Chapter 2
An Information-Theoretic Upper Bound on Planar Graphs Using Well-Orderly Maps

Nicolas Bonichon, Cyril Gavoille, and Nicolas Hanusse

Abstract This chapter deals with compressed coding of graphs. We focus on planar graphs, a widely studied class of graphs. A planar graph is a graph that admits an embedding in the plane without edge crossings. Planar maps (class of embeddings of a planar graph) are easier to study than planar graphs, but as a planar graph may admit an exponential number of maps, they give little information on graphs. In order to give an information-theoretic upper bound on planar graphs, we introduce a definition of a quasi-canonical embedding for planar graphs: well-orderly maps. This appears to be an useful tool to study and encode planar graphs. We present upper bounds on the number of unlabeled[1] planar graphs and on the number of edges in a random planar graph. We also present an algorithm to compute well-orderly maps and implying an efficient coding of planar graphs.

Keywords Compact coding • Enumerative combinatorics • Planar embedding • Planar graph

MSC2000 Primary 05C10; Secondary 05C10, 05C30, 05C85.

1 Introduction

In graph theory, a planar graph is a graph which can be embedded in the plane, i.e., it can be drawn on the plane in such a way that its edges intersect only at their endpoints. A planar graph drawn in the plane without edge intersections is called a

[1]Nodes and edges are not assumed to be labeled.

N. Bonichon (✉)
LaBRI, University of Bordeaux, 351 Cours de la libération, 33405 Bordeaux, France
e-mail: bonichon@labri.fr

M. Dehmer et al. (eds.), *Towards an Information Theory of Complex Networks: Statistical Methods and Applications*, DOI 10.1007/978-0-8176-4904-3_2,

planar map or a planar embedding of the graph. The class of planar graphs is one of the most studied graphs.

How much information can contain a simple planar graph of n nodes? The question is highly related to the number of planar graphs. Counting the number of (non-isomorphic) planar graphs with n nodes is a well-known and long-standing unsolved graph-enumeration problem (cf. [24]). There is no known close formula, neither asymptotic nor even an asymptotic on the logarithm of this number. Any asymptotic on the logarithm would give a bound on the number of independent random bits needed to generate a planar graph uniformly at random (but not necessary in polynomial time).

Random combinatorial object generation is an important activity regarding average case complexity analysis of algorithms and testing algorithms on typical instances. Unlike random graphs (the Erdös–Rényi graph Model), still little is known about random planar graphs. Indeed adding an edge in a planar graph highly depends on the location of all previous edges. Random planar maps, i.e., plane embeddings of planar graphs, have been investigated more successfully. Schaeffer [35] and then Banderier et al. [2] have showed how to generate in polynomial time several planar map families, e.g., 3-connected planar maps. Unfortunately, this generating does not give much information about random planar graphs because there are many ways to embed a planar graph into the plane. On the positive side, some families of planar graphs support efficient random generation: trees [1], maximal outerplanar graphs [3, 14], and more recently labeled and unlabeled outerplanar graphs [4].

Besides the combinatorial aspect and random generation, an important attention is given in Computer Science to *efficiently* represent discrete objects. Efficiently means that the representation is succinct, i.e., the storage of these objects uses few bits, and that the time to compute such representation is polynomial in their size. Fast manipulation of the so-encoded objects and easy access to a part of the code are also desirable properties. At least two scopes of applications of high interests are concerned with planar graph representation: Computer Graphics and Networking.

Surface discretization of a 3D object outputs a list of 3D coordinates and a set of adjacency relations. In the case of convex objects, the set of adjacency relations is an unlabeled planar graph. In general, small degree faces are used for surface discretization, with triangle or quad meshes. Then, a compressor is applied on the planar graph. Performances are expressed averaging the number of bits per edge or per node. They are evaluated among a benchmark of standard examples [21], due to the lack of "good" random planar graph generator, or typical instance generator. For example, King and Rossignac [22, 34] gave a triangulation compressor that guarantees 3.67 bits per node, the best possible rate being $\log_2(256/27) \approx 3.24$ bits per node from Tutte's enumerative formula [39].

Routing table design for a network has been investigated in the case of planar networks [15, 16, 26, 37]. The underlying graph of the network is preprocessed to optimize routing tables, a data structure dedicated to each node in charge of finding the next output port given the destination address of an incoming message. The main objective is to minimize the size of the routing tables while maintaining routes as short as possible. The strategy used by Gavoille and Hanusse [16] based on a

k-page embedding, and then improved by Lu [26] with *orderly spanning trees*, demonstrates that a compact planar graph representation helps for the design of compact routing tables, especially when shortest paths are required.

1.1 Related Works

Succinct representation of n-node m-edge planar graphs has a long history. Turán [38] pioneered a $4m$ bit encoding, which has been improved later by Keeler and Westbrook [20] to $3.58m$. Munro and Raman [29] then proposed a $2m + 8n$ bit encoding based on the 4-page embedding of planar graphs (see [40]). In a series of articles, Lu et al. [8,11] refined the coding to $4m/3 + 5n$, thanks to orderly spanning trees, a generalization of Schnyder's trees [36]. Independently, codings have been proposed for triangulations, where $m = 3n-6$. A $4n$ bit encoding has been obtained by several authors [5,11,34], interestingly with rather different techniques, and then improved by the Rossignac's Edgebreaker [22], who guaranteed $3.67n$ bits for triangulations and computable in $O(n)$ time. Actually, He et al. [19] showed that, in $O(n \log n)$ time, a space optimal encoding for triangulations and for unlabeled planar graphs can be achieved. Hence, a $O(n \log n)$ time and a $3.24n$ bit encoder for triangulations exist. For that, they use a recursive separator decomposition of the graph, and an exponential coding algorithm for the very end components of sub-logarithmic size. However, the time complexity hidden in the big-O notation could be of limited use in practice. To implement the encoder, one needs, for instance, to implement planar isomorphism and Lipton–Tarjan planar separator [23]. The time complexity has been recently improved to $O(n)$ for planar graphs by Lu [27]. Although the length of the coding is optimal, the approach of [19,27] does not give any explicit bound of the number of bits used in the representation.

If we are interested only in the information-theoretic bound of planar graphs or in statistical properties of planar graphs (what a random planar graph looks like: number of edges, connectivity, etc.), other tools can be used. Denise et al. [12] specified a Markov chain on the space of all labeled planar graphs whose limit distribution is the uniform distribution. Their experiments show that random planar graphs have approximately $2n$ edges, and are connected but not 2-connected. Although the Markov chain converges to the uniform distribution, it is not proved whether this Markov chain becomes close enough to the uniform distribution after a polynomial number of steps. It is, however, proved that almost all labeled planar graphs have at least $1.5n$ edges, and that the number $p(n)$ of unlabeled planar graphs satisfies that $\frac{1}{n} \log_2 p(n)$ tends to a constant γ such that $\log_2(256/27) \leqslant \gamma \leqslant \log_2(256/27) + 3$. The bounds on γ easily derive from Tutte's formula [39]: Triangulations are planar graphs, and every planar graph is a subgraph of a triangulation, thus having 2^{3n-6} possible subsets of edges. There are also no more than $n!2^{\gamma n+o(n)}$ labeled planar graphs as there are at most $n!$ ways to label the nodes of a graph.

Osthus et al. [31] investigated triangulations containing any planar graph, and they showed that there is no more than $n!2^{5.22n+o(n)}$ labeled planar graphs. Osthus et al. [31] also showed that almost all labeled planar graphs have at most $2.56n$ edges, and that almost all unlabeled planar graphs have at most $2.69n$ edges. A lower bound of $13n/7 \approx 1.85n$ has been obtained by Gerke and McDiarmid [17], improving the $1.5n$ lower bound of the expected number of edges of [12]. Properties of random planar graphs have also been investigated in [28].

Gimenez and Noy [18] show that the number of edges of a random labeled planar graph is asymptotically normal and the mean is $2.213n$ and variance is $0.4303n$. Unlike general graphs, labeled and unlabeled planar graphs do not have the same growing rate (up to the $n!$ term) as proved in [28]. So upper bounds on labeled planar graphs do not transfer to upper bounds on unlabeled planar graphs, but the reverse is true.

Using generating function techniques, Noy and Gimenez [18] proved that the number of labeled connected planar graphs tends to $n!2^{4.767n+O(\log n)}$. The number of simple planar maps is asymptotic to $2^{5.098n+O(\log n)}$ (cf. algebraic generating function presented in [25]) providing an upper bound for unlabeled planar graphs.

1.2 Presented Results

In this chapter, we present a new representation of planar graphs called *well-orderly maps*. Starting from a planar graph, we show how to build and encode a well-orderly map in linear time. Our construction leads to counting results about planar graphs. More precisely, we show an upper bound of $2^{4.91n+o(n)}$ on $p(n)$, the number of unlabeled planar graphs with n nodes.

Since our upper bound can be parameterized with the number of edges, and using the lower bound of [18], we are able to show that almost all unlabeled graphs have at least $1.85n$ edges and at most $2.44n$ edges, setting a new lower bound and improving the $2.69n$ upper bound of [31]. The presented results are a synthesis of results presented in [6, 7].

1.3 Outline of the Chapter

Let us sketch our technique. Since the number of useful combinatorial objects are numerous, we first briefly describe in Fig. 2.1 the different steps toward the compact coding of planar graphs. Our real starting point is a planar map, sometimes called planar embedding. To get a planar map from a planar graph, well-known linear time algorithm can be used (see for instance [9]). Roughly speaking, we first present a very particular embedding of a planar graph called *well-orderly map* and show how to encode it using combinatorial tools like bijective combinatorics and a specific compression technic of binary strings.

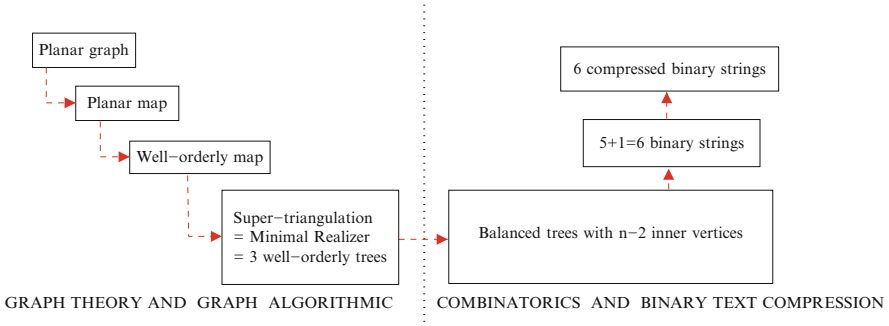

Fig. 2.1 Roadmap toward the compact coding of a planar graph

A First Upper Bound Based on Triangulation

A natural approach to represent an n-node planar graph G is to consider a triangulation of G, i.e., a supergraph S of G such that S is planar, has n nodes and $3n-6$ edges. Then, G can be obtained by coding S and a set M_S of edges such that $E(S) \setminus M_S = E(G)$. This way of representing a planar graph is suggested by the $(\log_2(256/27)+3)n = 6.24n$ bit upper bound of [12] mentioned above.

Introduction of Well-Orderly Maps and Super-Triangulation

To obtain a representation more compact than $6.24n$ bits, we need to carefully construct S. In particular, crucial steps are the way we embed G into the plane, and the way we triangulate its faces. In Sect. 2.2, we introduce a specific embedding of G called a *well-orderly map*, and we show that it can be computed in linear time. Given a well-orderly map, we present how to build the supergraph S of G, called hereafter *super-triangulation* and defined in Sect. 2. More precisely, a super-triangulation S has the property that for a given node $v \in S$ only, one can perform in a unique manner a traversal of S by following a specific spanning tree T rooted in v, called a *well-orderly tree*, such that T is contained in G. Hence, given the super-triangulation S of G, M_S is of cardinality at most $(3n-6)-(n-1) = 2n-5$, and the edges of M_S can be described among the possible edges of $S \setminus T$ only, i.e., with at most $2n$ bits. This already provides a $(\log_2(256/27)+2)n = 5.24n$ bit upper bound. Observe that the case G not connected can be easily transformed (in linear time) into a new connected graph $\tilde{G}$, e.g., by linking all the connected components of G into a single node (see Sect. 4 for more details).

Using Minimal Realizer Properties

The next step consists in encoding in a very compact way the super-triangulation. In Sect. 3.1, we represent the super-triangulation S by a *realizer*, that is a partition

of the edges into three trees (T_0, T_1, T_2) (see Schnyder's trees [36]). In our case, the partition has specific properties and corresponds to *minimal realizer*. We also show how to uniquely recover the three trees from such a super-triangulation. Different properties of minimal realizer are useful since the knowledge of two well-orderly trees implies a canonical description of the third one and can exploited to save bits. At this point, the following properties are only given as an illustration:

- Every edge (u, v) of S such that (1) u is the parent of v in T_1 and (2) u is an inner node in T_2, must belong to G. This significantly saves bits in the coding of M_S since many edges of G can be guessed from S.
- An extra property is that two nodes belonging to the same branch of T_2 have the same parent in T_1 (a branch is a maximal set of related nodes obtained in a clockwise depth-first search of the tree, and such that a node belongs to only one branch at the time, see Sect. 3). This latter property simplifies a lot the representation of S. Knowing T_2, T_1 does not need to be fully represented. Only one relevant edge per branch of T_2 is enough. As any tree of a realizer can be deduced from the two others, the representation of S can be compacted in a very efficient way, storing for instance T_2 and the relevant edges of T_1.

Combining such properties and the optimal coding of realizer using the bijection of Poulalhon and Schaeffer [33] (see also Theorem 2), we get an encoding of super-triangulation presented in [7].

Compact Coding in Binary Strings

Finally, we show in Sect. 3 that the explicit representation of G is done with six binary strings of different density (namely the ratio between the number of "ones" it contains and its length): five for S and one for M_S. We compact each string with a variant of the Pagh's compressor [32]. This allows to reach an optimal entropy coding, i.e., with $\log_2 \binom{n}{k} + o(n)$ bits for an n-bit string of k ones.[2] If we parametrize the number of branches of T_2 (or equivalently its number of leaves), an entropy analysis shows that $4.91n$ bits are enough to represent G.

2 Embedding and Triangulating Algorithms

A *plane embedding of a graph*, or shortly a *plane graph*, is a mapping of each node to a point of the plane and of each edge to the continuous curve joining the two ends of this edge such that edges do not cross except, possibly, on a common extremity. A graph that has a plane embedding is a planar graph.

[2]The original compressor runs in expected linear time. We give in this chapter a simpler guaranteed linear time construction with asymptotically the same performances.

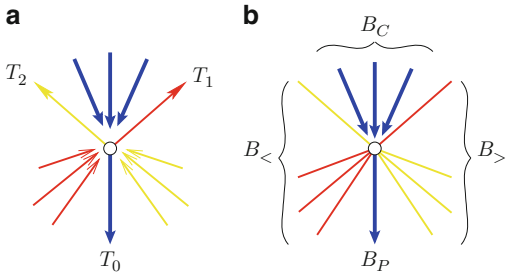

Fig. 2.2 Relationship between realizer and orderly tree: (**a**) edge-orientation rule around a node for a realizer, and (**b**) blocks ordering around an orderly node (T is represented by directed edges because the edge (v, w) of T_1)

In this chapter, we deal with simple (no loops and no multi-edges) and undirected graphs. If we cut the plane along the edges, the remainder falls into connected regions of the plane, called *faces*. Each plane graph has a unique unbounded face, called the *outerface*. The *boundary* of a face is the set of incident edges. The *interior* edges are the edges non-incident to the boundary of the outerface, similarly for interior nodes. Precise definitions can be founded for instance in [13, 30].

A *triangulation* is a plane embedding of a maximal planar graph, that is a planar graph with n nodes and $3n - 6$ edges. There is only one way to embed in the plane (up to a continuous transformation), a maximal planar graph whose three nodes are chosen to lie on the outerface.

2.1 Well-Orderly Tree, Realizer and Super-Triangulation

Let T be a rooted spanning tree of a plane graph H. Two nodes are *unrelated* if neither of them is an ancestor of the other in T. An edge of H is unrelated if its endpoints are unrelated.

We introduce *well-orderly trees*, a special case of *orderly spanning trees* of Chiang, Lin, and Lu in [8], referred as simply orderly trees later. Let $v_1, \ldots, v_n$ be the clockwise preordering of the nodes in T (nodes ordered by their first visit in a clockwise traversal of the tree T). Recall that a node v_i is *orderly* in H with respect to T if the incident edges of v_i in H form the following four blocks (possibly empty set of vertices) in clockwise order around v_i (see Fig. 2.2b):

- $B_P(v_i)$: the edge incident to the parent of v_i
- $B_<(v_i)$: unrelated edges incident to nodes v_j with $j < i$
- $B_C(v_i)$: edges incident to the children of v_i
- $B_>(v_i)$: unrelated edges incident to nodes v_j with $j > i$

A node v_i is *well orderly* in H with respect to T if it is orderly, and if:

- The clockwise first edge $(v_i, v_j) \in B_>(v_i)$, if it exists, verifies that the parent of v_j is an ancestor of v_i (in T).

In other words, if (v_i, v_j) the first edge of $B_>(v_i)$, then the parent of v_j is an ancestor of v_i in T.

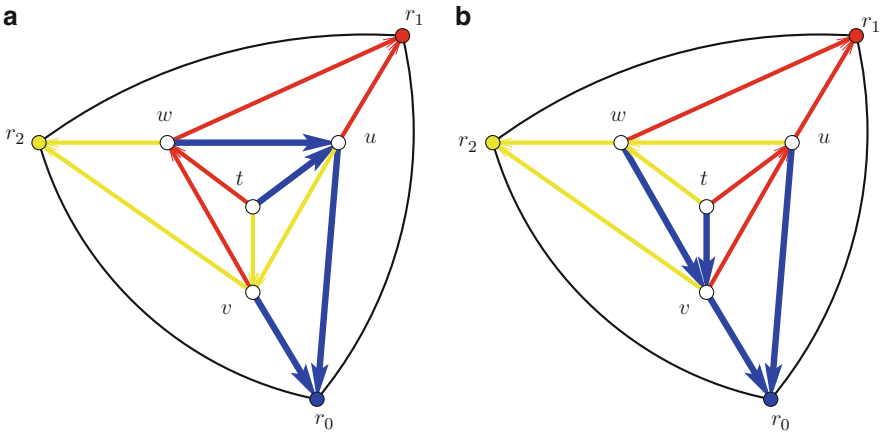

Fig. 2.3 Two realizers for a triangulation. The tree $\overline{T}_0$ rooted in r_0 (the tree with bold edges augmented with the edges (r_0, r_1) and (r_0, r_2)) is well orderly in (**b**), and simply orderly in (**a**) (the node v is not well orderly: (v, w) is the clockwise first edge of $B_>(v)$ and the parent of t is not an ancestor of v). The clockwise preordering of $\overline{T}_0$ in (**a**) is $r_0, r_2, v, u, t, w, r_1$

Definition 1 (well-orderly tree). T is a *well-orderly tree* of H if all the nodes of T are well orderly in H, and if the root of T belongs to the boundary of the outerface of H (similarly for simply orderly tree).

Note that an orderly tree (simply or well orderly) is necessarily a spanning tree. Observe also that the incident edges in H of a node of T are either in T or unrelated. In particular, if an edge of H is related (i.e., one endpoint is a descendant of the other one in T), then it has to belong to T. It follows that all the neighbors in H of the root of T are in T.

Definition 2 (well-orderly map). A plane graph H is a *well-orderly map rooted in* v if H has a well-orderly tree of root v.

A convenient way to manipulate triangulations is to deal with realizers.

Definition 3 (realizer). A *realizer* of a triangulation is a partition of its interior edges in three sets T_0, T_1, T_2 of directed edges such that for each interior node v it holds (see Fig. 2.2a):

- The clockwise order of the edges incident with v is: leaving in T_0, entering in T_1, leaving in T_2, entering in T_0, leaving in T_1, and entering in T_2.
- There is exactly one leaving edge incident with v in T_0, T_1, and T_2.

Observe that if (T_0, T_1, T_2) is a realizer, then (T_1, T_2, T_0) and (T_2, T_0, T_1) are also realizers. Cyclic permutations of a realizer are not in general the only distinct realizers of a given triangulation. Figure 2.3 depicts two realizers for a same

triangulation. Actually, the number of n-node realizers is asymptotically $2^{4n+O(\log n)}$ (cf. [5]), whereas the number of triangulations is only $(256/27)^{n+O(\log n)}$ (cf. [39]).

Schnyder showed in [36] that each set T_i of a realizer induces a tree rooted in one node of the outerface. Moreover, he described a linear time algorithm to compute such trees. Hereafter, if $R = (T_0, T_1, T_2)$ is a realizer, then for notational convenience R also denotes the underlying triangulation.

There are strong relationships between realizers and orderly trees (see Fig. 2.2). In every realizer $R = (T_0, T_1, T_2)$, T_0 (and by cyclic permutation each T_i) is an orderly tree of $R \setminus \{r_1, r_2\}$, where r_i denotes the root of T_i. Indeed, the incident edges with any node v that are not in T_0 (thus that are unrelated with T_0) are either clockwise before the entering edges of T_0 or clockwise after. Conversely, let T be an orderly tree of a triangulation. Observe that the root of T has at least two children (because its root is of degree at least two and all its neighbors must be in T), and thus T has at least two leaves. A realizer (T_0, T_1, T_2) can be obtained from T setting $T_0 = T \setminus \{r_1, r_2\}$, where r_1, r_2 are, respectively, the clockwise last and first leaf of T – actually it is not difficult to see that r_i is the root of T_i – and setting, for all inner nodes v, that the clockwise first edge of $B_>(v)$ and the clockwise last edge of $B_<(v)$ belong to T_1 and T_2, respectively (as illustrated in Fig. 2.2b). Observe that this latter assignment for T_1 and T_2 is the only possible realizer with $T_0 = T \setminus \{r_1, r_2\}$.

For each tree T_i of a realizer, we denote by $\overline{T}_i$ the tree composed of T_i augmented with the two edges of the outerface incident to the root of T_i. A node of a rooted tree is *inner* if it is neither the root nor a leaf. For every non-root node $u \in T_i$, we denote by $p_i(u)$ the parent of u in T_i.

Definition 4 (super-triangulation). A realizer $S = (T_0, T_1, T_2)$ is a *super-triangulation* of a graph G if:

1. $V(S) = V(G)$ and $E(G) \subseteq E(S)$
2. $E(T_0) \subseteq E(G)$
3. $\overline{T}_0$ is a well-orderly tree of S
4. For every inner node v of T_2, $(v, p_1(v)) \in E(G)$

Intuitively, a super-triangulation of a graph G is a specific triangulation of the faces of a specific plane embedding of G. Before exploring more deeply the properties of super-triangulations, observe that, from Definition 4, the tree T_0 does not span in general the graph G (cf. example in Fig. 2.4). Moreover, a non-connected graph may have a super-triangulation. For example, if G has an edge and two isolated nodes, in that case, $E(T_0) = E(G)$ is possible.

Theorem 1. *Every connected planar graph with at least three nodes has a super-triangulation, computable in linear time.*

In particular, Theorem 1 implies that every connected planar graph has an embedding which is a well-orderly map.

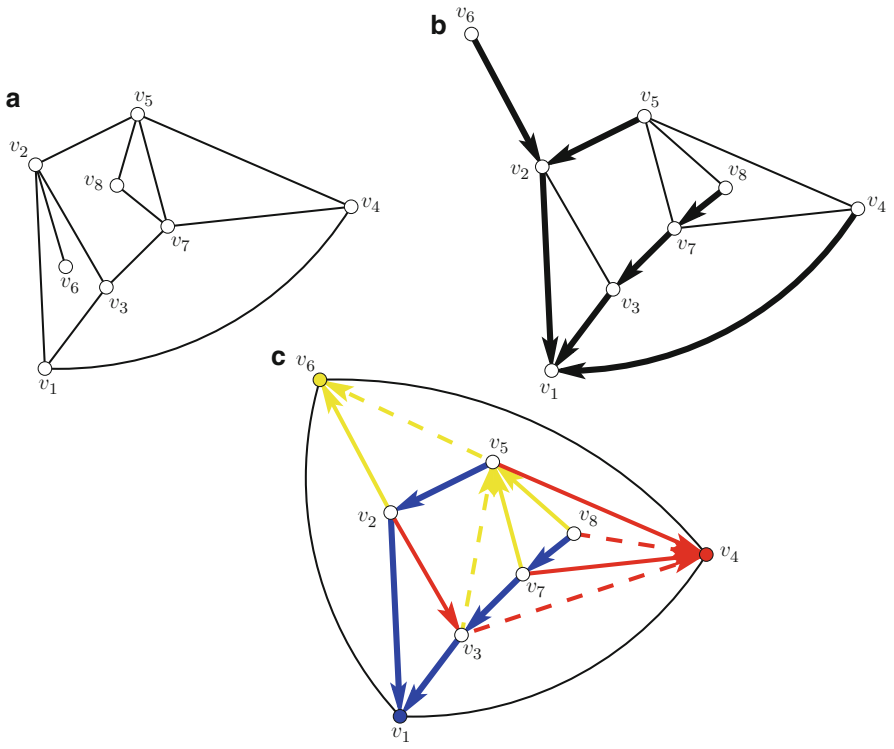

Fig. 2.4 A planar graph G (**a**), a well-orderly map of G rooted at v_1 with its well-orderly tree (*bold edges*) (**b**), and a super-triangulation of G (**c**) (*dotted edges* are non-edges of G)

2.2 Computing a Super-Triangulation from a Well-Orderly Map

In order to prove Theorem 1, we need the next three lemmas. The proofs of these lemmas are given after the proof of Theorem 1.

Lemma 1. *Every well-orderly map rooted in some node v has a unique well-orderly tree of root v.*

Lemma 2. *Let G be a connected planar graph, and let v be any node of G. Then G has a well-orderly map of root v. Moreover, well-orderly trees and the well-orderly map can be computed in linear time.*

In [8], a result similar to Lemma 2 about simply orderly trees and embeddings is proved. However, the extra condition reduces much more the choice of the embedding for the input planar graph and leads to the uniqueness of the tree

(Lemma 1). In the case of simply orderly embeddings, several orderly trees may exist (cf. Fig. 2.3 where both orderly trees $\overline{T}_0$ span the same triangulation). Actually, the uniqueness concerns also the way to triangulate the faces of well-orderly maps, thanks to the next lemma.

Lemma 3. *Let T be the well-orderly tree of H rooted in some node r_0, and assume that T has at least two leaves. Let r_2 and r_1 be the clockwise first and last leaves of T, respectively. Then, there is a unique super-triangulation (T_0, T_1, T_2) of the underlying graph of H, preserving the embedding H, and such that each T_i has root r_i. Moreover, $T_0 = T \setminus \{r_1, r_2\}$ and the super-triangulation are computable in linear time.*

First of all, let us show that Lemmas 1, 2, and 3 imply Theorem 1.

Proof of Theorem 1. Consider a connected planar graph G with at least three nodes, and let v be any node of G with the only constraint that if G is a path, then v is chosen to be of degree two (this is feasible since G has at least three nodes). Thanks to Lemma 2, one can compute in linear time a well-orderly map H of G and a well-orderly tree T rooted in v. Let us show that T has at least two leaves r_1, r_2 lying on the outerface of H, r_2 traversed before r_1 in a clockwise preordering of T.

We show that T cannot be a chain, and thus has a node with at least two children (and thus has two leaves). If G is a path, then T rooted in a node of degree two is not a chain. Assume that G is not a path, but T is a chain. Then there exists an edge of G that is not in T. However, all pairs of nodes of a chain are related, thus must belong to T. Therefore, T is not a chain.

Lemma 3 can be therefore applied, and one can compute for G a super-triangulation in linear time. □

Proof of Lemma 1. Assume that H has two well-orderly trees T, T' rooted in v. Let $v_1, \ldots, v_n$ (resp. $v'_1, \ldots, v'_n$) be the clockwise preordering of the nodes of T (resp. T'). Let v_i be the node such that the neighbors of v_i in T and in T' differ, and such that i is minimum. We have $v_t = v'_t$ for all $t \leqslant i$, and $B_C(v_i) \neq B'_C(v_i)$, where $B'_C(v_i)$ denotes the children edge block around v_i in T'.

W.l.o.g. assume $|B_C(v_i)| \leqslant |B'_C(v_i)|$ (the symmetric case is proved by exchanging the role of T and T'). Note that $B_<(v_i) = B_>(v_i) = \varnothing$ is impossible, otherwise $B_C(v_i)$ would consist of all the neighbors of v_i (maybe the v_i's parent excepted) and $|B_C(v_i)| \leqslant |B'_C(v_i)|$ and $B_C(v_i) \neq B_C(v_i)$ would be incompatible. Let e_1 (resp. e_2) be the clockwise first (resp. last) edge of $B_C(v_i)$. Let e be an arbitrary edge of $B'_C(v_i)$. In the following, $e_1 \leqslant e$ means either $e_1 = e$, or e_1 is clockwise before e around v_i.

Let us show that $e_1 \leqslant e$. This is clearly true if $B_<(v_i) = \varnothing$. If $B_<(v_i) \neq \varnothing$, then consider any edge $(v_i, v_h) \in B_<(v_i)$. Then, $(v_i, v_h) \notin B'_C(v_i)$. Indeed, as $h < i$, the path from v_h to v_i in T exists also in T', and the edge (v_i, v_h) of T' would create a cycle in T'. Thus, $e_1 \leqslant e$.

If $B_>(v_i) = \varnothing$, then $e \leqslant e_2$. Hence, $e_1 \leqslant e \leqslant e_2$ which is incompatible with the fact that $B'_C(v_i)$ and $B_C(v_i)$ are blocks of consecutive edges such that $|B_C(v_i)| \leqslant |B'_C(v_i)|$. Thus, we must have $B_>(v_i) \neq \varnothing$.

Let (v_i, v_j) be the clockwise first edge of $B_>(v_i)$. Then, $(v_i, v_j) \notin B'_C(v_i)$. Indeed, as T is well orderly, the v_j's parent in T, say v_k, is an ancestor of v_i, so $k < i$. As $B_C(v_k) = B'_C(k)$ for $k < i$, the edge (v_k, v_j) exists in T'. Hence, the path from v_j to v_i in T exists also in T', and the edge (v_i, v_j) of T' would create a cycle in T'. It follows that every edge $e \in B'_C(v_i)$ is such that $(v_i, v_j) \leqslant e$ and $e \neq (v_i, v_j)$. As the path from v_i to v_j in T exists also in T', the node v_j is after v_i in a clockwise preorder of T'. It follows that v_i is not well orderly in T': a contradiction. □

Proof of Lemma 2. We first give a simple algorithm to construct a well-orderly map of G. Then we give some hints for an $O(n)$ time implementation.

We start by computing an arbitrary plane embedding H of G such that v belongs to the outerface of H. This can be done in $O(n)$ time [10]. Then we traverse H from v to build a well-orderly tree T. However, not every plane embedding allows the construction of a well-orderly tree. If during the construction, T does not span all the nodes, the embedding of H is modified, and a new traversal is run again. We show that, after a finite number of steps, the number of nodes covered by the traversal increase, and so the construction converges to a well-orderly tree. To describe more precisely the traversal and the modification of H, we need some definitions.

Let T be any tree of H rooted in v, not necessarily a spanning tree. A node is *free* if it does not belong to T. An edge is free if one of its endpoints if free. We extended the notion of well-orderly node as follows: a node is *partially well orderly* (with respect to H and T) if it is well orderly except that its edge-blocks $B_<$ and $B_>$ (relative to a clockwise preordering of the nodes of T) may contain zero or more free edges. Moreover, the clockwise last edge of $B_<$ and the clockwise first edge of $B_>$ are never free. T is partially well orderly if all the nodes of T are partially orderly. Every well-orderly tree is a partially well-orderly tree that spans H. The four edge-blocks around a partially well-orderly node u in T are denoted by $B_P(v, T)$, $B_<(v, T)$, $B_C(v, T)$, and $B_>(v, T)$. For convenience, the clockwise last edge of $B_<(v, T)$ is named the *back-edge* of u in T, and the clockwise first edge of $B_>(v, T)$ is named the *front-edge* in T.

We run a procedure traversal (H, v) applied on the current embedding H of G, which returns a partially well-orderly tree T of H rooted in v (see Fig. 2.5 for example). It initializes $T := \{v\}$ and treats v as follows:

(1) List in a clockwise manner around v the edges $(v, u_1), \ldots, (v, u_k)$ that are after the back-edge and before the front-edge of v in T (if the back-edge or the front-edge does not exist, the list consists of all the edges incident with v not already in T)
(2) Update T with the edges $(v, u_1), \ldots, (v, u_k)$
(3) Recursively treat the nodes $u_1, \ldots, u_k$ in that order

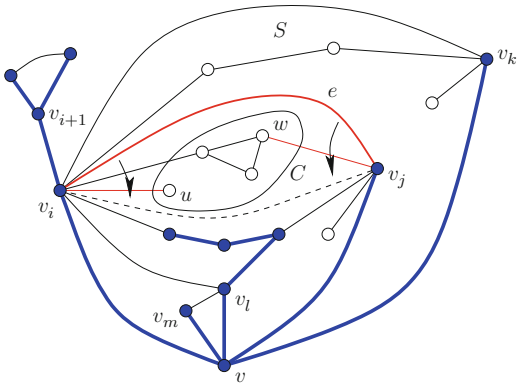

Fig. 2.5 The partial tree constructed by traversal (H, v) is in *bold*. Free nodes are drawn *white*. The *dotted edge* is the new location of e after running flip $(H, e, (v_i, u), (v_j, w))$

Let $v_1, \ldots, v_p$ be the clockwise preordering of the nodes of T, the tree returned by traversal (H, v). Consider a node v_i, and let T_{v_i} be the tree obtained by traversal (H, v) after the treatment of v_i. A key observation is that $B_<(v_i, T_{v_i}) = B_<(v_i, T)$ and that $B_>(v_i, T_{v_i}) = B_>(v_i, T)$. In particular, the back-edge and front-edge of v_i with respect to T and with respect to T_{v_i} (if they exist) are the same. After the treatment of v_i, the edges around v_i in T_{v_i} form the four (possibly empty) edge-blocks $B_P(v_i, T_{v_i})$, $B_<(v_i, T_{v_i})$, $B_C(v_i, T_{v_i})$, and $B_>(v_i, T_{v_i})$. Hence in T, the edge-blocks around v_i are: $B_P(v_i, T)$, $B_<(v_i, T)$, $B_C(v_i, T)$, and $B_>(v_i, T)$. To show that T is partially well orderly, it remains to show that if $(v_i, v_j) \in B_>(v_i, T)$ is the front-edge, then the parent of v_j in T is an ancestor of v_i. When visiting the node v_i in T, the edges of the tree constructed up to v_i (i.e., $T_{v_{i-1}}$) are either between nodes v_t with $t < i$, or are (v_k, v_j) with $k < i$ and $j > i$. Moreover, v_k belongs to the path from v_i to the root of T, v. Thus a front-edge (v_i, v_j) is such that the parent of v_j is an ancestor of v_i. Therefore, T is partially well orderly.

Assume that T does not cover all the nodes (if T is a spanning tree, then we are done). Let v_i be a node of T having an incident free edge. W.l.o.g., we assume that $B_>(v_i, T)$ contains a free edge (cf. Fig. 2.5). The case where v_i has a free edge in $B_<(v_i, T)$ is symmetric. Let $e_i = (v_i, u)$ be the clockwise last free edge of $B_>(v_i, T)$. (Actually one can choose any free edge that clockwise ends a block of free edges in $B_>(v_i, T)$). By definition, $B_>(v_i, T)$ contains at least one unrelated edge (in particular the front-edge). Hence, let $e = (v_i, v_j)$ be the clockwise last unrelated edge of $B_>(v_i, T)$ that is before e_i. Finally, let $e_j = (v_j, w)$ be the clockwise first free edge of v_j before e and such that there is no unrelated edge between e_j and e (so e_j is the first edge of the block of free edges just before e). If such edge does not exist, we set $e_j := e$. In other words, e, e_i, and e_j are chosen such that the edges between e and e_i in v_i, and between e and e_j in v_j form a maximal block of free edges. We change the embedding H by running the procedure flip (H, e, e_i, e_j) that works as follows: In v_i, e is moved and inserted clockwise after e_i, and in v_j, e is moved and inserted clockwise before e_j. For convenience, we say that a flip has been performed around e.

Once H has been updated by flip (H, e, e_i, e_j), we reapply traversal on the new embedding. Procedures traversal and flip are so applied up to get a spanning tree. To complete the correctness, we need to show that flip keeps planarity, and that the partial well-orderly tree obtained after calling traversal converges to a spanning tree.

Let X be the set of free nodes forming the free edges between e and e_i, and between e and e_j. For every $x \in X$, let C_x denote the connected component containing x in the subgraph of H induced by the free nodes. Let $C = \cup_{x \in X} C_x$. To prove that flip(H, e, e_i, e_j) keeps planarity, we show that every path P from $y \in C$ to the root v contains either v_i or v_j. Let R be the bounded connected region of $\mathbb{R}^2 \setminus B$, where B is the cycle composed of the path in T (the tree obtained before calling flip) from v_i to v_j, and closed by the edge (v_i, v_j). Assume that P contains a node $v_k \in T$ with $v_k \in R \cup B$, and $k \notin \{i, j\}$. W.l.o.g., assume that v_k is the first node T from y in P, and let (v_k, z) be the free edge of P. We have $(v_k, z) \in B_<(v_k, T)$, or $(v_k, z) \in B_>(v_k, T)$. Let us assume $(v_k, z) \in B_<(v_k, T)$, the other case is symmetric. As $B_<(v_k, T) \neq \emptyset$, v_k has a back-edge. The back-edge is (v_k, v_i). Indeed, if the back-edge is (v_k, v_t), $t \neq i$, then the cycle composed of the path in T between v_k and v_t, and closed by (v_k, v_t), would disconnect C_z and $\{v_i, v_j\}$, and v_i or v_j has a free neighbor in C_z. Thus the edge (v_i, v_k) exists and is clockwise after (v_i, v_j) since it belongs to R. As (v_i, v_k) disconnects v_j from C_z, there must exist a free edge (v_i, s), for some $s \in C_z$. This edge is clockwise after (v_i, v_k): a contradiction with the definition of (v_i, v_j).

Therefore, the embedding returned by flip (H, e, e_i, e_j) is a plane embedding of G. Observe that whenever e has been moved below C, thanks to flip, the tree returned by a new call to traversal contains T. Indeed, C is connected to $G \setminus C$ only by v_i and v_j. Thus, the move of e cannot create an unrelated edge (v_t, x) with $x \in C$ and $t \notin \{i, j\}$. Assume that after moving e, traversal does not visit any new node. Then, either v_i has a front-edge and a free edge clockwise after its front-edge, or v_j has a back-edge and a free edge clockwise before its back-edge. Indeed, if not, all the neighbors of v_i and of v_j would be related, and the size of T would increase. Assume that v_i has front-edge e' and a free edge. It follows that a next call to flip can be applied on an edge e' clockwise before e. Hence, in at most $\deg(v_i)$ calls to flip, a new free neighbor of v_i (or of v_j) will be visited. Observe that for every directed edge e, there is at most one flip around e. On Fig. 2.5, after running flip around e, and calling traversal, the tree T is augmented with (at least) the edge (v_j, w).

This completes the correctness of the well-orderly map algorithm for G. Let us evaluate its time complexity. There is at most $O(n)$ calls to traversal and to flip (as there is at most one call to flip per directed edges), each one taking $O(n)$ time. Thus, a naive implementation of that algorithm gives a $O(n^2)$ time algorithm.

We first remark that flip can be implemented in $O(1)$ worst-case time, using double pointers for the incident edge list of a node, and using for each edge a pointer for each endpoint toward the edge pointer in the incident list. Moreover, as the tree grows by adding edges, the construction of the whole tree costs $O(n)$ time. The only difficulty is to efficiently manage the edges e, e_i, and e_j for preparing the call to flip.

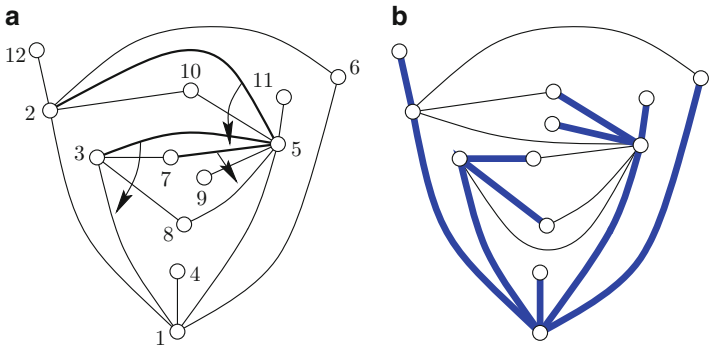

Fig. 2.6 An application of the recursive algorithm with the visit ordering of nodes (**a**), and the final embedding with the well-orderly tree (**b**). Node labeled i is visited at step i, that is the edge of the tree between i and its parent is created at step i. When node 6 is reached, a recursive step is run on the nodes 3, 4, and 5, below the edge $(2, 6)$. In node 3, after a trivial recursive step (treatment of node 4), a flip around $(3, 5)$ is performed. This allows to visit the nodes 7 and 8. At the time node 5 is treated, a flip around $(5, 7)$, and then around $(5, 2)$ is performed, allowing to visit the nodes 9, 10, and 11, completing the recursive step below the edge $(2, 6)$. Then, the last node 12 is visited

Note that when one treats a node v_i with a front-edge (v_i, v_k), we have choice to continue the construction of T (the notations refer to the example depicted on Fig. 2.5): either one can continue T from v_i, by treating v_{i+1}, or we can consider the subgraph S delimited by the cycle composed of the path from v_i to v_k in T and close by (v_i, v_k), and one can recursively apply the treatments of the nodes of S (by treating v_m, v_l, and v_j in Fig. 2.5). If in T_{v_i} the nodes of S have been visited in the order $v_{i_1}, \ldots, v_{i_r}$, then the nodes are recursively treated in the order $v_{i_r}, \ldots, v_{i_1}$. Indeed both parts of the embedding (the part after v_i, and the part inside S) cannot interact because of the edge e. The part of the tree composed of the nodes after v_j can be computed after computing the trees for S and after applying flip (H, e, e_i, e_j). It is not difficult to see that the recursive version of the algorithm allows to manage e, e_i, and e_j with a total of $O(\sum_{i=1}^{n} \deg(v_i)) = O(n)$ time. Figure 2.6 shows an example of the recursive algorithm. □

Proof of Lemma 3. Let T be the well-orderly tree of H rooted in r_0, and let $T_0 = T \setminus \{r_1, r_2\}$, where r_2, r_1 are the clockwise first and last leaves of T.

We first show that r_1 and r_2 belong to the boundary of the outerface of H. Consider P_i the path in T from r_0 to r_i, for $i \in \{1, 2\}$. All the nodes of P_i must belong to the outerface, in particular r_i. Indeed, by induction (this is true for r_0), a node v of P_2 (resp. P_1) has an empty edge-bock $B_<(v)$ (resp. $B_>(v)$). Thus, the clockwise first (resp. last) children of v (if it exists) must belong to the outerface.

Let H' (resp. G') be the plane graph (resp. planar graph) obtained from H (resp. from G) and augmented with the three edges between the r_i's – keeping planarity as r_1, r_2 lie on the outerface – such that they form the boundary of the outerface of H'. (Each edge is added only if it does not create multi-edges.) As the edges

between the r_i's belong to any super-triangulation of G, it remains to show that the super-triangulation $S = (T_0, T_1, T_2)$ for G' preserving H' and with r_i root of T_i, is unique. The super-triangulation for G must be same, S (these three edges cannot create inner node for T_2). We will first show how to construct S and then show that it is unique.

First observe that T_0 is a tree rooted in r_0 (removing two leaves from T maintains its connectivity). Clearly, $E(T_0) \subseteq E(G) \subseteq E(G')$, and $\overline{T}_0$ is a well-orderly tree of H' rooted in r_0. By Lemma 1, $\overline{T}_0$ is unique.

Let us construct the set T_1. Let $v_i \notin \{r_0, r_1\}$ be a node of $\overline{T}_0$. We apply the following assignment rules (for the induction, we assume that the assignment has been applied to $r_0 = v_1, \ldots, v_{i-1}$, and that $p_1(r_0) = r_1$):

1. The clockwise first edge of $B_>(v_i)$ (this edge-block is relative to $\overline{T}_0$), if it exists, belongs to T_1.
2. If $B_>(v_i) = \varnothing$, then the edge leaving v_i to the child of $p_0(v_i)$ immediately after v_i clockwise around $p_0(v_i)$, if it exists, belongs to T_1.
3. If $B_>(v_i) = \varnothing$, and if v_i is the clockwise last child of $p_0(v_i)$, then $(v_i, p_1(p_0(v_i)))$ belongs to T_1.

Let us check that $H' \cup T_1$ is still a plane graph. No edge is added if Rule 1 applies. For Rule 2, there are no edges incident to $p_0(v_i)$ between v_i and its next sibling, keeping planarity of the embedding. And for Rule 3, $p_0(v_i)$ has no children between v_i and $p_1(v_i)$, and allows to freely connect v_i with $p_1(v_i)$ since, by induction, one can assume that the property holds for every node v_j, $j < i$.

Let us check that $\{T_0, T_1\}$ are two sets of a realizer. We have seen that $\overline{T}_0$ is well orderly. Every non-root node v_i has assigned a parent in T_1. Hence, T_1 is connected. We check that in each of the three rules, the parent of v_i in T_1 is assigned to a node v_j of $\overline{T}_0$ with $j > i$. As a consequence, T_1 has no cycle and is connected, so it is a tree. We also check that the edge between v_i and its parent in T_1 is clockwise after the children (if they exist) of v_i in T_0, and clockwise before the edge to $p_0(v_i)$. Hence, such T_1 set is compatible with the edge-orientation rule of realizers.

At this step $H' \cup T_1$ may contain edges that are assigned neither to $\overline{T}_0$ nor to $\overline{T}_1$. Let $X = E(H') \setminus (E(\overline{T}_0) \cup E(\overline{T}_1))$ be this edge set. Constructing T_2 can be done using Property 6. As there is only one way to select T_2 from $\{T_0, T_1\}$, we need to check that the edges of X are compatible with such set T_2. Let e be an arbitrary edge of X. Assume $e = (v_i, v_j)$ with $i < j$. Since $e \notin T_0$, then $e \in B_>(v_i)$. Moreover, $e \notin T_1$ implies that e is not the clockwise first edge of $B_>(v_i)$. Indeed, the clockwise first edge of $B_>(v_i)$ has been assigned to T_1 by Rule 1. Therefore, $e \in T_2$ fulfills the edge-orientation rule of realizers.

It follows that $S = (T_0, T_1, T_2)$ is a realizer of the plane graph $H' \cup T_1 \cup T_2$. We have seen that $E(T_0) \subseteq E(G) \subseteq E(G')$ and that $\overline{T}_0$ is well orderly in H'. By the assignment rules, we remark that if the edge $(v, p1(v)) \notin E(G')$ (Rule 2 or 3), then v cannot have any child in T_2 (the edge $(v, p_1(v))$ forms a triangle with some T_0 and T_1 edges). In other words, for every inner node v of T_2, $(v, p_1(v)) \in E(G')$ (actually $(v, p_1(v)) \in E(G)$). Thus, S is a super-triangulation for G'.

It remains to show that S is the unique super-triangulation of G' that preserve H' and with r_i root of T_i. As T_0 is unique, and as, given $\{T_0, T_1\}$, T_2 is unique, it remains to prove that T_1 is unique.

The clockwise first edge of $B_>(v_i)$ must belong to T_1 since the parent of v_i in T_1 must be clockwise before the edges of T_2 entering in v_i. If $B_>(v_i) = \varnothing$, and if v_i has a sibling v_j immediately clockwise after v_i around $p_0(v_i)$, then (v_i, v_j) must be in T_1. Otherwise, (v_i, v_j) would be in T_2 and $v_i = p_2(v_j)$ would create an inner node of T_2 in v_i. However, as $B_>(v_i) = \varnothing$, $(v_i, p_1(v_i)) \notin E(G')$, contradicting the super-triangulation definition. Finally, if $B_>(v_i) = \varnothing$, and if v_i is the clockwise last child of $p_0(v_i)$, then $(v_i, p_1(p_0(v_i)))$ must be in T_1. Otherwise, $(v_i, p_1(p_0(v_i)))$ would be in T_2 and $v_i = p_2(p_1(p_0(v_i)))$ would create an inner node of T_2 in v_i. This contradicts the super-triangulation definition since, as $B_>(v_i) = \varnothing$, $(v_i, p_1(v_i)) \notin E(G')$.

Therefore, T_1 is unique, and S is unique, completing the proof. □

3 Encoding a Planar Graph with a Super-Triangulation

We start this section with some useful properties for coding super-triangulations.

3.1 Properties of Super-Triangulations

Straightforward from the equivalence between realizers and orderly trees of triangulations, we have the following basic property:

Lemma 4. *Let (T_0, T_1, T_2) be any realizer, and let $v_1, \ldots, v_n$ be the clockwise preordering of the nodes of $\overline{T}_0$. For every v_j with $v_i = p_2(v_j)$ and $v_k = p_1(v_j)$, then $i < j < k$ and neither v_i nor v_k is related to v_j in $\overline{T}_0$.*

A *cw-triangle* is a triple of nodes (u, v, w) of a realizer such that $p_2(u) = v$, $p_1(v) = w$, and $p_0(w) = u$. In the realizer depicted in Fig. 2.3a, (u, v, w) forms a cw-triangle, whereas the realizer of Fig. 2.3b has no cw-triangle.

Let $v_1, \ldots, v_n$ be the clockwise preordering of the nodes of a tree T. The subsequence $v_i, \ldots, v_j$ is a *branch* of T if it is a chain (i.e., v_t is the parent of v_{t+1} for every $i \leqslant t < j$), and if $j - i$ is maximal. Observe that v_{t+1} is necessarily the first child of v_t because of the ordering of the vertices. Branches partition the nodes of T, and there is exactly one branch per leaf.

The tree $\overline{T}_0$ of a realizer (T_0, T_1, T_2) has the *branch property* if for all nodes v_j and $v_i = p_0(v_j)$, either $p_2(v_j) = p_2(v_i)$, or $v_k = p_2(v_j)$ with $i < k < j$ (i.e., $p_2(v_j)$ is a descendant of v_i clockwise before v_j in $\overline{T}_0$). An important feature of the branch property is that all the nodes of a given branch of T_0 (maybe except the root

of T_0) must have the same parent in T_2. Indeed, v_j and $v_i = p_0(v_j)$ belong to the same branch implies that $j = i + 1$, and thus, because there is no index k such that $i < k < j$, $p_2(v_j) = p_2(v_i)$ must hold.

Definition 5 (minimal realizer). A minimal realizer is a realizer having the cw-triangle property.

Lemma 5. *Let $S = (T_0, T_1, T_2)$ be any realizer. The following statements are equivalent:*

1. *S is a super-triangulation for some graph G.*
2. *S has no cw-triangle.*
3. *$\overline{T}_i$ is well orderly in S, for every $i \in \{0, 1, 2\}$.*
4. *$\overline{T}_i$ has the branch property in S, for every $i \in \{0, 1, 2\}$.*

Proof. Let $3'$ (resp. $4'$) denote the property 3 (resp. 4) for $\overline{T}_0$ only. To prove $1 \Leftrightarrow 2 \Leftrightarrow 3 \Leftrightarrow 4$, one can restrict our attention to $1 \Leftrightarrow 2 \Leftrightarrow 3' \Leftrightarrow 4'$ since "S has/has no cw-triangle" is a property stable by cyclic permutation of the trees. Thus, if $2 \Leftrightarrow 3'$, then $2 \Leftrightarrow 3$, and similarly for $4'$.

By definition of a super-triangulation, $1 \Rightarrow 3'$. Moreover, if S is a realizer with $\overline{T}_0$ well orderly, then the realizer S is a super-triangulation of the underlying triangulation S (the four conditions trivially hold). Hence, $1 \Leftrightarrow 3'$.

To prove that $3' \Rightarrow 2$, assume that S has a cw-triangle (u, v, w), and that $\overline{T}_0$ is well orderly in S. As $w = p_1(v)$, then u is an ancestor of w in $\overline{T}_0$. It follows that $v = p_2(u)$ is a descendant of u in $\overline{T}_0$, contradicting Property 4. Hence, $3' \Rightarrow 2$.

To prove that $4' \Rightarrow 2$, assume that S has a cw-triangle (u, v, w), and that $\overline{T}_0$ has the branch property. $u = p_0(w)$ and thus either $p_2(u) = p_2(w)$ or $p_2(w)$ is a descendant of u contained inside the region bounded by the cw-triangle (u, v, w). Clearly, $p_2(u) = p_2(w)$ is impossible, and by the edge-ordering rule of realizers $p_2(w)$ is outside the region bounded by the cw-triangle (u, v, w). Hence, $4' \Rightarrow 2$.

Let $v_1, \ldots, v_n$ be the clockwise preordering of the nodes of $\overline{T}_0$. To prove that $4' \Rightarrow 3$, assume that $\overline{T}_0$ is orderly, but not well orderly in S. Thus, there is an edge (v_p, v_j) with $v_j = p_1(v_p)$ such that the nearest common ancestor between v_p and v_j in $\overline{T}_0$, say v_t, is not $v_i = p_0(v_j)$. Consider the cycle C formed by the path in $\overline{T}_0$ between v_p and v_j, and closed by the edge of (v_p, v_j). Let B be the bounded connected region of $\mathbb{R}^2 \setminus C$. Let us calculate $p_2(v_i)$ and $p_2(v_j)$. By the edge-ordering rule of realizers: (1) $p_2(v_j) \notin B$; (2) $p_2(v_j)$ cannot belong to the path from v_p to v_t in $\overline{T}_0$ with the edge $(v_j, p_2(v_j))$ lying outside B; and (3) $p_2(v_j)$ cannot belong to the path from v_j to v_t in $\overline{T}_0$ since $p_2(v_j)$ is never a descendant of v_j in $\overline{T}_0$ (Property 4). Thus, $p_2(v_j) \notin B \cup C$. Again, from the edge-ordering rule of realizers, $p_2(v_i)$ must belong to $B \cup C$. It follows that $p_2(v_i) = p_2(v_j)$ or $v_k = p_2(v_j)$ descendant of v_i such that $i < k < j$ is impossible, contradicting the branch property. Hence, $4' \Rightarrow 3$.

It remains to show that $2 \Rightarrow 4'$. We assume that S has no cw-triangle. Let v_j be any node, and let $v_i = p_0(v_j)$ and let $v_l = p_2(v_i)$. We assume that $k < l$ ($\overline{T}_0$ has not the branch property), and we will show a contradiction.

Let $v_h = p_0(v_l)$, and let P be the path in $\overline{T}_0$ between v_h and v_k. Note that $v_l \notin P$ because v_l is not an ancestor of v_k ($k < l$). Consider C the cycle obtained by traveling v_i, v_l, v_h, P, v_k, v_j, v_i, and let B be the bounded connected region of $\mathbb{R}^2 \setminus C$. Let Q_1 be the path in $\overline{T}_1$ from v_l to r_1, the root of $\overline{T}_1$. By the edge-ordering rule of realizers, the first edge of Q_1 must belong to $B \cup C$. Because $r_1 \notin B$, Q_1 must intersect C. By Property 4, the intersection must be in some v_t with $t > l$. This intersection cannot be in v_i from the edge-ordering rule of realizers. Since the nodes of P are v_t's with $t \leqslant \max\{h, k\} < l$, we have also that $P \cap Q_1 = \varnothing$. It follows that Q_1 intersect C in v_j.

Let C_1 be the cycle composed of Q_1, v_j, v_i, v_l, and let B_1 be the bounded connected region of $\mathbb{R}^2 \setminus C_1$. Let us show that $B_1 \cup C_1$ contains a cw-triangle. Either C_1 is a cw-triangle and we are done with a contradiction, or consider the path Q_2 in $\overline{T}_0$ from $p_1(v_l)$ to r_0 (the root of $\overline{T}_0$). Similarly, Q_2 must intersect C_1 in v_i. The cycle C_2 traveling v_i, v_l, $p_1(v_l)$, Q_2 defines a connected region B_2 of $\mathbb{R}^2 \setminus C_2$ with a number of faces at least one lower than the number of faces of B_1. Proceeding as previously, we can construct either a cw-triangle or a smaller region with the same inductive property. As the number of faces is finite, it follows that S has a cw-triangle: a contradiction completing the proof of $2 \Rightarrow 4'$, and thus of Lemma 5. □

Lemma 6. *A tree of a realizer is uniquely determined by given the two others. Moreover, given the embedding of the two trees, it takes a linear time to construct the third one.*

Proof. By cyclic permutation of the trees, we only have to prove that the pair $\{T_0, T_1\}$ uniquely determines T_2. Consider the plane graph H composed of $T_0 \cup T_1$. To construct T_2, one needs to triangulate all the faces of H, except the outerface. Let F be any face of H, and let B be its boundary. We assume that F is not the outerface, and that $T_2 \cap F$ contains at least one edge, i.e., that B has at least four nodes.

Once F has been triangulated, B contains only three types of nodes depending on whether the directed edges of $T_2 \cap F$ are leaving, entering, or whether no edge of $T_2 \cap F$ are incident with them. Indeed, a node of B having leaving and entering edges of $T_2 \cap F$ would contradict the edge-ordering rule of realizers. Let us call a *sink* a node of B having at least one entering edge of $T_2 \cap F$. Observe that B contains exactly one sink: zero sink is clearly impossible ($T_2 \cap F \neq \varnothing$), and two or more sinks would provide a node with leaving and entering edges of $T_2 \cap F$, contradicting the edge-ordering rule of realizers. Clearly, once the sink has been located on B, there is only one way to triangulate F. It remains to show that there is only one place on B for a sink.

Traveling B clockwise around F, the two adjacent edges of B at every node v form some cases named $X_i Y_j$, where $X, Y \in \{E, L\}$ and $i, j \in \{0, 1\}$, with the following interpretation: v is in the Case $E_0 L_1$ if the edge clockwise before v in B is entering (E) in v and belongs to T_0, and if the edge clockwise after v in B is leaving (L) v and belongs to T_1. The other cases are defined similarly.

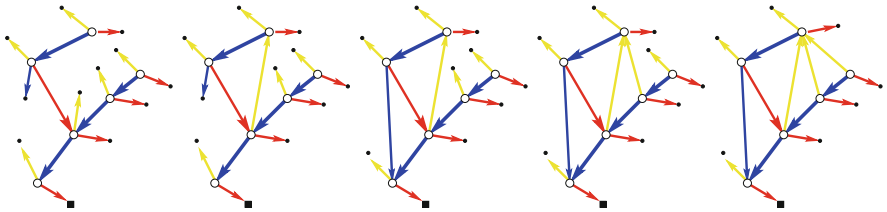

Fig. 2.7 On the *left*, a tree of $\mathcal{B}_n$ (the root is indicated by a *square*). Then from *left* to *right*, the partial closure of the tree

Let us show that v is a sink if and only if v is in the Case L_0L_1. If v is a sink, then two adjacent edges of B with v must be leaving, and L_0L_1 is the only possibility satisfying the edge-orientation rule of realizers. Now, assume that v occurs in the Case L_0L_1. Then, $T_2 \cap F$ has no edge leaving v. If v has no entering and leaving edges in $T_2 \cap F$, then a node $u \in B$ adjacent to v must be a sink. The node u is therefore in the Case E_1X_i (if u is clockwise after v in B), or in the Case Y_jE_0. However, we have already seen that a sink must be in the Case L_0L_1: a contradiction. Thus, v is a sink.

It follows that the place of the unique sink on B is entirely determined by the edge pattern of B induced by the pair $\{T_0, T_1\}$. Clearly, triangulate all the faces (by determining the sinks) takes linear time, completing the proof. □

In this section, we briefly recall a result from [33] about minimal realizers and plane trees. An encoding of well-orderly maps follows.

3.2 Minimal Realizers and Plane Trees

The key point of this section is the definition of the operation of *closure* of planted tree to get a realizer (Fig. 2.7). A tree is *p*lanted if it is rooted on a leaf, that is a leaf is distinguished. Let $\mathcal{B}_n$ be the set of planted plane trees with n inner nodes and $2n$ leaves such that each node is adjacent to 2 leaves. Given a tree T in $\mathcal{B}_n$, we can easily say that:

- Its *canonical orientation* shall be toward the root for all inner edges and toward the leaf for all dangling edges.
- Its *canonical coloring* described by the rule of Fig. 2.2a.

A triple (e_1, e_2, e_3) of edges of a map M is an *admissible triple* if and only if: i/ $e_1 = (v_0, v_1)$, $e_2 = (v_1, v_2)$, and $e_3 = (v_2, v_3)$ appear consecutively in the clockwise direction around the outer face and ii/ if v_3 is a vertex of degree 1.

Definition 6 (local closure). The *local closure* of M with respect to an admissible triple (e_1, e_2, e_3) is obtained by merging the leaf v_3 on node v_0 so as to create triangular face.

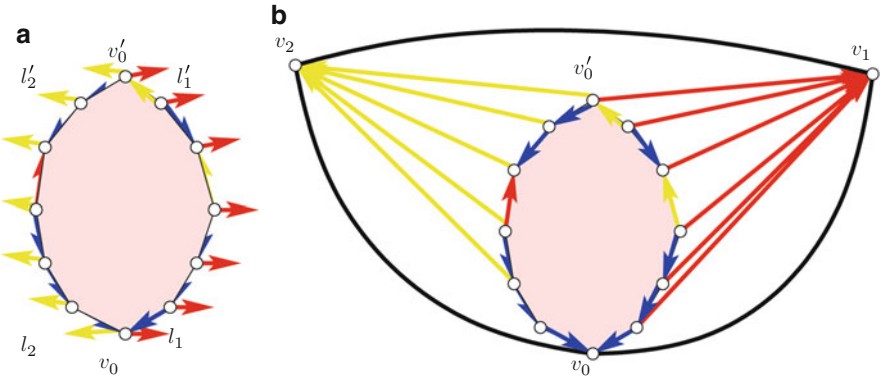

Fig. 2.8 The structure after a partial closure and the complete closure

The *partial closure* of a tree T of $\mathcal{B}_n$ is the map obtained by performing iteratively the local closure of any available admissible triple in a greedy way. As shown in [33], the local closure is well defined independently of the order of local closures. Moreover, all the bounded faces of the resulting map are triangular, and the outer face has the structure shown in Fig. 2.8a. In particular, in the partial closure of T, there are exactly two *canonical nodes* separating the dangling edges in the outer face into two parts. Each of these parts contains dangling edges of same color. A tree T is *balanced* if its root is one of the two canonical leaves. Finally, the *complete closure* of a balanced tree T is the map obtained from the partial closure of T by merging each remaining noncanonical leaf of each part into a root edge, as illustrated in Fig. 2.8b.

Theorem 2 ([33]). Complete closure *is a one-to-one correspondence between* $\mathcal{B}_{n-2}$ *and triangulations with n nodes.*

Observe that by construction the orientation of the dangling edge prevents the formation of cw-triangles implying that using Complete Closure, we get a minimal realizer.

Corollary 1. Complete closure *is a one-to-one correspondence between balanced trees with* $n-2$ *and* minimal realizers *of triangulation with n nodes.*

The following new lemma will serve to predict the entering edges created by complete closure at a node.

Lemma 7. *Let v be an inner node of a balanced tree B. Let* $e_1 = (v, u)$ *and* $e_2 = (v, w)$ *be two consecutive edges around v in clockwise order. During the closure algorithm, no edges will be inserted between* e_1 *and* e_2 *if and only if:*

(a) w is a leaf of B or
(b) w is an inner node of B and the node t such that the edge $e_3 = (w, t)$ *is the next edge around w after* e_2 *in clockwise order is a leaf of B.*

Proof. Let v an inner node of a balanced tree B. Let us consider two consecutive edges (v, u), (v, w) around v in clockwise order. If w is a leaf, then during the closure it will merge with a node w' and close a triangular face enclosing the corner between (v, u) and (v, w). No other edge can thus arrive at this corner. Assume now that w is an inner node of B. Let (w, t) be the next edge around w in clockwise order. If t is a leaf of B, then it will merge with u to form a triangular face, and again no edge can arrive in the corner between (v, u) and (v, w). In the other cases, (v, w) is an inner edge followed by another inner edge (w, t). Since an edge that forming a triangular face that encloses the corner between (v, u) and (v, w) must be from w, the corner is not enclosed. But at the end of the partial closure, there are no more pairs of consecutive inner edges: some edge must have arrived in the corner. □

Lemma 8. *Let (T_0, T_1, T_2) be the minimal realizer encoded by a balanced tree B. A node v of B is a leaf of T_2 if and only if v has no incoming edge colored 2 in B and,*

1. *The parent edge of v in B is colored 2 or*
2. *The parent edge of v in B is colored 1 or*
3. *The parent edge of v in B is colored 0 and v is the last child with an edge colored 0 in clockwise order around $P_B(v)$ and*
   a) *The parent edge of $P_B(v)$ is colored 0 or*
   b) *The parent edge of $P_B(v)$ is colored 2.*

Proof. For the node v to be a leaf in T_2, it must have no incoming edge of color 2 in B, and no edge must be inserted between its outgoing edges of color 0 and 1. When the parent edge of v has color 2 or 1, the outgoing edge of color 0 connects to a leaf and Case (a) of the previous lemma ensures that no edge arrives between this outgoing edge of color 0 and the outgoing edge of color 1. When the parent edge of v has color 0, if the next edge in clockwise order around the parent $P_B(v)$ of v in B is an outgoing edge (of color 1), then Case (b) of the previous lemma ensures that no edge of color 2 arrives.

Finally, we need to check in the remaining cases that an incoming edge of color 2 indeed arrives between the two outgoing edges of color 0 and 1. This could happen if the corner we consider was part of the unbounded face after the partial closure. But in the remaining cases, both the edge $(v, P_B(v))$ and the next edge in clockwise order around $P_B(v)$ are incoming. Since the form of the boundary after partial closure prohibits two consecutive incoming edges, the proof of the lemma is complete. □

3.3 Representation of Planar Graphs with Binary Strings

Along this section, we consider $S = (T_0, T_1, T_2)$ be any super-triangulation of G, a connected planar graph with n nodes and m edges. We show how to use S to

efficiently represent G. Let $\ell(B)$ be the number of vertices of B corresponding to the leaves of T_2, where $S = (T_0, T_1, T_2)$ is obtained by the complete closure of B.

Theorem 3. *Any well-orderly map with n nodes can be coded by a pair (B, W) where B is a balanced tree of $\mathcal{B}_{n-2}$ and W a bit string of length $n + \ell(B)$. Encoding and decoding takes linear time.*

The following lemmas describe in detail the key points of Theorem 3.

Lemma 9. *Let B be a balanced tree such that the corresponding super-triangulation $S = (T_0, T_1, T_2)$ has i_2 inner nodes in the tree T_2. The balanced tree B can be encoded with five binary strings S_1, S_2, S_3, S_4, and S_5 and four integers $a_0, a'_0, a_1, i_2 \leqslant n$ such that:*

$\#S_1 = \binom{n-a_0}{i_2-a_0}$, $\#S_2 = \binom{n-a_1}{a'_0}$, $\#S_3 = \binom{n+a_1}{a_1}$, $\#S_4 = \binom{a_1+a_0+a'_0}{a_0}$ *and* $\#S_5 = \binom{n-a_1-a'_0}{n-a_1-a'_0-i_2}$.

Proof. Let B be a colored balanced tree. We partition the nodes of B in the following way:

- A_1: the set of nodes v such that the edge $(v, P_B(v))$ is colored 1.
- A_2: the set of nodes v such that the edge $(v, P_B(v))$ is colored 2.
- A'_0: the set of nodes v and such that the edge $(P_B(v), P_B(P_B(v))$ is colored either 0 or 2, and such that v is the last child in clockwise order with the edge $(v, P_B(v))$ is colored 0.
- A_0: the set of nodes that are not in the previous sets.

Note that the root of B is in A_0 and for every node v of A_0, the edge $(v, P_B(v))$ is colored 0. Assume that we are coding the balanced tree B. The only information we need, for each node in the prefix clockwise order, is its number of children in A_0, A'_0, A_1, and A_2. In order to encode efficiently a well-orderly map, we need to introduce another parameter in our encoding. Let I_2 be the set of nodes of B that will be inner nodes in the tree T_2 of the corresponding realizer $S = (T_0, T_1, T_2)$.

We give some preliminary remarks:

1. Nodes of A_1 cannot have children in A'_0.
2. Every node of $A_0 \cup A'_0 \cup A_2$ has at most one child in A'_0.
3. $A_0 \subseteq I_2$ (see Lemma 8).
4. Every node of $A'_0 \cup A_1 \cup A_2$ which is also in I_2 has at least one child in A_2 (see Lemma 8).
5. Every node of $V \setminus A_1$ can have children in A_0 only if it has a child in A'_0.
6. Only nodes of I_2 can have children in T_2.

To encode the balanced tree, we will build five binary strings. With these strings we will determine, for each node, its number of children in each subset.

The first string, S_1, tells which node belongs to I_2. Since all the nodes of A_0 are in I_2 (see Remark 3), S_1 stores the information for all the other nodes. Hence for each node of $V \setminus A_0$, the corresponding bit is set to 1 if the node belongs to I_2 and is set to 0 otherwise. Hence, the string S_1 contains $n - a_0$ bits and $i_2 - a_0$ 1's.

The second string S_2 is used to determine whether a node has a child in A'_0. Since all the nodes of A_1 have a child in A'_0 (see Remark 1), S_2 stores this information for all the other nodes: the corresponding bit is set to 1 if the node has one child in A'_0 and to 0 otherwise. Hence, the string S_2 contains $n - a_1$ bits and a'_0 1's.

The string S_3 stores, for each node, its number of children in A_1 in a "Lukasiewicz" way. For each v node of B in the prefix clockwise order, we append to S_3 as many 1's as the number of children of v in A_1 and then we insert a 0. Hence, the string S_3 contains $n + a_1$ bits and a_1 1's.

The string S_4 stores the number of children in A_0. This information has to be stored for each node of A_1 and for each node that has a child in A'_0 (see Remark 5). Hence for each of these nodes, we proceed as for the string S_3. Hence, the string S_4 contains $a_1 + a'_0 + a_0$ bits and a_0 1's.

The string S_5 helps to determine the number of children in A_2. We only need to store this information for the nodes of I_2 (see Remark 6). Moreover, for these nodes that are in $A_0 \cup A'_0 \cup A_2$, we already know that they have at least one child in A_2; so we only need to count the other 1's. Hence for each of these nodes, we proceed as for the strings S_3 and S_4. We obtain a string $i_2 + (a_2 - (i_2 - a_0)) = n - a_1 - a'_0$ bits with $a_2 - (i_2 - a_0) = n - a_1 - a'_0 - i_2$ 1's. □

Lemma 10. *Let H be a well-orderly map with n nodes and m edges. H can be encoded with six binary strings (five for the minimal realizer and a last one to store the missing edges) and four integers $a_0, a_1, a'_0, i_2 \in [0, n]$ such that:* $\#S_1 = \binom{n-a_0}{i_2-a_0}$, $\#S_2 = \binom{n-a_1}{a'_0}$, $\#S_3 = \binom{n+a_1}{a_1}$, $\#S_4 = \binom{a_1+a_0+a'_0}{a_0}$, $\#S_5 = \binom{n-a_1-a'_0}{n-a_1-a'_0-i_2}$, $\#S_6 = \binom{2n-i2}{m-n-i_2}$.

Proof. With $S_1 - S_5$ a minimal realizer is encoded (Lemma 9). The last string indicates the edges to delete to rebuild the well-orderly map: for each v, one bit is used to indicate if the edge $(v, p_2(v))$ has to be removed, and for each leaf v of T_2, one bit is used to indicate if the edge $(v, p_1(v))$ has to be removed. □

We present here a variant of the Pagh's compressor [32]. We denote by $\#S$ the number of binary strings having the same length and the same number of ones than S.

Lemma 11. *Every binary string S of length n can be coded into a binary string of length $\log_2(\#S) + O(n \log\log n / \log n)$. Moreover, knowing n, coding and decoding S can be done in linear time, assuming a* RAM *model of computation on words of $\omega \geqslant \log_2 n$ bits.*

Proof. The main idea is to split S into blocks of equal size b, and to code each block optimally. Each block encoding takes a time exponential in b. However, the code of all possible blocks can be tabulated once in time $O(2^b) = O(n)$, for suitable b small enough. Optimality of the coding derives from optimality of each block by super-additivity of binomials. More precisely, we proceed as follows.

Let $b = \lfloor \log_2 n - \log_2 \log_2 n \rfloor$. Note that $2^b \leqslant n/\log_2 n$. Standard arithmetic operations on integers in the range $[0, 2^b)$ can be done in constant time since $\omega \geqslant b$, by loading each integer into a word of length ω and padded with left extra zeros. Let k_p denote $\binom{b}{p}$. We need the construction of some tables.

We first construct a table L such that for every $p \in [0, b]$, $L[p] = \lceil \log_2 k_p \rceil$. All the numbers $k_0, \ldots, k_b$ are computed, thanks to the Pascal's method, namely using iteratively the formula $\binom{i}{j} = \binom{i-1}{j-1} + \binom{i-1}{j}$ for all $i \leqslant b$ and $j \leqslant \min\{i, p\}$. This uses $O(b^2)$ b-bit numbers. In total, the construction of L is done in $O(b^2 + \sum_p \log k_p) = O(\log^2 n)$ time, as it costs no more than $O(\log k_p) = O(b)$ time to compute $\lceil \log k_p \rceil$, that is the position of the leading bit of k_p in its binary representation.

We construct a table P of integers in the range $[0, b]$ such that for every $i \in [0, 2^b)$, $P[i]$ is the number of ones in the binary representation of i. The table P can be constructed in time and space $O(b2^b) = O(n)$. However, the time can be reduced to $O(b2^{b/2}) = O(\sqrt{n \log n})$ (and even smaller) using a table P' for half-words of $\lceil b/2 \rceil$ bits. Indeed, we have $P[i] = P'[i/2^{\lfloor b/2 \rfloor}] + P'[i \bmod 2^{\lfloor b/2 \rfloor}]$.

For each $p \in \{0, \ldots, b\}$, we compute a table D_p (used for decoding) such that, for every $i \in [0, k_p)$, $D_p[i]$ is a distinct binary string of length b having p ones. Strings of D_p are lexicographically ordered. Generating all D_p's costs $O(2^b) = O(n/\log n)$ time and $O(b2^b) = O(n)$ space by running all binary strings $s \in [0, 2^b)$ by increasing value, and filling the right entry $D_{P[s]}[i_p]$ (and updating the current index i_p).

Finally, we construct a table C (used for coding) such that for every $s \in [0, 2^b)$, $C[s]$ denotes the index i such that $D_p[i] = s$, where $p = P[s]$. The index $i = C[s]$ is stored on b bits, although only the $L[p] = \lceil \log_2 k_p \rceil$ least significant bits of i are useful since $i \in [0, k_p)$. To construct C, we iterate for all $p \in [0, b]$ and all $i \in [0, k_p)$: $C[D_p[i]] = i$. Once D_p and P have been computed, constructing C costs $O(\sum_{p=0}^{b} k_p) = O(2^b) = O(n/\log n)$ of time and $O(b2^b) = O(n)$ of space.

Let $q = \lfloor n/b \rfloor$ be the number of blocks of b bits in S. If b does not divide n, the last $n \bmod b$ bits will be treated separately. For the coding and decoding procedure, we iterate on each block of S times (so q times) the following steps:

1. Read from S the next b-bit block s, manipulated as an index of $[0, 2^b)$
2. Write in S' (the coding string) the value $P[s]$ as a binary number on $\lceil \log_2 b \rceil$ bits
3. Write in S' the string composed of the $L[s]$ most significant bits of $C[s]$

We end the coding process by writing in S' the $n \bmod b$ remaining bits of S (if any).

The decoding procedure of S' in S is:

1. Read from S' the $\lceil \log_2 b \rceil$ bits to form the value p
2. Read from S' the next $L[p]$ bits, representing an integer $i \in [0, k_p)$
3. Write in S the string $D_p[i]$

We end the decoding process by writing in S the $n \bmod b$ remaining bits of S' (if any).

Coding and decoding procedures clearly take $O(q) = O(n/\log n)$ time, once the tables L, P, D_p, C have been generated. The correctness of the coding and decoding is clear from symmetry of the above procedures.

It remains to show that the length of S' does not exceed $\log_2(\#S) + O(n \log\log n/\log n)$. Let p_i be the number of ones in the ith b-bit block of S, for $i \in \{1, \ldots, q\}$. From the coding procedure, the number of bits written in S' for the ith block is: $\lceil \log_2 b \rceil + \lceil \log_2 k_{p_i} \rceil$. Summing over all the blocks, we obtain the following upper bound for the length of S':

$$\sum_{i=1}^{q} \left(\lceil \log_2 b \rceil + \lceil \log_2 k_{p_i} \rceil \right) + (n \bmod b) \;=\; \left(\sum_{i=1}^{q} \log_2 k_{p_i} \right) + O(b + q \log b).$$

Observe that by super-additivity $\binom{a}{b} \cdot \binom{a'}{b'} \leqslant \binom{a+a'}{b+b'}$, so

$$\prod_{i=1}^{q} k_{p_i} \;=\; \prod_{i=1}^{q} \binom{b}{p_i} \;\leqslant\; \binom{bq}{\sum_i p_i} \;=\; \#\overline{S},$$

where $\overline{S}$ is the string composed of the first bq bits of S. Since the length and the number of ones between S and $\overline{S}$ differ by at most b, it follows that $|\log_2(\#S) - \log_2(\#\overline{S})| = O(b \log n) = O(\log^2 n)$. Therefore, we have that the length of S' is no more than:

$$\left(\log_2 \prod_{i=1}^{q} k_{p_i} \right) + O(b + q \log b) \;\leqslant\; \log_2(\#S) + O(n \log\log n/\log n)$$

as claimed, completing the proof. □

4 Entropy Analysis

The length of the coding of well-orderly map depends of the number of the edges of the well-orderly map.

The following two results are obtained from the analysis of the length of the code given in Lemma 10. The length of this code depends on the number of edges of the well-orderly map (see Fig. 2.9).

Theorem 4. *Every connected planar graph with n nodes and m edges can be encoded in linear time with at most* $4.91n + o(n)$ *bits or* $2.82m + o(m)$ *bits. In particular, the number of well-orderly maps with n nodes (resp. with m edges) is at most* $2^{4.91n+o(n)}$ *(resp.* $2^{2.82m+o(m)}$*).*

Proof (Sketch). From Lemma 10, we obtain an explicit coding composed of six binary strings $S_1, S_2, ..., S_6$ and four integers a_0, a_1, a'_0, i_2.

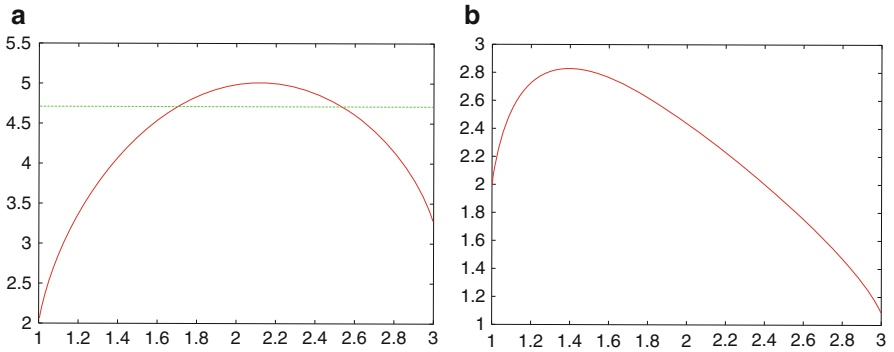

Fig. 2.9 (**a**) Number of bits necessary to encode a well-orderly map with $m = \alpha n$ edges, where $1 \leqslant \alpha \leqslant 3$. (**b**) Coding analyses: Number of bits per edges of a well-orderly map with $m = \alpha n$ edges, where $1 \leqslant \alpha \leqslant 3$

Thanks to Lemma 11, we can encode in linear time a planar graph with at most $\log_2(\#S_1) + \log_2(\#S_2) + \log_2(\#S_3) + \log_2(\#S_4) + \log_2(\#S_5) + \log_2(\#S_6) + O(\log n)$ bits. Computing the maximum length of the codes (over all parameters $a_0, a_1, ..., i_2$ and m or n) :

$$W = W(n,m) = \max_{a_0,a_1,...,i_2} \left\{ \log_2 \binom{n-a_0}{i_2-a_0} + \log_2 \binom{n-a_1}{a_0'} + \log_2 \binom{n+a_1}{a_1} + \log_2 \binom{a_1+a_0+a_0'}{a_0} + \log_2 \binom{n-a_1-a_0'}{n-a_1-a_0'-i_2} + \log_2 \binom{2n-i2}{m-n-i_2} \right\} + O(\log n),$$

we obtain (see Fig. 2.9a) that $W \leqslant 4.91n + o(n)$. Since G is connected, we have $n - 1 \leqslant m \leqslant 3n - 6$ and so $\log n = \log m + O(1)$. Hence, we also have (see Fig. 2.9b) that $W \leqslant 2.28m + o(m)$. □

Theorem 5. *Almost all unlabeled connected planar graphs on n nodes have at least* $1.85n$ *edges and at most* $2.44n$ *edges.*

Proof. (sketch). Our code can be parameterized with the number of edges. The length of the coding is no more than $W(m,n)$ bits. Using a reduction from arbitrary planar graphs to connected planar graphs, we can apply our upper bound. Combined with the $4.767n$ bit lower bound derived from the number of labelled planar graphs of [18], we derive two numbers $\alpha_1 = 1.85$ and $\alpha_2 = 2.44$ such that for a connected planar graph with αn edges, $\alpha_1 \leqslant \alpha \leqslant \alpha_2$, our representation is below $4.767n$ (See Fig. 2.9a). □

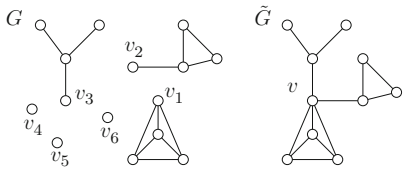

Fig. 2.10 Representation of a non-connected planar graph G by a triple $(k, \tilde{G}, v)$

The bound proposed here was for connected planar graphs. This bound can also be apply on planar graphs (not necessarily connected).

Remind that $p(n)$ is the number of unlabeled n-node planar graphs. Let $q(n)$ denote the number of unlabeled *connected* planar graphs on n nodes. To relate $p(n)$ and $q(n)$, we represent every planar graph G with $k \geqslant 1$ connected components by a triple $(k, \tilde{G}, v)$, where $\tilde{G}$ and v are defined as follows. Let v_i be any non-cut-vertex of the ith component of G. (Recall that a *cut-vertex* of a graph is a node whose removal strictly increases the number of connected components. A leaf of a tree being not a cut-vertex, it is clear that every connected graph has a non-cut-vertex). We merge all the connected components of G by identifying all the v_i's into a single node v as shown in Fig. 2.10.

Clearly $\tilde{G}$ is planar and connected. One can obtain G from $(k, \tilde{G}, v)$ by splitting v in $\tilde{G}$. All the $k' \leqslant k$ connected components obtained by this way are included in G (there is no risk to disconnect a single connected component of G as v_i's are not cut-vertices of each connected component of G). To fully recover G, we may add $k-k'$ isolated nodes. The number of nodes of $\tilde{G}$ is $n-(k'-k)-k'+1 = n-k+1$. From this representation, it follows that:

$$p(n) \leqslant \sum_{k=1}^{n} k \cdot q(n-k+1) \cdot (n-k+1) \leqslant n^3 q(n) \tag{2.1}$$

as $q(n)$ clearly increases with n.

In [7], it is shown that the number of n-node well-orderly maps is upper bounded by $\gamma^{n+o(n)}$ with $\gamma = 8/(\sqrt{189+114\sqrt{3}}-6\sqrt{3}-9) \approx 30.0612$. From the above discussion, we have:

Theorem 6. *The number $p(n)$ of unlabeled n-node planar graphs, for n large enough, satisfies*

$$p(n) \leqslant \gamma^{n+o(n)} \leqslant 2^{4.91n}.$$

5 Summary and Conclusion

Counting the number of *unlabeled* planar n-node graph is still an open problem. In this chapter, we show the best known upper bound that equals $30.0612^{n+o(n)}$. Moreover, we propose a compact encoding of planar graphs using a new combinatorial

object called well-orderly map with $\log_2 30.0612 \approx 4.91$ bits per node. The coding and the decoding can be done in linear time.

Since a planar graph can be represented by several well-orderly maps, a possible way to get a smaller bound would be to define a smaller class of planar maps. Finding a way to assign to any planar graph at most $2^{o(n)}$ planar embeddings is enough to get the asymptotic value of constant growth. Another direction consists in counting the number of *labeled* planar graphs and computing the average number of symmetries (automorphisms) per graph. The first step has been done by Gimenez and Noy [18], but the knowledge of the average number of symmetries is still a problem.

References

1. Alonso, L., Rémy, J.L., Schott, R.: A linear-time algorithm for the generation of trees. Algorithmica **17**(2), 162–182 (1997)
2. Banderier, C., Flajolet, P., Schaeffer, G., Soria, M.: Planar maps and airy phenomena. In: 27th International Colloquium on Automata, Languages and Programming (ICALP), vol. 1853 of Lecture Notes in Computer Science, pp. 388–402. Springer, New York (2000)
3. Barcucci, E., del Lungo, A., Pergola, E.: Random generation of trees and other combinatorial objects. Theor. Comput. Sci. **218**(2), 219–232 (1999)
4. Bodirsky, M., Kang, M.: Generating random outerplanar graphs. In: 1st Workshop on Algorithms for Listing, Counting, and Enumeration (ALICE) (2003)
5. Bonichon, N.: A bijection between realizers of maximal plane graphs and pairs of non-crossing Dyck paths. In: Formal Power Series & Algebraic Combinatorics (FPSAC) (2002)
6. Bonichon, N., Gavoille, C., Hanusse, N.: An information-theoretic upper bound of planar graphs using triangulation. In: 20th Annual Symposium on Theoretical Aspects of Computer Science (STACS), vol. 2607 of Lecture Notes in Computer Science, pp. 499–510. Springer, New York (2003)
7. Bonichon, N., Gavoille, C., Hanusse, N., Poulalhon, D., Schaeffer, G.: Planar graphs, via well-orderly maps and trees. Graph. Combinator. **22**, 1–18 (2006)
8. Chiang, Y.T., Lin, C.C., Lu, H.I.: Orderly spanning trees with applications to graph encoding and graph drawing. In: 12th Symposium on Discrete Algorithms (SODA), pp. 506–515. ACM-SIAM (2001)
9. Chiba, N., Nishizeki, T., Abe, S., Ozawa, T.: A linear algorithm for embedding planar graphs using pq-trees. J. Comput. Syst. Sci. **30**(1), 54–76 (1985)
10. Chiba, N., Nishizeki, T., Abe, S., Ozawa, T.: A linear algorithm for embedding planar graphs using pq-trees. J. Comput. Syst. Sci. **30**(1), 54–76 (1985)
11. Chuang, R.C.N., Garg, A., He, X., Kao, M.Y., Lu, H.I.: Compact encodings of planar graphs via canonical orderings and multiple parentheses. In: Guldstrand Larsen, K., Skyum, S., Winskel, G. (eds.) 25th International Colloquium on Automata, Languages and Programming (ICALP), vol. 1443 of Lecture Notes in Computer Science, pp. 118–129. Springer, New York (1998)
12. Denise, A., Vasconcellos, M., Welsh, D.J.A.: The random planar graph. Congressus Numerantium **113**, 61–79 (1996)
13. Diestel, R.: Graph Theory, 2nd edn., vol. 173 of Graduate Texts in Mathematics. Springer, New York (2000)
14. Epstein, P., Sack, J.R.: Generating triangulations at random. ACM Trans. Model. Comput. Simul. **4**, 267–278 (1994)
15. Frederickson, G.N., Janardan, R.: Efficient message routing in planar networks. SIAM J. Comput. **18**(4), 843–857 (1989)

16. Gavoille, C., Hanusse, N.: Compact routing tables for graphs of bounded genus. In: Wiedermann, J., van Emde Boas, P., Nielsen, M. (eds.) 26th International Colloquium on Automata, Languages and Programming (ICALP), vol. 1644 of Lecture Notes in Computer Science, pp. 351–360. Springer, New York (1999)
17. Gerke, S., McDiarmid, C.J.H.: On the number of edges in random planar graphs. Combinator. Probab. Comput. **13**, 165–183 (2004)
18. Giménez, O., Noy, M.: Asymptotic enumeration and limit laws of planar graphs. J. Am. Math. Soc. **22**, 309–329 (2009)
19. He, X., Kao, M.Y., Lu, H.I.: A fast general methodology for information-theoretically optimal encodings of graphs. SIAM J. Comput. **30**(3), 838–846 (2000)
20. Keeler, K., Westbrook, J.: Short encodings of planar graphs and maps. Discrete Appl. Math. **58**, 239–252 (1995)
21. Khodakovsky, A., Alliez, P., Desbrun, M., Schröder, P.: Near-optimal connectivity encoding of 2-manifold polygon meshes. Graph. Model. (2002). To appear in a special issue
22. King, D., Rossignac, J.: Guaranteed 3.67V bit encoding of planar triangle graphs. In: 11th Canadian Conference on Computational Geometry (CCCG), pp. 146–149 (1999)
23. Lipton, R.J., Tarjan, R.E.: A separator theorem for planar graphs. SIAM J. Appl. Math. **36**(2), 177–189 (1979)
24. Liskovets, V.A., Walsh, T.R.: Ten steps to counting planar graphs. Congressus Numerantium **60**, 269–277 (1987)
25. Liu, Y.: Enumeration of simple planar maps. Utilitas Math. **34**, 97–104 (1988)
26. Lu, H.I.: Improved compact routing tables for planar networks via orderly spanning trees. In: 8th Annual International Computing & Combinatorics Conference (COCOON), vol. 2387 of Lecture Notes in Computer Science, pp. 57–66. Springer, New York (2002)
27. Lu, H.I.: Linear-time compression of bounded-genus graphs into information-theoretically optimal number of bits. In: 13th Symposium on Discrete Algorithms (SODA), pp. 223–224. ACM-SIAM (2002)
28. McDiarmid, C.J.H., Steger, A., Welsh, D.J.A.: Random planar graphs. J. Combin. Theor. B **93**, 187–205 (2005)
29. Munro, J.I., Raman, V.: Succinct representation of balanced parentheses, static trees and planar graphs. In: 38th Annual IEEE Symposium on Foundations of Computer Science (FOCS), pp. 118–126. IEEE Computer Society Press (1997)
30. Nishizeki, T., Chiba, N.: Planar Graphs: Theory and Algorithms. North-Holland Mathematics Studies 140, Amsterdam (1988)
31. Osthus, D., Prömel, H.J., Taraz, A.: On random planar graphs, the number of planar graphs and their triangulations. J. Combin. Theor. B **88**, 119–134 (2003)
32. Pagh, R.: Low redundancy in static dictionaries with constant query time. SIAM J. Comput. **31**(2), 353–363 (2001)
33. Poulalhon, D., Schaeffer, G.: Optimal coding and sampling of triangulations. In: 30th International Colloquium on Automata, Languages and Programming (ICALP), vol. 2719 of Lecture Notes in Computer Science, pp. 1080–1094. Springer, New York (2003)
34. Rossignac, J.: Edgebreaker: Connectivity compression for triangle meshes. IEEE Trans. Visual. Comput. Graph. **5**(1), 47–61 (1999)
35. Schaeffer, G.: Random sampling of large planar maps and convex polyhedra. In: 31st Annual ACM Symposium on Theory of Computing (STOC), pp. 760–769 (1999)
36. Schnyder, W.: Embedding planar graphs on the grid. In: 1st Symposium on Discrete Algorithms (SODA), pp. 138–148. ACM-SIAM (1990)
37. Thorup, M.: Compact oracles for reachability and approximate distances in planar digraphs. In: 42nd Annual IEEE Symposium on Foundations of Computer Science (FOCS), pp. 242–251. IEEE Computer Society Press (2001)
38. Turán, G.: Succinct representations of graphs. Discrete Appl. Math. **8**, 289–294 (1984)
39. Tutte, W.T.: A census of planar triangulations. Cana. J. Math. **14**, 21–38 (1962)
40. Yannakakis, M.: Embedding planar graphs in four pages. J. Comput. Syst. Sci. **38**, 36–67 (1989)