Surface Split Decompositions and Subgraph Isomorphism in Graphs on Surfaces

Paul Bonsma

Surface Split Decompositions and Subgraph Isomorphism in Graphs on Surfaces

Paul Bonsma[1]

1 **Humboldt University Berlin, Computer Science Department, Unter den Linden 6, 10099 Berlin, Germany.**
bonsma@informatik.hu-berlin.de.

Abstract

The Subgraph Isomorphism problem asks, given a *host graph* G on n vertices and a *pattern graph* P on k vertices, whether G contains a subgraph isomorphic to P. The restriction of this problem to planar graphs has often been considered. After a sequence of improvements, the current best algorithm for planar graphs is a linear time algorithm by Dorn (STACS '10), with complexity $2^{O(k)} \cdot O(n)$.

We generalize this result, by giving an algorithm of the same complexity for graphs that can be embedded in surfaces of bounded genus. In addition, we simplify the algorithm and analysis. The key to these improvements is the introduction of surface split decompositions for bounded genus graphs, which generalize sphere cut decompositions for planar graphs. We extend the algorithm for the problem of counting and generating all subgraphs isomorphic to P, even for the case where P is disconnected. This answers an open question by Eppstein (JGAA'99).

1998 ACM Subject Classification F.2.2 Computations on discrete structures, G.2.2 Graph algorithms

Keywords and phrases Analysis of algorithms, parameterized algorithms, graphs on surfaces, subgraph isomorphism, dynamic programming, branch decompositions, counting problems

Digital Object Identifier 10.4230/LIPIcs.STACS.2012.531

1 Introduction

The Subgraph Isomorphism problem asks, given a *host graph* G on n vertices and a *pattern graph* P on k vertices, whether G contains a subgraph isomorphic to P. This is a well-studied problem that generalizes many other important problems, such as finding cliques, determining the girth, finding complete bipartite subgraphs, and finding a Hamilton path or cycle. See Eppstein [15] for a survey on previous results for this problem and its many applications. This problem is NP-complete in general, even for planar graphs. However, in many cases P can be considered to be a small fixed graph. In that case, a trivial polynomial time algorithm of complexity $n^{O(k)}$ exists. For general graphs, nothing better is known. When restricting G to be planar, this can be improved significantly: Eppstein [15] gave a linear time algorithm for Planar Subgraph Isomorphism for any fixed graph P on k vertices. This seems best possible. However to judge the practicality of such an algorithm, the dependency of the complexity on the value k is also essential. Hence we view the problem as a *parameterized problem* with parameter k. (See [14, 20] for background on parameterized algorithms.) Using this refined viewpoint, the complexity of Eppstein's algorithm [15] is $2^{O(k \log k)} \cdot O(n)$. This improved on previous algorithms for Planar Subgraph Isomorphism by Plehn and Voigt [21] of complexity $2^{O(k \log k)} \cdot n^{O(\sqrt{k})}$, and Alon et al. [3], of complexity $2^{O(k)} \cdot n^{O(\sqrt{k})}$. Finally, Dorn [11] improved the previous results and gave an algorithm of

29th Symposium on Theoretical Aspects of Computer Science (STACS'12).
Editors: Christoph Dürr, Thomas Wilke; pp. 531–542
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

complexity $2^{O(k)} \cdot O(n)$. Eppstein [16] also considered graphs of bounded genus, which generalize planar graphs. In [16], an algorithm for Subgraph Isomorphism in bounded genus graphs is given, with complexity $2^{O(k \log k)} \cdot O(n)$. In addition, Eppstein [16] considered the even more general graph class of apex-minor free graphs, and gave an $f(k) \cdot O(n)$ time algorithm, where $f(k)$ is a rapidly growing function of k. Subsequent results of Demaine and Hajiaghayi [7] imply that this complexity can be improved to $f(k) = 2^{O(k \log k)}$. The aforementioned results by Eppstein [15, 16] in fact hold for the more general *counting version* of the problem, where the number of subgraphs of G that are isomorphic to P should be computed. In addition, in the case P is connected, he gave an algorithm for *listing* all of these subgraphs in time $2^{O(k \log k)} \cdot O(n) + z \cdot k^{O(1)}$, where z is the number of such subgraphs.

New results In this paper, we give an algorithm of complexity $2^{O(k)} \cdot O(n)$ for the counting version of Subgraph Isomorphism, for the case where G has bounded genus. This generalizes the result for planar graphs in [11] and improves the complexity of the bounded genus result in [16]. In addition, we give an algorithm that lists all z subgraphs isomorphic to P in in time $2^{O(k)} \cdot O(n) + z \cdot k^{O(1)}$. This also holds for the case where P is disconnected, and therefore answers an open question from Eppstein [15]. This is made possible by developing a simpler method for counting disconnected subgraphs. Our results hold for graphs of bounded non-orientable genus as well. For simplicity, we describe the orientable case only, and discuss the non-orientable case in the full version of this paper [5]. We also remark that our algorithm can easily be modified to count *induced* subgraphs [5].

Related research There are many examples of problems that can be solved faster on planar graphs and generalizations such as bounded genus graphs and H-minor free graphs. For instance, for the aforementioned graph classes, many parameterized problems can be solved in *subexponential time* $2^{O(\sqrt{k})} \cdot O(n)$, see e.g. [2, 10, 6, 8]. The *theory of bidimensionality* [6, 8] easily gives subexponential time algorithms for many problems restricted to the aforementioned graph classes. However, Subgraph Isomorphism is not one of these, except for very special cases of P such as paths. An essential ingredient for many of these algorithmic results on planar graphs, bounded genus graphs and H-minor free graphs is *fast dynamic programming over tree decompositions and branch decompositions*. For many problems (such as for instance Longest Path, Maximum Leaf Tree), the best known dynamic programming algorithms for general graphs have a complexity of $2^{O(w \log w)} \cdot O(n)$, where w is the width of the given decomposition. However, when restricted to sparse graph classes, a lot of research has been devoted to showing that this can often be improved to $2^{O(w)} \cdot O(n)$ [1, 10, 11, 12, 13, 22]. In the case of planar graphs, an essential tool is given by a special kind of branch decompositions, called *sphere cut decompositions*. These were introduced by Seymour and Thomas [23], and their algorithmic usefulness was first demonstrated in 2005 (conference version) by Dorn et al. [13]. Loosely speaking, a branch decomposition for a graph G consists of a labeled tree T, and every edge $e \in E(T)$ partitions the edges of G into two graphs G_1 and G_2. In a sphere cut decomposition, for every $e \in E(T)$ a simple closed curve in the plane exists (a *noose*), that separates the plane into two regions, one containing G_1 and the other containing G_2. In the case that a sphere cut decomposition of width w is given, an improved complexity of $2^{O(w)} \cdot O(n)$ can be proved for many dynamic programming algorithms, by using the fact that solutions can cross the nooses in a limited number of ways, which can be encoded by *non-crossing partitions*; see [13, 10].

It is generally believed that many algorithms for planar graphs can be extended to graphs of bounded genus. However, in the past, making this step has always been an intricate task. For instance, Dorn et al. [12] consider the Hamilton Cycle problem and related problems on graphs of bounded genus, and reduce this case to the planar case by cutting the surface a

number of times along nooses. For the remaining planar case, dynamic programming over sphere cut decompositions is used, but this is relatively complex because the previous cuts need to be taken into account. Rué et al. [22] proposed a different dynamic programming method for graphs on surfaces: they define surface cut decompositions, where two subgraphs G_1 and G_2 defined by an edge of the branch decomposition are separated by a limited number of nooses, which have a limited number of common points. Since in the case of surfaces of higher genus, the boundary between G_1 and G_2 may become quite complex, an elaborate definition is required to characterize this [22]. After a few intermediate steps, the complexity bounds given in [22] are again based on counting non-crossing partitions.

New techniques and overview of the paper In Section 3 we give a dynamic programming algorithm for Subgraph Isomorphism that works for all graphs, when a branch decomposition is given. However, in order to give a good bound on its complexity, we restrict to bounded genus graphs and introduce a special kind of branch decomposition. *One of the main contributions of this paper is that we introduce surface split decompositions for graphs of bounded genus, and a strong but simple technique for bounding the resulting dynamic programming complexity.* This is a type of branch decomposition that directly generalizes sphere cut decompositions. It allows algorithms and analysis that are significantly simpler than previous dynamic programming algorithms for bounded genus graphs. In fact, our algorithm and analysis is even significantly simpler than that of some previous algorithms for planar graphs (e.g. [11]). Informally, our basic but crucial observation is that for surfaces of higher genus, it is irrelevant that the two subgraphs G_1 and G_2 defined by an edge of the branch decomposition can share a complex boundary; it is only relevant that there are two disjoint (connected) regions R_1 and R_2 in the surface such that G_i is drawn in R_i for $i = 1, 2$. This is because it is not necessary to consider the number of ways in which partial solutions can cross the boundary of the regions; we can argue on a higher level by appropriately applying a $2^{O(k)}$ bound on the total number of bounded genus graphs on k vertices, which avoids many of the technical challenges that were faced in previous papers for similar problems [1, 22, 11, 12]. For more details, see Section 4. In a subsequent paper, we will show how this technique can be applied to various other problems for graphs on surfaces.

In Section 5 we give an algorithm for finding surface split decompositions in linear time, where the width depends linearly on the graph diameter. For this we generalize a result by Tamaki [24] and Dorn [11], for finding low width sphere cut decompositions for planar graphs of bounded diameter. This is then applied in Section 6 to prove our main algorithmic results. To this end, we start with a standard technique for decomposing embedded graphs into *layers* of small branch width (see e.g. [2, 11, 15]). The main innovation in Section 6 is a new, simpler way (compared to [15]) of avoiding double-counting in the case where the pattern graph P is disconnected, by considering color-coded solutions.

We expect that the notion of surface split decompositions, our algorithm for finding them, and our technique for bounding the size of corresponding dynamic programming tables will be a stimulus for the algorithmic research on bounded genus graphs. We believe that it will enable the generalization of various existing algorithmic results for planar graphs, and that it should allow for the simplification of various known results for bounded genus graphs, and more general graphs. We remark that algorithms that are conceptually simpler are not only convenient for the reader or programmer; often they are also faster in practice. Indeed, when restricted to planar graphs, the constants in our complexity bound are significantly smaller than those in the algorithm by Dorn [11]. This is discussed in the full version of this paper [5]. The proofs and proof details that are omitted because of space constraints can also be found in [5]. We start by giving definitions in Section 2.

2 Preliminaries

For basic graph theoretical notations not defined here we refer to [9]. The main graphs that we will consider throughout will be *simple*, but we will construct auxiliary graphs that may have parallel edges and loops, i.e. that may be *multi-graphs*. The *distance* from u to v is the length of a shortest path from u to v. The *eccentricity* of a vertex $u \in V(G)$ is the maximum distance from u to v, over all $v \in V(G)$. By $d(v)$ we denote the *degree* of $v \in V(G)$, which is the number of incident edges. An *isomorphism* between two simple graphs G_1 and G_2 is a bijective function $\phi : V(G_1) \to V(G_2)$ such that $uv \in E(G_1)$ if and only if $\phi(u)\phi(v) \in E(G_2)$. The *Subgraph Isomorphism (counting) problem* takes as input a simple (host) graph G and a simple (pattern) graph P. The objective is to compute the number of subgraphs G' of G that are isomorphic to P. Such a subgraph G' is called a *P-isomorph*, or *isomorph* if the graph P is clear.

The set of leaves of a tree T is denoted by $L(T)$. A *branch decomposition* of a graph G is a tuple (T, μ) consisting of a ternary tree T, and a bijection $\mu : L(T) \to E(G)$. For a subset $S \subseteq L(T)$, we will use $\mu(S)$ to denote the set of images. Every edge $e_T \in E(T)$ defines a *middle set* $mid(e_T) \subseteq V(G)$ in the following way: let $T_1 \subseteq V(T)$ and $T_2 \subseteq V(T)$ denote the vertex sets of the two tree components of $T - e_T$. The edge sets $\mu(L(T) \cap T_1)$ and $\mu(L(T) \cap T_2)$ partition the edges of G. By G_1 and G_2 we denote the respective subgraphs of G induced by these edge sets. Now $\text{mid}(e_T)$ is defined as $V(G_1) \cap V(G_2)$. The *width* of (T, μ) is defined as $\max_{e_T \in E(T)} |\text{mid}(e_T)|$. A *rooted* branch decomposition is a tuple (T, μ) where T is a ternary tree, and a root $r \in L(T)$ is identified. In this case, μ is a bijection from $L(T) \backslash \{r\}$ to $E(G)$. Note that a rooted branch decomposition can easily be obtained from a branch decomposition. In the case of a rooted branch decomposition, for $e \in E(T)$, by $T_e \subseteq V(T)$ we denote the set of vertices of the component of $T - e$ that does not contain r. Similar to above, T_e defines a subgraph of G, which is denoted by G_e. Since T is ternary, every edge $e \in E(T)$ for which T_e is non-trivial (i.e. not a single vertex) has two *children*; these are the two edges of T_e that are adjacent to e. Observe that if $e_r \in E(T)$ is the edge incident with r, then $G_{e_r} = G$.

For an introduction to graphs embedded on surfaces we refer to [19]. Formally, a *surface* is a connected compact 2-manifold without boundary. For every integer $g \geq 0$, Let $\mathbb{S}_g$ denote a surface that is obtained by adding g handles to a sphere. Hence $\mathbb{S}_g$ is an orientable surface of genus g. For ease of presentation, all surfaces that we consider will be orientable. In the full version of this paper [5], we discuss the non-orientable case. A *region* of a surface is a connected open set. The *boundary* of a region R consists of all the points that lie in the closure of R but not in R itself. For a simple curve $C \subseteq \mathbb{S}_g$, all the points in C that are not the end points are called *interior points*.

An *embedding* ψ of a graph G into the surface $\mathbb{S}_g$ consists of an injective mapping of the vertices v of G to points $\psi(v)$ in $\mathbb{S}_g$, and a mapping of edges $e = uv$ of G to a simple curve $\psi(e)$ in $\mathbb{S}_g$ with end points $\psi(u)$ and $\psi(v)$, such that two edges may only intersect in their end points. To simplify terminology and notation, if ψ is an embedding of a graph G, then the images $\psi(v)$ and $\psi(e)$ (which are subsets of $\mathbb{S}_g$) for $v \in V(G)$ and $e \in E(G)$ will also be called *vertices* and *edges* of G, respectively. Let $X \subseteq \mathbb{S}_g$ be the union of all vertices and edges of an embedded graph G. The *faces* of G are the maximal regions of $\mathbb{S}_g \backslash X$. An embedding is a *2-cell embedding* if every face is homeomorphic to an open disc. For a graph G with n vertices and m edges, which is 2-cell embedded in $\mathbb{S}_g$ with f faces, by *Euler's formula* it holds that $n - m + f = 2 - 2g$. The boundary of every face F of G defines a closed walk (vertex sequence) $v_0, v_1, \ldots, v_{k-1}, v_0$ in G in a straightforward way, which will be called the

facial walk for F. If the length of W is k, then F will be called a *k-face*.

Given a 2-cell embedding ψ of a graph G in $\mathbb{S}_g$, we can define the following two related (multi-) graphs, and their 2-cell embeddings in $\mathbb{S}_g$. To construct the *dual graph* G^*, we draw one vertex v_F in every face F of G. For every edge $e \in E(G)$ that is incident with faces F and F', we draw an edge e^* in G^* from v_F to $v_{F'}$. e^* is called the *dual edge* of e and e the *primal edge* of e^*. The *radial graph* $\mathcal{R}_G$ of G (which is also called the *vertex-face incidence graph*) is obtained by starting with the vertex set $V(G)$ (the *original vertices*), and adding a vertex v_F for every face F of G (the *face vertices*). For every face F in G with corresponding facial walk $u_0, u_1, \ldots, u_{k-1}, u_0$, add k edges u_0v_F, u_1v_F, $\ldots$, $u_{k-1}v_F$ to $\mathcal{R}_G$, drawn in the region F, in the given order around v_F.

A *combinatorial embedding* π of a graph G consists of a cyclic order π_v on the incident edges for every vertex $v \in V(G)$. From an embedding of a graph we obtain the corresponding combinatorial embedding by considering the clockwise order of edges around every vertex. A *map* is a connected graph G together with a combinatorial embedding π. The set of all facial walks and therefore the number of faces of a map can easily be deduced (without constructing the embedding), so the number of faces f of a map is well-defined. The *genus* g of a map G, π is the solution to $n - m + f = 2 - 2g$, where $n = |V(G)|$, $m = |E(G)|$ and f is the number of faces of G, π. The *genus of a connected graph* G is the minimum g such that G admits a combinatorial embedding of genus g. Given a map G, π of genus g, a corresponding 2-cell embedding of G in $\mathbb{S}_g$ exists.

3 A dynamic program for counting colorful isomorphs

In this section we give a dynamic programming algorithm for the following generalization of Subgraph Isomorphism. An instance of the *Colorful Subgraph Isomorphism* problem consists of a colored host graph G, and a pattern graph P. The *coloring* of G is a function $\alpha : V(G) \to C$, with $C = \{1, \ldots, q\}$. This encodes a partition of $V(G)$ into q sets. We remark that this is not required to be a *proper* vertex coloring, so adjacent vertices may receive the same color. A subgraph G' of G is called *colorful* if for every color $x \in C$, G' contains a vertex v of color x. (Note that G' may have more than q vertices.) The objective is to count the number of colorful P-isomorphs of G. We now present an algorithm for this problem. (When $q = 1$, this is the original counting problem.)

Dynamic programming table Let (T, μ) be a rooted branch decomposition of G. For every edge $e \in E(T)$, we will form a dynamic programming table $\mathcal{T}_e$. Informally, this table will store information about all possible subgraphs of the graph G_e, on at most k vertices. Firstly, we distinguish between non-isomorphic subgraphs. Furthermore, subgraphs of G_e that are isomorphic but intersect differently with the 'boundary' $\mathrm{mid}(e)$ of G_e are also considered distinct. Finally, we keep track of the set of colors that appear in these subgraphs. Two subgraphs of G_e are only considered equivalent if they match in all three regards. In that case, there will be a single entry in the table that represents both subgraphs. We now define this formally.

Let H be a graph, and let γ be a mapping from $\mathrm{mid}(e)$ to $V(H) \cup \{\mathrm{nil}\}$, which is *injective on* $V(H)$. To be precise, every vertex of $V(H)$ occurs at most once as a γ-image, but multiple vertices may be mapped to nil. Furthermore, let $A \subseteq \{1, \ldots, q\}$. For such a tuple (H, γ, A), a subgraph G' of G_e is called an *(H, γ, A)-subgraph* if the following two properties hold.

- There is an isomorphism $\phi : V(G') \to V(H)$ with $\gamma(v) = \phi(v)$ for all $v \in \mathrm{mid}(e) \cap V(G')$, and $\gamma(v) = \mathrm{nil}$ for all $v \in \mathrm{mid}(e) \setminus V(G')$.
- For all colors $x \in C$: $x \in A$ if and only if G' contains a vertex of color x.

For $e \in E(T)$, the *dynamic programming table* $\mathcal{T}_e$ will now contain *entries* (H, γ, A, η), where H, γ and A are as defined above, and η is a non-negative integer. The idea is that such a table entry indicates that G_e contains exactly η non-equivalent (H, γ, A)-subgraphs. Table entries $(H_1, \gamma_1, A_1, \eta_1)$ and $(H_2, \gamma_2, A_2, \eta_2)$ are *equivalent* if the following properties hold.

- There is an isomorphism $\phi : V(H_1) \to V(H_2)$ such that for all $v \in \text{mid}(e)$, either $\gamma_1(v) = \gamma_2(v) = \text{nil}$, or $\phi(\gamma_1(v)) = \gamma_2(v)$ holds.
- $A_1 = A_2$.

Observe that the above definition satisfies the following property: a subgraph G' of G_e is both an (H_1, γ_1, A_1)-subgraph and an (H_2, γ_2, A_2)-subgraph if and only if $(H_1, \gamma_1, A_1, \eta_1)$ and $(H_2, \gamma_2, A_2, \eta_2)$ are equivalent. Therefore, when two entries $(H_1, \gamma_1, A_1, \eta_1)$ and $(H_2, \gamma_2, A_2, \eta_2)$ are equivalent, we can *merge* them by replacing them with the single entry $(H_1, \gamma_1, A_1, \eta_1 + \eta_2)$. We say that the table $\mathcal{T}_e$ is *k-correct* if

- for every tuple (H, γ, A), $\mathcal{T}_e$ contains an entry (H, γ, A, η) if and only if G_e contains exactly $\eta \geq 1$ graphs G' with $|V(G')| \leq k$ that are (H, γ, A)-subgraphs, and
- $\mathcal{T}_e$ contains no pairs of equivalent entries.

Dynamic programming update step Let $e \in E(T)$ be an edge with children f and g. We will now show how a k-correct table $\mathcal{T}_e$ for e can be obtained from k-correct tables $\mathcal{T}_f$ and $\mathcal{T}_g$ for f and g, respectively. We define two entries $(H_f, \gamma_f, A_f, \eta_f) \in \mathcal{T}_f$ and $(H_g, \gamma_g, A_g, \eta_g) \in \mathcal{T}_g$ to be *compatible* if for all $v \in \text{mid}(f) \cap \text{mid}(g)$, it holds that $\gamma_f(v) = \text{nil}$ if and only if $\gamma_g(v) = \text{nil}$. We now show how to *combine* two such compatible entries into an entry $(H_e, \gamma_e, A_e, \eta_e)$:

- For all $v \in \text{mid}(f) \cap \text{mid}(g)$ with $\gamma_f(v) \neq \text{nil}$ (and thus $\gamma_g(v) \neq \text{nil}$): identify the vertex $\gamma_f(v)$ of H_f with the vertex $\gamma_g(v)$ of H_g, and call the new vertex $\nu(v)$. This gives the graph H_e.
- For all $v \in \text{mid}(e)$: If $v \in \text{mid}(f) \backslash \text{mid}(g)$ then set $\gamma_e(v) = \gamma_f(v)$. If $v \in \text{mid}(g) \backslash \text{mid}(f)$ then set $\gamma_e(v) = \gamma_g(v)$. If $v \in \text{mid}(g) \cap \text{mid}(f)$ then set $\gamma_e(v) = \nu(v)$. By definition of $\text{mid}(e)$, this covers all cases and thus defines the function γ_e.
- Set $A_e = A_f \cup A_g$.
- Set $\eta_e := \eta_f \cdot \eta_g$.

It can be verified that if there are η_f (H_f, γ_f, A_f)-subgraphs in G_f, and η_g (H_g, γ_g, A_g)-subgraphs in G_g, then there are $\eta_f \cdot \eta_g$ (H_e, γ_e, A_e)-subgraphs in G_e that are the result of combining graphs of the former two types in every possible combination. However, there may also be (H_e, γ_e, A_e)-subgraphs of G_e that are the result of combining different types of subgraphs of G_f and G_g. Therefore, merging entries is required as well.

Assuming that we have k-correct tables $\mathcal{T}_f$ and $\mathcal{T}_g$ for f and g respectively, we construct a k-correct table $\mathcal{T}_e$ for e as follows: We start with an empty table $\mathcal{T}_e$. Then we consider every pair of compatible entries from $\mathcal{T}_f$ and $\mathcal{T}_g$, and combine them as described above. For every such combination, this yields a possible entry (H, γ, A, η) for $\mathcal{T}_e$. In case that H has more than k vertices, we ignore this possible entry. Otherwise, we check whether $\mathcal{T}_e$ already contains an equivalent entry (H', γ', A', η'). If so, we merge the two entries. If not, we add the entry (T, γ, A, η) to the table $\mathcal{T}_e$. Then we continue with the next pair of compatible entries from $\mathcal{T}_f$ and $\mathcal{T}_g$. This yields the following lemma.

▶ **Lemma 1.** *Let (T, μ) be a rooted branch decomposition for a colored graph G, and k be an integer. Let $e \in E(T)$ be an edge with children f and g, for which k-correct tables $\mathcal{T}_f$ and $\mathcal{T}_g$ are given. Then the table $\mathcal{T}_e$ that is constructed with the above dynamic programming update step is k-correct for e. The construction takes time $X^3 \cdot f(k) \cdot k^{O(1)}$, where $f(k)$ is the complexity of deciding whether two entries are equivalent, and X is an upper bound on the size of a k-correct table.*

This update step is the core of the following dynamic programming algorithm: First, for every edge $e \in E(T)$ that has no children, G_e consists of a single edge, so it is trivial to construct a k-correct table $\mathcal{T}_e$. For every edge $e \in E(T)$ with two children f and g, we compute a k-correct table $\mathcal{T}_e$ using the dynamic programming update step. After computing k-correct tables for all edges of T, we inspect the table $\mathcal{T}_{e_r}$ where e_r is the root edge of (T, μ). Since $G_{e_r} = G$, $\text{mid}(e_r) = \emptyset$. Therefore, $\mathcal{T}_{e_r}$ contains at most one entry (H, γ, A, η) such that H is isomorphic to P and $A = \{1, \ldots, q\}$. If there is such an entry (H, γ, A, η), we return η, and otherwise we return 0. The correctness of this procedure follows from the definitions. Combined with Lemma 1 this gives the following theorem. Note that a branch decomposition (T, μ) of a graph on m edges has $|E(T)| \in O(m)$.

▶ **Theorem 2.** *Let (T, μ) be a rooted branch decomposition of a colored graph G with m edges. In time $X^3 \cdot f(k) \cdot k^{O(1)} \cdot O(m)$, it can be computed how many colorful subgraphs of G are isomorphic to a given graph P on k vertices, where $f(k)$ is the complexity of deciding whether two entries are equivalent, and X is an upper bound on the size of a k-correct table.*

The above theorem applies to general graphs, for which an appropriate bound for X can be given (this way we would match Eppstein's result [15]). However, to obtain the desired improved complexity, we consider bounded genus graphs and introduce surface split decompositions in the next section. In the case that an embedding π of G of bounded genus is given, and (T, μ) is a surface split decomposition, we give a good upper bound for X.

4 Surface split decompositions and a bound for the table size

In this section, we introduce surface split decompositions for graphs embedded in $\mathbb{S}_g$, and demonstrate their usefulness. For graphs of genus 0 (planar graphs), the following definition is an alternative way to define sphere cut decompositions.

▶ **Definition 3.** Let G be a graph embedded in $\mathbb{S}_g$. A branch decomposition (T, μ) of G is called a *surface split decomposition* if for every $e \in E(T)$ and corresponding subgraphs G_1 and G_2 of G, there are disjoint regions $R_1 \subseteq \mathbb{S}_g$ and $R_2 \subseteq \mathbb{S}_g$ such that for $i = 1, 2$, all vertices and edges of G_i are drawn entirely in the closure of R_i.

Observe that this definition implies that all vertices in $\text{mid}(e)$ lie on the boundary of the closure of R_1 and on the boundary of the closure of R_2, so the closures are *not* disjoint (if $\text{mid}(e) \neq \emptyset$). We stress that it is crucial that R_1 and R_2 are connected *open* sets. If not, then firstly the above definition is not a generalization of sphere cut decompositions, but more importantly, the proof of Lemma 5 below fails. Even if the boundaries of the regions R_1 and R_2 are the same, this boundary is not necessarily a simple curve if $g \geq 1$. It may even be quite complex [22], but this does not matter. The definition easily extends to *maps* G, π of genus g: (T, μ) is a *surface split decomposition* of G, π if it is a surface split decomposition for any embedding of G in $\mathbb{S}_g$ that corresponds to the combinatorial embedding π.

Let (T, μ) be a surface split decomposition for a map G, π. We now give a bound on the size of a k-correct table. We will use a bound on the number of different graphs on n vertices, embedded in a surface of genus g. To be precise, we will consider *simple, connected* graphs G, that come with a combinatorial embedding π. Such a pair G, π is called a *simple map*. In addition, a tuple (u, v) of vertices is given such that $uv \in E(G)$. This is the *root*. Such a combination $G, \pi, (u, v)$ is called a *simple rooted map*. Two simple rooted maps $G, \pi, (u, v)$ and $G', \pi', (u', v')$ are *equivalent* if there is an isomorphism $\phi : V(G) \to V(G')$ with $\phi(u) = u'$, $\phi(v) = v'$, and that maps facial walks of G to facial walks of G'. In case

an edge labeling is given for both simple rooted maps, we also require that ϕ maps edges to edges of the same label. To bound the number of simple rooted maps, we apply a result by Bender and Canfield [4], which implies the following bound.

▶ **Theorem 4.** *There are $2^{O(n)} \cdot n^{O(g)}$ simple rooted maps of genus at most g on at most n vertices.*

▶ **Lemma 5.** *Let G, π be a simple map of genus at most g, for which a coloring with q colors and a surface split decomposition (T, μ) of width w are given. Let k be an integer. For $e \in E(T)$, let $\mathcal{T}_e$ be a k-correct table. Then $|\mathcal{T}_e| \in 2^q \cdot 2^w \cdot 2^{O(k)} \cdot k^{O(g)}$.*

Proof sketch: For all entries $(H, \gamma, A, \eta) \in \mathcal{T}_e$ where H has at most k vertices, we will *encode* H and γ by a simple rooted map H', π', (u, v) on at most $k+1$ vertices, together with a 0/1-labeling λ of a subset of the edges of H', and a 0/1-labeling ρ of the vertices in $\mathrm{mid}(e)$. This means that any two non-equivalent entries (H, γ, A, η) and (H', γ', A', η') either have $A \neq A'$, or yield a different labeling ρ, or yield non-equivalent rooted maps.

We use the following auxiliary graph. Since (T, μ) is a surface split decomposition, there are disjoint regions R_1 and R_2 in $\mathbb{S}_g$, such that G_e lies in the closure of R_1, and $\mathrm{mid}(e)$ lies on the boundary of both R_1 and R_2. Thus we can extend G_e by drawing a vertex u in R_2, and drawing edges in the closure of R_2 from u to every vertex in $\mathrm{mid}(e)$, while maintaining an embedding in $\mathbb{S}_g$. Number the vertices of $\mathrm{mid}(e)$ $v_1, \ldots, v_t$, corresponding to the clockwise order of edges around u.

Now we show how to construct the encoding H', π', ρ, λ for an entry $(H, \gamma, A, \eta) \in \mathcal{T}_e$. Firstly, for all vertices $v \in \mathrm{mid}(e)$, set $\rho(v) = 0$ if and only if $\gamma(v) = \mathrm{nil}$. The graph H' is constructed as follows. Since $\mathcal{T}_e$ is k-correct, H corresponds to a subgraph of G_e, so H can also be drawn in the closure of R_1, such that all vertices that are γ-images are drawn on the boundary of R_1. Start with such an embedding. Next, add the vertex u, and edges uv_i for every $v_i \in \mathrm{mid}(e)$ with $\gamma(v_i) \neq \mathrm{nil}$. Draw these as described in the previous paragraph. This yields a simple graph embedded in $\mathbb{S}_g$, on at most $k+1$ vertices. However, it may not be connected. Add edges between different components until the graph is connected. This yields H'. Clearly, drawing these new edges can be done while maintaining an embedding. Hence H' is embedded in $\mathbb{S}_g$, and the corresponding combinatorial embedding π' has genus at most g. To obtain a *rooted map*, choose i to be the lowest index such that $\rho(v_i) = 1$. Then the tuple (u, v_i) is chosen as the root of H', π'. That is, $v = v_i$. A *bridge* of a connected graph G is an edge $e \in E(G)$ such that $G - e$ is disconnected. For all bridges $e \in E(H')$, we set $\lambda(e) = 1$ if $e \in E(H)$, and $\lambda(e) = 0$ otherwise. It can now be verified that the rooted edge labeled map $H', \pi', (u, v)$, λ, function ρ and set A encode the entry; informally, when knowing $H', \pi', (u, v)$, λ and ρ, we can reconstruct H and γ.

There are at most 2^w possibilities for ρ, at most 2^q possibilities for A, at most $2^{O(k)} \cdot k^{O(g)}$ simple rooted maps on at most $k+1$ vertices of genus at most g (Theorem 4), and at most 2^k possible labelings λ (since a graph on $k+1$ vertices contains at most k bridges.) Therefore, the number of entries in a k-correct table is bounded by $2^q \cdot 2^w \cdot 2^{O(k)} \cdot k^{O(g)}$. □

Using the isomorphism testing algorithm for graphs of bounded genus by Miller [18] or Grohe [17], we can test in time $k^{O(g)}$ whether two entries are equivalent. Combining this fact with Theorem 2 and Lemma 5 gives the following theorem.

▶ **Theorem 6.** *Let G, π be a simple map with m edges of genus at most g, for which a coloring with q colors and a surface split decomposition (T, μ) of width w are given. Let P be a graph on k vertices. In time $8^q \cdot 8^w \cdot 2^{O(k)} \cdot k^{O(g)} \cdot O(m)$, it can be computed how many colorful subgraphs of G are isomorphic to P.*

5 Constructing surface split decompositions

Tamaki [24] gave a linear time algorithm for constructing a branch decomposition of width $2d+1$ of a graph G embedded in the sphere, when a vertex $r \in V(G)$ of eccentricity d is given. Dorn [11] gave a different presentation of the construction and showed that it yields in fact a sphere cut decomposition. We now generalize this result to surfaces of higher genus.

▶ **Theorem 7.** *Let G, π be a map of genus at most g, for which a vertex $r \in V(G)$ of eccentricity d is given. In linear time, a surface split decomposition (T, μ) of G of width at most $(2g+1)(4d+3)/2$ can be constructed.*

Before we construct the surface split decomposition (T, μ), we will construct a number of auxiliary graphs. The objective of the first stage of the construction is to construct a tree T^*, such that for every $e \in E(G)$, there is a unique vertex of T^* associated with e. This can be thought of as the function μ, from a subset of $V(T^*)$ to $E(G)$. We will show in Lemmas 8 and 9 below that this T^* is already 'almost' the desired surface split decomposition. However, T^* will not yet be ternary (though it will have maximum degree 3), and the vertices that are associated with edges of G are not necessarily leaves. This is subsequently addressed in the second stage of the construction.

For the *first stage*, we start by modifying the graph G as follows: for every edge $e = uv$, add two extra parallel edges between u and v, one on either side of e (while maintaining a combinatorial embedding). This ensures that all original edges of G are incident with two 2-faces, and that every vertex has degree at least 3. Denote the resulting embedded graph by G'. The edges in $E(G') \cap E(G)$ are called *original edges*, and edges in $E(G') \backslash E(G)$ are called *new edges*. Now construct $\mathcal{R}_{G'}$, the radial graph of G'. Let T_S be a BFS spanning tree of $\mathcal{R}_{G'}$, rooted at a vertex $r \in V(\mathcal{R}_{G'}) \cap V(G)$ of eccentricity d. Choose T^* to be a spanning tree of the dual graph of $\mathcal{R}_{G'}$, such that for all edges $e^* \in E(T^*)$, the corresponding primal edge e is not in T_S. Using Euler's formula it can be shown that there are now $2g$ edges of $\mathcal{R}_{G'}$ that are neither in T_S, nor are their dual edges in T^*. Add all of these edges to T_S, to obtain the graph T_S^+. This completes the first stage of the construction of a surface split decomposition.

We remark that there is a trivial bijection between the faces of $\mathcal{R}_{G'}$ and the edges of G', and a trivial bijection between the vertices of T^* and the faces of $\mathcal{R}_{G'}$, and therefore a resulting bijection from $V(T^*)$ to $E(G')$. Below, we refer to these bijections when speaking of 'the set of edges of G' or faces of $\mathcal{R}_{G'}$ that *corresponds to* a subset of $V(T^*)$'. The *subgraph of G' that corresponds to a set $T_1 \subseteq V(T^*)$* is the subgraph of G' induced by the edges that correspond to T_1. For an edge $e_T \in E(T^*)$, let T_1 and T_2 be the vertex sets of the two components of $T^* - e_T$. Informally, by joining the faces of $\mathcal{R}_{G'}$ that correspond to the vertices in T_1 and T_2, we can construct regions $R(T_1)$ and $R(T_2)$ that satisfy the following two properties.

▶ **Lemma 8.** *For an edge $e_T \in E(T^*)$, let T_1 and T_2 be the vertex sets of the two components of $T^* - e_T$, and let G'_1 and G'_2 be the subgraphs of G' that correspond to T_1 and T_2 respectively. For every such edge e_T, there exist disjoint regions $R(T_1) \subseteq \mathbb{S}_g$ and $R(T_2) \subseteq \mathbb{S}_g$, such that for $i = 1, 2$, all vertices and edges of G_i are drawn entirely in the closure of R_i.*

▶ **Lemma 9.** *For an edge $e_T \in E(T^*)$, let T_1 be the vertex set of one of the components of $T^* - e_T$, and let G'_1 be the subgraph of G' that corresponds to T_1. For every such edge e_T, there are at most $(2g+1)(4d+3)/2$ vertices of G'_1 that lie on the boundary of $R(T_1)$.*

Proof sketch: We use that the boundary of a region $R(T_1)$ is a subgraph of $\mathcal{R}_{G'}$, and contains at most $2g+1$ edges that are not in T_S. This holds because one of these edges is the primal

edge of e_T, and the other edges are edges of $E(T_S^+)\backslash E(T_S)$, of which there are at most $2g$ (which can be shown using Euler's formula). Secondly, one can show that the maximum length of a path in T_S is $4d+2$. Hence the boundary of the region $R(T_1)$ contains at most $(2g+1)(4d+3)$ edges. This number may be divided by 2 since $\mathcal{R}_{G'}$ is bipartite and only half of its vertices are vertices of G'. □

At this point, T^* is already close to a low-width surface split decomposition of G. It remains to make it into a ternary tree, of which only the leaves correspond to original edges of G'. This is easily done with elementary operations. The reason is that, because of the parallel edges that were added to G', a vertex $v_d \in V(T^*)$ has degree at most 2 if it corresponds to an original edge of G', and degree at most 3 otherwise. We omit the details of this final stage of the proof of Theorem 7.

6 Summary of the algorithm for bounded genus graphs

In this section we show how to combine the ingredients of the previous sections to obtain our main results; we present an algorithm for counting the number of subgraphs isomorphic to P in a graph G of bounded genus, and an algorithm for listing all of them. Without loss of generality, we may assume that G is connected. Throughout this section we use the following notations. We choose an arbitrary vertex $r \in V(G)$. Let d be the eccentricity of r. For $i = 0, \ldots, d$, define $L_i \subseteq V(G)$ to be the set of vertices at distance i of r. We call such a set a *layer*. Let $G_i^j = G[L_i \cup L_{i+1} \cup \ldots \cup L_j]$. The following property easily follows from Theorem 7 using a standard argument, see e.g. [15, 11].

▶ **Lemma 10.** *Let G, π be a map of genus g, with a vertex r of eccentricity d, and subgraphs G_i^j as defined above. For i, j with $0 \leq i \leq j \leq d$, a surface split decomposition of G_i^j of width at most $(2g+1)(4j-4i+7)/2$ can be constructed in time $O(n'+m')$, where $n' = |V(G_i^j)|$ and $m' = |E(G_i^j)|$.*

We now present a novel algorithm for counting the number of P-isomorphs in G, even for the case where P is disconnected. Let c be the number of components of P, which are denoted by $P^1, \ldots, P^c$. For $S \subseteq \{1, \ldots, c\}$, denote by P^S the subgraph of P induced by the components with labels in S. Formally, $P^S = P[\cup_{i \in S} V(P^i)]$. For every G_i^j, we consider the following coloring, which uses the color set $\{1, \ldots, j-i+1\}$: the vertices of layer L_x are all colored with color $x-i+1$. For $0 \leq i \leq j \leq d$ with $j-i < k$ and $S \subseteq \{1, \ldots, c\}$, we define $\mathrm{DPC}_i^j(S)$ to be the number of *colorful* subgraphs of G_i^j that are isomorphic to P^S. In other words, this is the number of P^S-isomorphs that use all layers of G_i^j. Combining Lemma 10 with Theorem 6 yields the following proposition.

▶ **Proposition 11.** *Let G, π be a connected simple map of genus at most g, let P be a graph on k vertices with c components, and let $DPC_i^j(S)$ and G_i^j be as defined above. For given i, j with $0 \leq i \leq j \leq d$ and $j-i < k$, and given set $S \subseteq \{1, \ldots, c\}$, $DPC_i^j(S)$ can be computed in time $2^{O(gk)} \cdot O(n'+m')$, where $n' = |V(G_i^j)|$ and $m' = |E(G_i^j)|$.*

For $j \in \{0, \ldots, d\}$, we define $\mathrm{DPT}^j(S)$ to be the number of subgraphs of G_0^j that are isomorphic to P^S. Note that these are not required to be colorful. In particular, if $S = \emptyset$, then we define $\mathrm{DPT}^j(S) = 1$. To avoid the discussion of trivial cases, we simply define $\mathrm{DPC}_i^j(S) = 0$ if $i < 0$, and $\mathrm{DPT}^j(S) = 0$ if $j < 0$. It can be shown that the following recursion for computing $\mathrm{DPT}^j(S)$ is correct.

▶ **Lemma 12.** *For every $j \in \{0,\dots,d\}$ and $S \subseteq \{1,\dots,c\}$, it holds that $DPT^j(S) = DPT^{j-1}(S) + \sum DPT^{j-x-1}(S_1) \cdot DPC^j_{j-x+1}(S_2)$, where the summation is over all partitions of S into S_1 and S_2 with $S_2 \neq \emptyset$, and all $x \in \{1,\dots,k\}$.*

▶ **Theorem 13.** *Let G,π be a simple map with m edges, of genus at most g, and let P be a (possibly disconnected) graph on k vertices. In time $2^{O(gk)} \cdot O(m)$, the number of subgraphs of G that are isomorphic to P can be computed.*

Proof: In the *first stage* of the algorithm, we compute $\mathrm{DPC}^j_i(S)$ for every i, j with $0 \leq i \leq j \leq d$ and $j - i < k$, and every $S \subseteq \{1,\dots,c\}$. There are 2^c choices of S, so computing DPC^j_i for one choice of i and j and all possible choices of S takes time $2^c \cdot 2^{O(gk)} \cdot O(n'+m')$, where $n' = |V(G^j_i)|$ and $m' = |E(G^j_i)|$ (Proposition 11). Since every vertex and every edge of G appears in G^j_i for at most $O(k^2)$ choices of i and j, this yields a total complexity of $2^c \cdot 2^{O(gk)} \cdot k^2 \cdot O(n+m) \subseteq 2^{O(gk)} \cdot O(m)$ for the first stage. (G is connected, so $n \in O(m)$.) The *second stage* of the algorithm uses the recursion from Lemma 12 for computing $\mathrm{DPT}^j(S)$ for every $j \in \{0,\dots,d\}$ and $S \subseteq \{1,\dots,c\}$. One can show that this takes time $3^c \cdot O(k) \cdot O(d) \subseteq 3^c \cdot O(k) \cdot O(n) \subseteq 2^{O(k)} \cdot O(m)$. Finally, the algorithm returns the value of $\mathrm{DPT}^d(\{1,\dots,c\})$, which is the total number of P-isomorphs of $G^d_0 = G$. □

▶ **Corollary 14.** *Let g be a constant. Let G,π be a simple map with m edges, of genus at most g, and let P be a (possibly disconnected) graph on k vertices. In time $2^{O(k)} \cdot O(n)$, the number of subgraphs of G that are isomorphic to P can be computed.*

(Here we used $m \in O(n+g) = O(n)$.) Since the above algorithm is a rather simple algorithm that uses only additions and multiplications for the counting, it can be extended to an algorithm for listing all isomorphs. We sketch how this can be done for the second stage of the algorithm. We first run the above counting algorithm and compute all values $\mathrm{DPT}^j(S)$ and $\mathrm{DPC}^j_i(S)$. Next, we apply a backtracking stage where we recursively mark the combinations of i, j and S that contribute to the total number of isomorphs. In a third stage, we apply the counting algorithm again, but instead of computing the values for $\mathrm{DPT}^j(S)$ and $\mathrm{DPC}^j_i(S)$, we compute a *list* of all corresponding subgraphs of G for all combinations of i, j and S that actually contribute to the total number of isomorphs. The steps of the algorithm where lists are constructed can then be attributed to the construction of the final list of P-isomorphs, and thus their complexity can be bounded by $zk^{O(1)}$, where z is the number of P-isomorphs. All other steps can be done with the same complexity as the counting algorithm. This yields:

▶ **Theorem 15.** *Let G,π be a simple map with m edges, of genus at most g, and let P be a (possibly disconnected) graph on k vertices. In time $2^{O(gk)} \cdot O(m) + zk^{O(1)}$, all subgraphs of G that are isomorphic to P can be generated, where z is the number of such subgraphs.*

Acknowledgement The author would like to thank Frederic Dorn for the introduction to the subject and the many inspiring discussions.

References

1 I. Adler, F. Dorn, F. Fomin, I. Sau, and D. Thilikos. Fast minor testing in planar graphs. In *ESA '10*, volume 6346 of *LNCS*, pages 97–109. Springer, Berlin, 2010.

2 J. Alber, H. Fernau, and R. Niedermeier. Parameterized complexity: exponential speed-up for planar graph problems. *J. Algorithms*, 52(1):26–56, 2004.

3 N. Alon, R. Yuster, and U. Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995.

4 E. A. Bender and E. R. Canfield. The asymptotic number of rooted maps on a surface. *J. Combin. Theory Ser. A*, 43(2):244–257, 1986.

5 P. Bonsma. Surface split decompositions and subgraph isomorphism in graphs on surfaces. eprint arXiv:1109.4554, 2011.

6 E. D. Demaine, F. V. Fomin, M. Hajiaghayi, and D. M. Thilikos. Subexponential parameterized algorithms on bounded-genus graphs and H-minor-free graphs. *J. ACM*, 52(6):866–893, 2005.

7 E. D. Demaine and M. Hajiaghayi. Equivalence of local treewidth and linear local treewidth and its algorithmic applications. In *SODA '04*, pages 840–849. SIAM, Philadelphia, PA, USA, 2004.

8 E. D. Demaine and M. Hajiaghayi. The bidimensionality theory and its algorithmic applications. *Comput. J.*, 51(3):292–302, 2008.

9 R. Diestel. *Graph theory*, volume 173 of *Graduate Texts in Mathematics*. Springer, Heidelberg, fourth edition, 2010.

10 F. Dorn. *Designing subexponential algorithms: problems, techniques and structures*. PhD thesis, University of Bergen, Norway, 2007.

11 F. Dorn. Planar subgraph isomorphism revisited. In *STACS '10*, volume 5 of *LIPIcs*, pages 263–274. Dagstuhl Publishing, Dagstuhl, Germany, 2010.

12 F. Dorn, F. V. Fomin, and D. M. Thilikos. Fast subexponential algorithm for non-local problems on graphs of bounded genus. In *Algorithm theory—SWAT '06*, volume 4059 of *LNCS*, pages 172–183. Springer, Berlin, 2006.

13 F. Dorn, E. Penninkx, H. L. Bodlaender, and F. V. Fomin. Efficient exact algorithms on planar graphs: exploiting sphere cut decompositions. *Algorithmica*, 58(3):790–810, 2010.

14 R. G. Downey and M. R. Fellows. *Parameterized complexity*. Springer-Verlag, New York, 1999.

15 D. Eppstein. Subgraph isomorphism in planar graphs and related problems. *J. Graph Algorithms Appl.*, 3(3):1–27, 1999.

16 D. Eppstein. Diameter and treewidth in minor-closed graph families. *Algorithmica*, 27(3-4):275–291, 2000.

17 M. Grohe. Isomorphism testing for embeddable graphs through definability. In *STOC'00*, pages 63–72. ACM, New York, 2000.

18 G. Miller. Isomorphism testing for graphs of bounded genus. In *STOC'80*, pages 225–235. ACM, New York, 1980.

19 B. Mohar and C. Thomassen. *Graphs on surfaces*. Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press, Baltimore, MD, 2001.

20 R. Niedermeier. *Invitation to fixed-parameter algorithms*, volume 31 of *Oxford Lecture Series in Mathematics and its Applications*. Oxford University Press, Oxford, 2006.

21 J. Plehn and B. Voigt. Finding minimally weighted subgraphs. In *WG '90*, volume 484 of *LNCS*, pages 18–29. Springer, Berlin, 1991.

22 J. Rué, I. Sau, and D. Thilikos. Dynamic programming for graphs on surfaces. In *ICALP '10*, volume 6198 of *LNCS*, pages 372–383. Springer Berlin / Heidelberg, 2010.

23 P. D. Seymour and R. Thomas. Call routing and the ratcatcher. *Combinatorica*, 14(2):217–241, 1994.

24 H. Tamaki. A linear time heuristic for the branch-decomposition of planar graphs. In *ESA '03*, volume 2832 of *LNCS*, pages 765–775. Springer Berlin / Heidelberg, 2003.