} \\ & + 5449439\,t^{30} + 12233848\,t^{31} + 22245655\,t^{32} + 43480188\,t^{33} \\ & + 97330468\,t^{34} + 182619250\,t^{35} + 358968639\,t^{36} + 800299302\,t^{37} \\ & + 1542254973\,t^{38} + 3051310056\,t^{39} + 6783358130\,t^{40} + 13362733296\,t^{41} \\ & + 26648120027\,t^{42} + 59101960412\,t^{43} + 118628268978\,t^{44} \\ & + 238533003938\,t^{45} + 528281671324\,t^{46} + 1077341937144\,t^{47} \\ & + 2184915316390\,t^{48} + 4835392099548\,t^{49} + 9997568771074\,t^{50} + o(t^{50})\end{aligned}$$

5.2. Associated trivalent diagram

To any triangular map Λ one can associate in a natural way a trivalent diagram $\Gamma = \Lambda^{inc}$ which is called the *incidence diagram* of the triangular map. To any face of Λ one associates a black vertex of Γ and to any edge of Λ one associates a white vertex of Γ; then one connects black and white vertices of Γ by an edge if the corresponding face and edge are incident in Λ. The cyclic orientations around black vertices is the one given by the cyclic orientation of the corresponding triangular faces of Λ. Table 8 gives three simple examples of the correspondence. This operation has already been described by Walsh in [29], namely: taking the face-vertex dual of Λ one gets a trivalent map and then applying the construction of Walsh, one gets the incidence diagram Γ.

The *incidence diagram* of a triangular map Λ is the trivalent diagram, denoted by Λ^{inc}, whose sets of white vertices, black vertices and edges are the following,

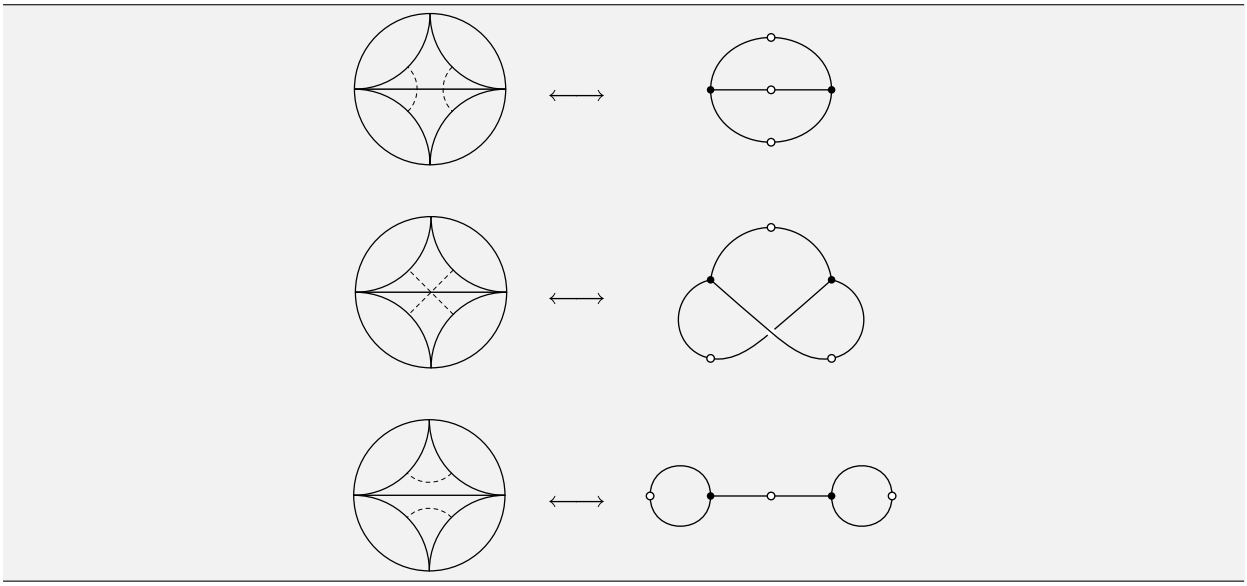
$$\Lambda_{\circ}^{inc} = \{\mathfrak{a}_a\}_{a\in\Lambda_1^*}, \qquad \Lambda_{\bullet}^{inc} = \{\mathfrak{b}_f\}_{f\in\Lambda_2}, \qquad \Lambda_{-}^{inc} = \{\mathfrak{c}_a\}_{a\in\Lambda_1},$$

where Λ_1^* is the set of undirected edges of Λ, and whose structure mappings $\partial_{\circ} : \Lambda_{-}^{inc} \to \Lambda_{\circ}^{inc}$, $\partial_{\bullet} : \Lambda_{-}^{inc} \to \Lambda_{\bullet}^{inc}$ and the permutations $\sigma_{\bullet}$ and $\sigma_{\circ}$ of Λ_{-}^{inc} are defined by the following relations,

$$\begin{aligned}\partial_{\circ}(\mathfrak{c}_a) &= \mathfrak{a}_{\pi(a)}, & \sigma_{\circ}(\mathfrak{c}_a) &= \mathfrak{c}_{a^{-1}}, \\ \partial_{\bullet}(\mathfrak{c}_a) &= \mathfrak{b}_{\ell(a)}, & \sigma_{\bullet}(\mathfrak{c}_a) &= \mathfrak{c}_{a+1},\end{aligned}$$

where $\pi(a)$ is the undirected version of the arc a. This operation is functorial for it is easily extended to morphisms of triangular maps by the following process: to any morphism φ between two triangular maps Λ and Λ', we associate a morphism denoted by φ^{inc} between the corresponding incidence diagrams Λ^{inc} and $(\Lambda')^{inc}$. It is given by the following relations,

Table 8
The three triangular maps with two faces together with their incidence diagrams. The triangular map in the middle row corresponds to the only way to build a torus using two triangles, and the two others, top and bottom rows, correspond to the two different ways to build a sphere using two triangles.



$$\varphi_{\circ}^{inc}(\mathfrak{a}_a) = \mathfrak{a}'_{\varphi_1^*(a)}, \qquad \varphi_{\bullet}^{inc}(\mathfrak{b}_f) = \mathfrak{b}'_{\varphi_2(f)}, \qquad \varphi_{-}^{inc}(\mathfrak{c}_a) = \mathfrak{c}'_{\varphi_1(a)}.$$

Functoriality should be obvious by careful inspection.

The $.^{inc}$ functor we get by what precedes is full and faithful [17, p. 14 and 15], but not essentially surjective[1] in that trivalent diagrams we get as the incidence diagram of a trivalent map Λ having no univalent white vertex nor univalent black vertex. We call a trivalent diagram *regular* if it has no univalent vertex. The $.^{inc}$ functor is essentially surjective on the full subcategory of regular trivalent diagrams, and so,

Theorem 5.1. *The $.^{inc}$ functor realizes an equivalence between the category of triangular maps and the full subcategory of regular trivalent diagrams.*

Proof. To prove this theorem we shall describe a *reconstruction* operation, which associates to any regular trivalent diagram Γ a triangular map denoted by Γ^{map} with functorial property and such that for all regular trivalent diagrams Γ and triangular maps Λ, one has the following two natural *reciprocity isomorphisms*,

$$(\Gamma^{map})^{inc} \underset{nat.}{\simeq} \Gamma \quad \text{and} \quad (\Lambda^{inc})^{map} \underset{nat.}{\simeq} \Lambda.$$

We shall first introduce some notations. We call Ψ_Γ the subgroup of Φ_Γ generated by the elementary move T (cf. Section 4.2) and denote the natural projection by $\pi : \Gamma_- \to \Gamma_-/\Psi_\Gamma$. The mapping induced between Γ_-/Ψ_Γ and Γ'_-/Ψ_Γ by an equivariant mapping $\varphi : \Gamma_- \to \Gamma'_-$ will be denoted by φ_Ψ. The sets of vertices, edges and faces of the reconstructed map are the following,

$$\Gamma_0^{map} = \{\mathfrak{d}_x\}_{x\in\Gamma_-/\Psi_\Gamma} \Gamma_1^{map} = \{\mathfrak{e}_a\}_{a\in\Gamma} \Gamma_2^{map} = \{\mathfrak{f}_y\}_{y\in\Gamma_\bullet}.$$

The five structure mappings $s, t : \Gamma_1^{map} \to \Gamma_0^{map}$, $r, \ell : \Gamma_1^{map} \to \Gamma_2^{map}$ and $.^{-1} : \Gamma_1^{map} \to \Gamma_1^{map}$ and the group action $+ : \Gamma_1^{map} \times \mathbb{Z} \to \Gamma_1^{map}$ of the reconstructed map are given by the following equations:

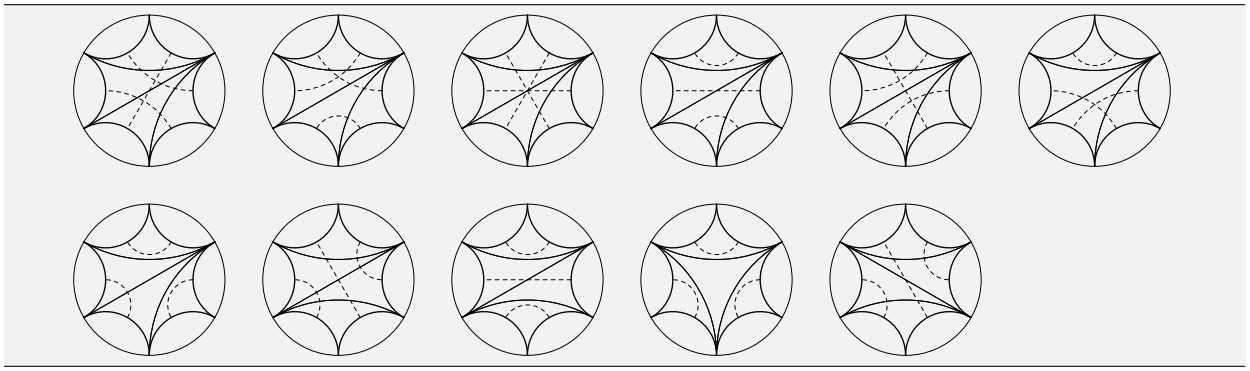
$$s(\mathfrak{e}_a) = \mathfrak{d}_{\pi(a)}, \qquad \ell(\mathfrak{e}_a) = \mathfrak{f}_{\partial_\bullet(a)}, \qquad \mathfrak{e}_a^{-1} = \mathfrak{e}_{\sigma_\circ(a)},$$
$$t(\mathfrak{e}_a) = \mathfrak{d}_{\pi(\sigma_\circ(a))}, \quad r(\mathfrak{e}_a) = \mathfrak{f}_{\partial_\bullet(a^{-1})}, \quad \mathfrak{e}_a + 1 = \mathfrak{e}_{\sigma_\bullet(a)}.$$

The construction then extends to morphisms in the sense that any morphism φ between two regular trivalent diagrams Γ and Γ' induces a morphism φ^{map} between the two reconstructed maps Γ^{map} and $(\Gamma')^{map}$ whose three components are the following,

$$\varphi_0^{map}(\mathfrak{d}_x) = \mathfrak{d}_{\varphi_{-,\Psi}(x)}, \qquad \varphi_1^{map}(\mathfrak{e}_a) = \mathfrak{e}_{\varphi_-(a)}, \qquad \varphi_2^{map}(\mathfrak{f}_y) = \mathfrak{f}_{\varphi_\bullet(y)}.$$

[1] A functor $F : A \to B$ is *essentially surjective* if for any objects $b \in B$ there is an object $a \in A$ such that $F(a)$ and b are isomorphic in B.

Table 9
The eleven triangular maps with four faces. According to Table 15, six of them are spheres while five of them are toruses. This is easily checked using Euler's formula $\chi_E = n_v - n_e + n_f = 2 - 2g$ [8, chap. 8, sec. 8a].



The functoriality of the reconstruction operation and the two reciprocity isomorphisms should be clear by careful inspection. □

Remark. The content of the above theorem is nothing but a special case of *Poincaré duality*.

5.3. Exhaustive generation of triangular maps

To adapt the generator algorithm to produce only regular rooted trivalent diagrams and thus rooted triangular maps, it suffices to remove the call to Algorithms 3.5 and 3.3 from lines 6 and 7 of Algorithm 3.2 and the lines 2–5 from Algorithm 3.1. Those two removals preserve the CAT property, we thus get a constant amortized time generator for regular rooted triangular maps, as announced.

Tables 8–11 show exhaustive lists of *unrooted* triangular maps produced from the output of the generator algorithm. An easy and efficient way to do that is described in [30]. Following the remarks in Section 2.3 we can put a linear order on the set of rooted trivalent diagrams of a given size. We then simply have to remove from the list the trivalent diagrams that are not minimal in their conjugacy class. Filtering out non-minimal representatives does not require storing a full list of generated diagrams. Since two conjugated rooted diagrams differ only in the position of their root, walking through the full conjugacy class of a rooted diagram is accomplished by successively moving its root to every possible position.

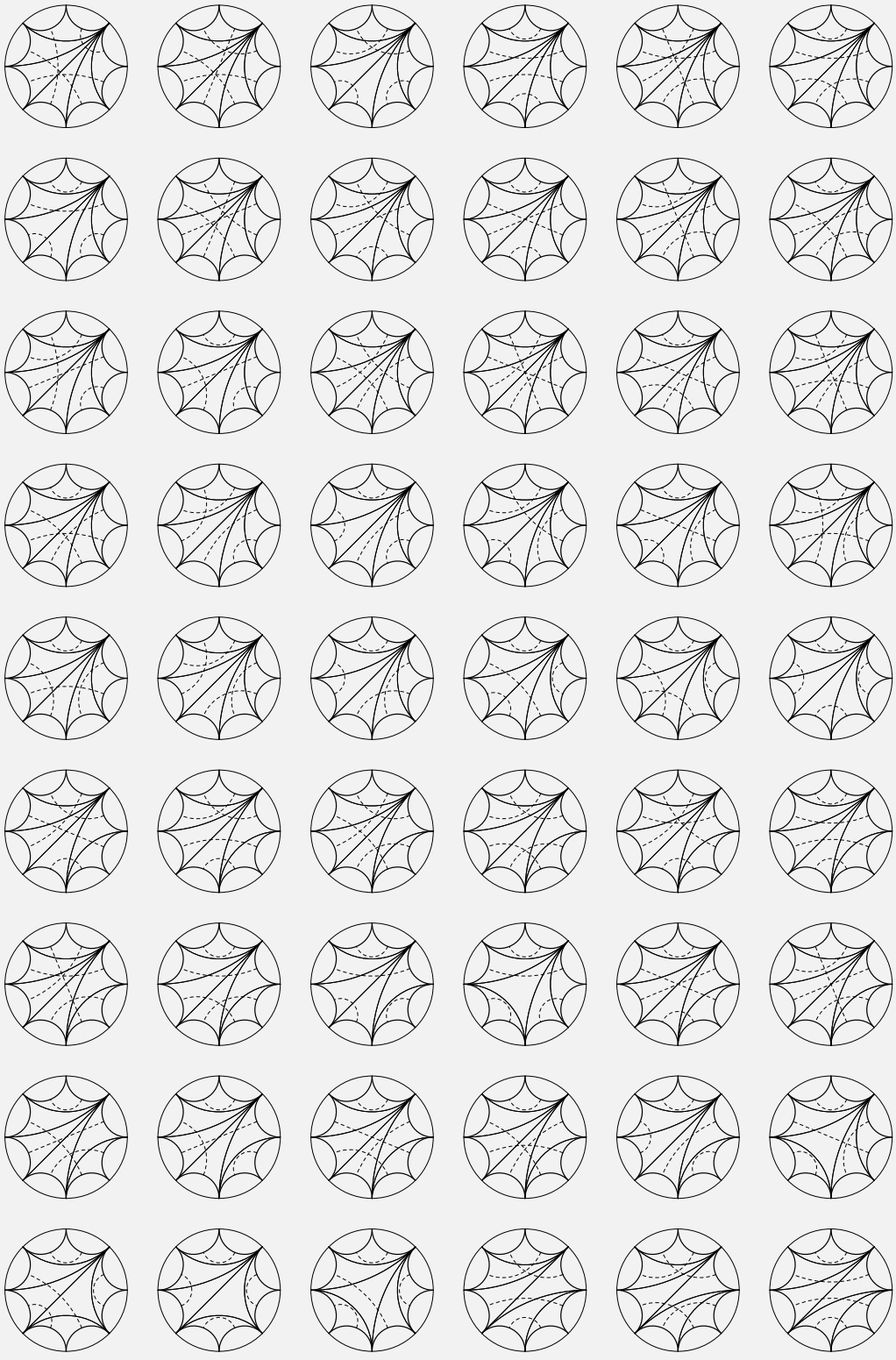
The interpretation of the drawings of Tables 8–11 requires some explanation. For that purpose we adopt a geographical terminology. Ignoring for a moment the surrounding circle and the dashed lines of the drawings, the triangular regions are called the countries of the maps and the solid lines are the boundaries of their incident countries. One can distinguish the boundaries that are bordering two distinct countries (the inner boundaries) from those that are bordering a single country (the outer boundaries). The roads of the maps are symbolized by dashed lines connecting, in a two-by-two fashion, the outer boundaries of the maps.

To produce them, we have considered the plane rooted binary tree of the depth-first traversal of Algorithm 2.1. As already noted, this algorithm provides natural cuttings for the associated trivalent diagrams. Those cuttings arise as what we previously called backward connections. In contrast, the edges of the traversal tree correspond to what we previously called forward connections. In the graphical representation, we use an embedding of the produced triangulated polygon in the Poincaré disc model of the hyperbolic plane as it seems the natural setting for generic non-overlapping triangular tilings. The surrounding circle around each figure is of course irrelevant to the structures.

Tables 12 and 13 give the number of rooted triangular maps and unrooted triangular maps in the form of generating series. They are also part of [22] under the references (A062980) and (A129114). Their computation is very similar to that of Tables 6 and 7, which is explained in detail in [26]. We shall explain in [25] in a unified fashion how one can compute generating series for rooted and unrooted unlabeled maps of various kind and in [24] the unexpected relation of this sequence to the asymptotics of the Airy function.

As another byproduct of the exhaustive list obtained from the generating algorithm, one can get the precise number of rooted and unrooted triangular maps having a given genus and a given number of triangular faces. Tables 14 and 15 summarize those results for a small number of faces. Recently, Krikun [15] kindly communicated to us the recurrence relations satisfied by the entries of Table 14, which he obtained by a clever recursive decomposition of rooted triangular maps. Those recurrence relations make it possible to evaluate easily those numbers without running the generator algorithm. Unfortunately, no general recurrence relation is known for the entries of Table 15. The thesis [27] and articles [31,29,28] contain the first enumerations of rooted maps of a given genus. Enumeration of unrooted maps by the number of edges and the genus were addressed in a recent article by Mednykh and Nedela [19].

Table 10
The eighty-one triangular maps with six faces (first part).



Some lines of those two tables were previously known. For instance, the first line of Table 14 is the number of spherical rooted triangular maps by the number of its faces [20]. The first line of Table 15 is its unrooted counterpart. It is computed

Table 11
The eighty-one triangular maps with six faces (last part, continued from Table 10).

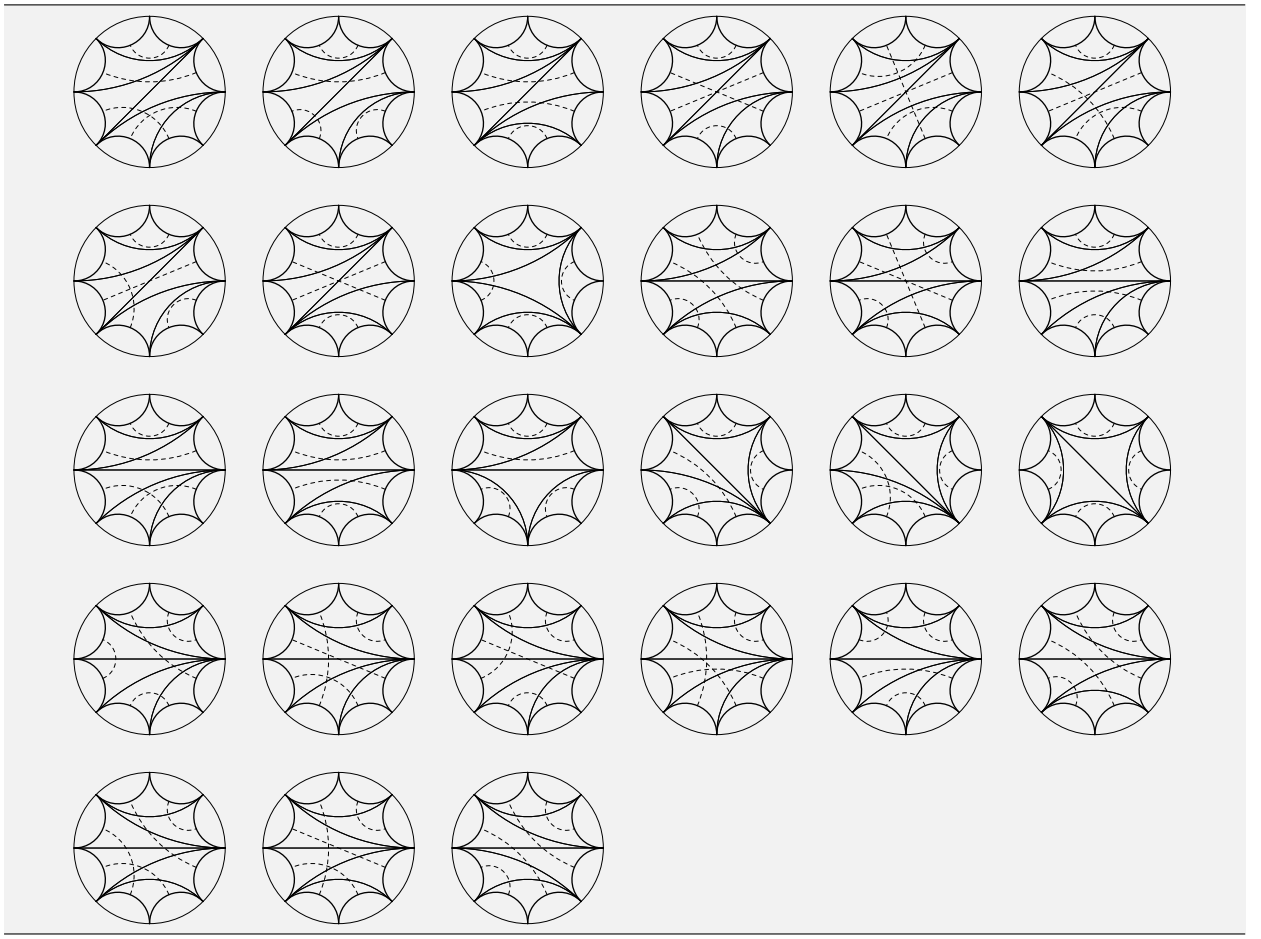


Table 12
Development of the generating series $\tilde{M}_3^{\bullet}(t)$, up to order one hundred and twenty. It gives as the coefficient of t^{6n} the number of connected unrooted unlabeled triangular maps with $6n$ arcs, thus $3n$ undirected edges and $2n$ triangular faces (A062980). If we denote by a_n that coefficient, the recurrence is as follows: $a_1 = 5$ and for $n \geq 1$, $a_{n+1} = (6n+6)\,a_n + \sum_{k=1}^{n-1} a_k\,a_{n-k}$.

$$\begin{aligned}\tilde{M}_3^{\bullet}(t) &= 5\,t^6 + 60\,t^{12} + 1105\,t^{18} + 27120\,t^{24} + 828250\,t^{30} + 30220800\,t^{36} \\ &+ 1282031525\,t^{42} + 61999046400\,t^{48} + 3366961243750\,t^{54} \\ &+ 202903221120000\,t^{60} + 13437880555850250\,t^{66} + 970217083619328000\,t^{72} \\ &+ 75849500508999712500\,t^{78} + 6383483988812390400000\,t^{84} \\ &+ 575440151532675686278125\,t^{90} + 55318762960656722780160000\,t^{96} \\ &+ 5649301494178851172304968750\,t^{102} \\ &+ 610768380520654474629120000000\,t^{108} \\ &+ 69692599846542054607811528918750\,t^{114} \\ &+ 8370071726919812448859648819200000\,t^{120} + o(t^{120})\end{aligned}$$

by impressive closed formulae in a recent paper by Liskovets et al. [18]. The diagonal terms of those two tables also received close attention. For instance, in [10] Harer and Zagier computed the Euler–Wall characteristic of the mapping class group of singly rooted genus g closed oriented surfaces by a remarkable combinatorial reduction of the problem in which rooted combinatorial maps with one vertex are counted by genus yielding the diagonal sequence of the first table 1, 105, 50 050, 56 581 525, ... The diagonal sequence of the second table: 1, 9, 172, 1 349 005, ... gives the number of unrooted triangular maps of genus g with only one vertex. It has been studied at depth in the article [1] by Vdovina and Bacher.

Table 13
Order one hundred and twenty development of the generating series $\tilde{M}_3(t)$ giving the number of connected unrooted unlabeled triangular maps with $6n$ arcs, thus $3n$ undirected edges and $2n$ triangular faces (A129114).

$$\begin{aligned}\tilde{M}_3(t) &= 3\,t^6 + 11\,t^{12} + 81\,t^{18} + 1228\,t^{24} + 28174\,t^{30} + 843186\,t^{36} + 30551755\,t^{42}\\ &+ 1291861997\,t^{48} + 62352938720\,t^{54} + 3381736322813\,t^{60}\\ &+ 203604398647922\,t^{66} + 13475238697911184\,t^{72} + 972429507963453210\,t^{78}\\ &+ 75993857157285258473\,t^{84} + 6393779463050776636807\,t^{90}\\ &+ 576237114190853665462712\,t^{96} + 55385308766655472416299110\,t^{102}\\ &+ 5655262782600929403228668176\,t^{108}\\ &+ 611338595145132827847686253456\,t^{114}\\ &+ 69750597724332100283681465962492\,t^{120} + o(t^{120})\end{aligned}$$

Table 14
The number of *rooted* triangular maps by genus (0–4) and number of faces (2–14).

	2	4	6	8	10	12	14
0	4	32	336	4 096	54 912	786 432	11 824 384
1	1	28	664	14 912	326 496	7 048 192	150 82 0608
2	0	0	105	8 112	396 792	15663360	544 475 232
3	0	0	0	0	50 050	6 722 816	518 329 776
4	0	0	0	0	0	0	56 581 525

Table 15
The number of *unrooted* triangular maps by genus (0–4) and number of faces (2–14).

	2	4	6	8	10	12	14
0	2	6	26	191	1 904	22 078	282 388
1	1	5	46	669	11 096	196 888	3 596 104
2	0	0	9	368	13 448	436 640	12 974 156
3	0	0	0	0	1 726	187 580	12 350 102
4	0	0	0	0	0	0	1 349 005

6. Concluding remarks and perspectives

The generating algorithm presented in this paper (Section 3) may lead to trivial adaptations to generate wider classes of diagrams and combinatorial maps, possibly with prescribed degree lists for vertices or faces. Basically it can be simply generalized to produce any connected pair of permutations with prescribed cyclic types, up to simultaneous conjugacy.

Another way to extend the study would be to modify the Dispatch procedure (Algorithm 3.2 of Section 3) to generate not an exhaustive cover of the partial cases, but instead a single case of them picked at random. This would result, in a fairly straightforward fashion, in a random sampler algorithm of the corresponding combinatorial structures instead of an exhaustive generator. The difficulty there, is to precompute precise conditional probability tables in order to control the probability distribution of the generated structures by bayesian techniques. Such tables of conditional probabilities could be computed with the help of generating series techniques, namely by following the particular recursive structure of the algorithm and translating this recursive structure into functional equations on the generating series. This calls for a further investigation and could be dealt with in a subsequent paper.

If one designs the Dispatch procedure to pick at random a case with uniform probability this would unfortunately not result in a uniform distribution of the output structures but it can still prove useful to produce test cases for many algorithms operating on triangulations. A much better approach but still not perfect is to generate at random the two permutations $\sigma_\bullet$ and $\sigma_\circ$ having the right cycle types with uniform distribution and rejecting each time the resulting map is not connected. This procedure produces each map with probability proportional to $1/A$ where A is the number of its automorphisms. Since the maps have no automorphism other than identity in the vast majority of cases, that statistical bias is in practice unnoticeable.

Acknowledgements

I am grateful to professors D. Bar Nathan, P. Flajolet, F. Hivert, M. Huttner, M. Krikun, M. Petitot, B. Salvy, G. Shaeffer, N. Thiery, and D. Zvonkine for useful discussions and warm encouragement. Comments and suggestions from the anonymous referees helped a lot to improve the clarity of the paper and the quality of presentation.

References

[1] R. Bacher, A. Vdovina, Counting 1-vertex triangulations of oriented surfaces, Discrete Math. 246 (1–3) (2002) 13–27.
[2] D. Bar-Nathan, On associators and the Grothendieck-Teichmuller group I, Selecta Math. (N.S.) 4 (1998) 183–212.
[3] F. Bergeron, G. Labelle, P. Leroux, Théorie des espèces et combinatoire des structures arborescentes, LACIM Montréal, 1994.
[4] F. Bergeron, G. Labelle, P. Leroux, Combinatorial Species and Tree-like Structures, Cambridge University Press, 1998, English edition of [3].
[5] R. Douady, A. Douady, Algèbre et théories galoisiennes, Cassini, France, 2003.
[6] V.G. Drinfel'd, Quasi-Hopf algebras, Leningrad Math. J. 1 (1990) 1419–1457.
[7] V.G. Drinfel'd, On quasitriangular quasi-Hopf algebras and a group closely connected with $\mathrm{Gal}(\bar{\mathbb{Q}}/\mathbb{Q})$, Leningrad Math. J. 2 (1991) 829–860.
[8] W. Fulton, Algebraic Topology, Springer, 1995.
[9] A. Grothendieck, Esquisse d'un programme, in: P. Lochak, L. Schneps (Eds.), Geometric Galois Actions Vol. I, in: London Math. Soc. Lecture Notes, vol. 242, Cambridge Univ. Press, 1997, pp. 5–48.
[10] J. Harer, D. Zagier, The Euler characteristic of the moduli space of curves, Invent. Math. 85 (1986) 457–486.
[11] A. Joyal, Une théorie combinatoire des séries formelles, Adv. Math. 42 (1981) 1–82.
[12] D. Kőnig, Sur les correspondances multivoques des ensembles, Fundamenta Math. 8 (1926) 114–134.
[13] D.E. Knuth, Dancing links, in: Millenial Perspectives in Computer Science, 2000, pages 187–214, Available online at: arXiv:cs/0011047v1 [cs.DS].
[14] M. Kontsevich, Intersection theory on the moduli space of curves and the matrix Airy function, Comm. Math. Phys. 147 (1992) 1–23.
[15] M. Krikun, Enumeration of triangulations by genus (incomplete draft), Private communication, 2007.
[16] S.K. Lando, A.K. Zvonkine, Graphs on Surfaces and Their Applications, Springer-Verlag, 2004.
[17] S. Mac Lane, Categories for the Working Mathematician, Springer, 1998.
[18] V.A. Liskovets, Z. Gao, N. Wormald, Enumeration of unrooted odd-valent regular planar maps, 2005 (in press).
[19] A. Mednykh, R. Nedela, Enumeration of unrooted maps of a given genus, J. Combin. Theory, Ser. B 96 (5) (2006) 706–729.
[20] R.C. Mullin, E. Nemeth, P.J. Schellenberg, The enumeration of almost cubic maps, in: R.C. Mullin, et al. (Eds.), Proceedings of the Louisiana Conference on Combinatorics, in: Graph Theory and Computer Science, vol. 1, 1970, pp. 281–295.
[21] L. Schneps, Dessins dıenfants on the Riemann sphere, in: P. Lochak, L. Schneps (Eds.), The Grothendieck Theory of Dessins d'Enfant, in: London Math. Soc. Lecture Notes, vol. 200, Cambridge Univ. Press, 1994, pp. 5–48.
[22] N.J.A. Sloane, The on-line encyclopedia of integer sequences, 2005. Available on the net at: http://www.research.att.com/~njas/sequences/.
[23] W. Thurston, Three-dimensional manifolds, Kleinian groups and hyperbolic geometry, Bull. Amer. Math. Soc. (N.S.) 6 (1982) 357–381.
[24] S.A. Vidal, M. Petitot, Counting rooted and unrooted triangular maps, J. Nonlinear Syst. Appl. (2010) (in press).
[25] S.A. Vidal, Multiparametric enumeration of unrooted combinatorial maps (in preparation).
[26] S.A. Vidal, Sur la classification et le dénombrement des sous-groupes du groupe modulaire et de leurs classes de conjugaison, Pub. IRMA, Lille, 66(II): 2006, 1–35. Available online at: arXiv:math/0702223v1 [math.CO].
[27] T.R.S. Walsh, Combinatorial enumeration of non-planar maps, Ph.D. Thesis, Univ. of Toronto, 1971.
[28] T.R.S. Walsh, Counting rooted maps by genus III, J. Combin. Theory 18 (2) (1975) 222–259.
[29] T.R.S. Walsh, Hypermaps versus bipartite maps, J. Combin. Theory 18 (1975) 155–163.
[30] T.R.S. Walsh, Generating nonisomorphic maps without storing them, SIAM J. Algebra. Discrete Methods 4 (2) (1983) 161–178.
[31] T.R.S. Walsh, A.B. Lehman, Counting rooted maps by genus, J. Combin. Theory 13 (1972) 122–141, 192–218.