Symposium on Theoretical Aspects of Computer Science 2010 (Nancy, France), pp. 263-274
www.stacs-conf.org

PLANAR SUBGRAPH ISOMORPHISM REVISITED

FREDERIC DORN [1]

[1] Department of Informatics, University of Bergen, Norway
E-mail address: frederic.dorn@ii.uib.no

ABSTRACT. The problem of SUBGRAPH ISOMORPHISM is defined as follows: Given a *pattern* H and a *host graph* G on n vertices, does G contain a subgraph that is isomorphic to H? Eppstein [SODA 95, J'GAA 99] gives the first linear time algorithm for subgraph isomorphism for a fixed-size pattern, say of order k, and arbitrary planar host graph, improving upon the $O(n^{\sqrt{k}})$-time algorithm when using the "Color-coding" technique of Alon et al [J'ACM 95]. Eppstein's algorithm runs in time $k^{O(k)}n$, that is, the dependency on k is superexponential. We improve the running time to $2^{O(k)}n$, that is, single exponential in k while keeping the term in n linear. Next to deciding subgraph isomorphism, we can construct a solution and count all solutions in the same asymptotic running time. We may enumerate ω subgraphs with an additive term $O(\omega k)$ in the running time of our algorithm. We introduce the technique of "embedded dynamic programming" on a suitably structured graph decomposition, which exploits the number and topology of the underlying drawings of the subgraph pattern (rather than of the host graph).

Introduction

In the literature, we often find results on polynomial time or even linear time algorithms for NP-hard problems. Take for example the NP-complete problem of computing an optimal tree-decomposition of a graph. Bodlaender [3] gives a linear time algorithm—restricted to graphs of constant treewidth. The Graph Minor Theory by Robertson and Seymour implies amongst others that there is an $O(n^3)$ algorithm for the disjoint path problem, that is for finding disjoint paths between a constant number of terminals. Taking a closer look at such results, one notices that a function exponential in size of some constant c is hidden in the O-notation of the running time—here, c is the treewidth and the number of terminals, respectively. In another line of research, *parameterized complexity*, the primary goal is to rather find algorithms that minimize the exponential term of the running time—the exponential function of the *problem parameter* k. The first step here is to prove that such an algorithm with a separate exponential function exists, that is, that the studied problem is *fixed parameter tractable (FPT)* [13, 16, 21]. Such problem has an algorithm with time complexity bounded by a function of the form $f(k) \cdot n^{O(1)}$, where the *parameter*

1998 ACM Subject Classification: F.2.2;G.2.1;G.2.2.
Key words and phrases: Graph algorithms; Subgraph Isomorphism; NP-hard problems; Dynamic programming; Topological graph theory.
Supported by the Research Council of Norway.

SYMPOSIUM ON THEORETICAL ASPECTS OF COMPUTER SCIENCE

27th Symposium on Theoretical Aspects of Computer Science, Nancy, 2010
Editors: Jean-Yves Marion, Thomas Schwentick
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany
Digital Object Identifier: 10.4230/LIPIcs.STACS.2010.2460

function f is a computable function only depending on k. The second step in the design of FPT-algorithms is to decrease the growth rate of the parameter function.

We can identify two different trends in which exact algorithms are improved. Either one decreases the degree of the polynomial term in the asymptotic running time, or one focusses on obtaining parameter functions with better exponential growth. In the present work, we achieve both goals for the computational problem Planar Subgraph Isomorphism.

Subgraph Isomorphism generalizes many important graph problems, such as Hamiltonicity, Longest Path, and Clique. It is known to be NP-complete, even when restricted to planar graphs [18]. Until now, the best known algorithm to solve Subgraph Isomorphism, that is to find a subgraph of a given host graph isomorphic to a pattern H on k vertices, is the naïve exhaustive search algorithm with running time $O(n^k)$ and no FPT-algorithm can be expected here [13]. For a pattern H of treewidth at most t, Alon et al. [1] give an algorithm of running time $2^{O(k)}n^{O(t)}$. For Planar Subgraph Isomorphism, given planar pattern and input graph, some considerable improvements have been made mostly during the 90's ([23], [1]). The current benchmark has been set by Eppstein [14] to $k^{O(k)}n$, by employing graph decomposition methods, similar to the Baker-approach [2] for approximating NP-complete problems on planar graphs. Eppstein's algorithm is actually the first FPT-algorithm for Planar Subgraph Isomorphism with k as parameter. Eppstein poses three open problems: a) whether one can extend the technique in [1] to improve the dependence on the size of the pattern from $k^{O(k)}$ to $2^{O(k)}$ for the decision problem of subgraph isomorphism; and whether one can achieve similar improvements, b) for the counting version, and c) for the listing version of the subgraph isomorphism problem.

Our results. In this work, we do not only achieve this single exponential behavior in k for all three problems—without applying the randomized coloring technique—we also keep the term in n linear. That is, we give an algorithm for Planar Subgraph Isomorphism for a pattern H of order k with running time $2^{O(k)}n$. Next to deciding subgraph isomorphism, we can construct a solution and count all solutions in the same asymptotic running time. We may list ω subgraphs with an additive term $O(\omega k)$ in the running time of our algorithm. Our algorithm also improves the time complexity of [17] for large patterns of size $k \in o(\sqrt{n}\log n)$.

The novelty of our result comes from *embedded dynamic programming*, a technique we find interesting on its own. Here, one decomposes the graph by separating it into induced subgraphs. In the dynamic programming step, one computes partial solutions for the separated subgraphs, that are updated to an overall solution for the whole graph. In ordinary dynamic programming, one would argue how the subgraph pattern hits separators of the host graph. Instead, in embedded dynamic programming for subgraph isomorphism, we proceed exactly the opposite way: we look at how separators can be routed through the subgraph pattern. As a consequence, we bound the number of partial solutions by a function of both the separator size of the host graph and the pattern size—as it turns out, for the planar subgraph isomorphism problem, that function is single exponential in the number of vertices of the pattern. To obtain a good bound on the parameter function, we apply several fundamental enumerative combinatorics results in the technical sections of this work. Next to the number of face-vertex sequences in embedded graphs, these counting results give an upper bound on the number of planar drawings of the pattern.

Our algorithm is divided into two parts with the second part being the aforementioned embedded dynamic programming. For keeping the time complexity of our algorithm linear in the size of the host graph, we give a fast method for computing *sphere-cut decompositions*—natural extensions of tree-decompositions to plane graphs—with separators of size linearly bounded by the size of the subgraph pattern.

Theorem 0.1. *Let G be a planar graph on n vertices and H a pattern of order k. We can decide if there is a subgraph of G that is isomorphic to H in time $2^{O(k)}n$. We find subgraphs and count subgraphs of G isomorphic to H in time $2^{O(k)}n$ and enumerate ω subgraphs in time $2^{O(k)}n + O(\omega k)$.*

Let us mention that for k-Longest Path on planar graphs, the authors of [12] give the first algorithm with subexponential running time behaviour, namely $2^{O(\sqrt{k})}n + O(n^3)$, employing the techniques *Bidimensionality* and topological dynamic programming. *Bidimensionality Theory* employs results of Graph Minor Theory for planar graphs [24] and other structural graph classes to algorithmic graph theory (entry [6], for a survey [7]). Unfortunately, Bidimensionality does only work for finding specific patterns in a graph, such as k-paths, but not for subgraph isomorphism problems in general. For a survey on other planar subgraph isomorphism problems with restricted patterns, please consider [14].

Organization. Following the definitions in Section 1, we state in Section 2 how to obtain a sphere-cut decomposition of small width. In Section 3 we restrict Planar Subgraph Isomorphism to Plane Subdrawing Equivalence. We give some technical lemmas in Section 3.1 to bound the number of ways a separator of the sphere-cut decomposition can be routed through a plane pattern. We describe embedded dynamic programming in Section 3.2 and subsume the entire algorithm for Plane Subdrawing Equivalence in Section 3.3. In Section 4 we extend our algorithm for solving Planar Subgraph Isomorphism.

1. Preliminaries

Subgraph isomorphism. Let G, H be two graphs. We call G and H *isomorphic* if there exists a bijection $\nu : V(G) \to V(H)$ with $\{v, w\} \in E(G) \Leftrightarrow \{\nu(v), \nu(w)\} \in E(H)$. We call H *subgraph isomorphic to* G if there is a subgraph H' of G isomorphic to H.

Branch decompositions. A *branch decomposition* $\langle T, \mu\rangle$ of a graph G consists of an unrooted ternary tree T (internal vertex-degree 3) and a bijection $\mu : L \to E(G)$ from the set L of leaves of T to the edge set of G. We define for every edge e of T the *middle set* $\mathrm{mid}(e) \subseteq V(G)$ as follows: Let T_1 and T_2 be the two connected components of $T \setminus \{e\}$. Then let G_i be the graph induced by the edge set $\{\mu(f) : f \in L \cap V(T_i)\}$ for $i \in \{1, 2\}$. The *middle set* is the intersection of the vertex sets of G_1 and G_2, i.e., $\mathrm{mid}(e) := V(G_1) \cap V(G_2)$. The *width* bw of $\langle T, \mu\rangle$ is the maximum order of the middle sets over all edges of T, i.e., $\mathrm{bw}(\langle T, \mu\rangle) := \max\{|\,\mathrm{mid}(e)| \colon e \in T\}$. An optimal branch decomposition of G is defined by a tuple $\langle T, \mu\rangle$ which provides the minimum width, the *branchwidth* $\mathrm{bw}(G)$.

Plane graphs and equivalent drawings. Let Σ be the unit sphere. A *planar drawing* or simply *drawing* of a graph G with vertex set $V(G)$ and edge set $E(G)$ maps vertices to points in the sphere, and edges to simple curves between their end vertices, such that edges do not cross, except in common end vertices. A *plane graph* is a graph G together with a planar drawing. A *planar graph* is a graph that admits a planar drawing. For details, see e.g. [9]. The set of *faces* $F(G)$ of a plane graph G is defined as the union of the connected regions of $\Sigma \setminus G$. A subgraph of a plane graph G, induced by the vertices and edges incident to a face $f \in F(G)$, is called a *bound* of f. If G is 2-connected, each bound of a face is a cycle. We call this cycle *face-cycle* (for further reading, see e.g. [9]). For a subgraph H of a plane graph G, we refer to the drawing of G reduced to the vertices and edges of H as a *subdrawing* of G. Consider any two drawings G_1 and G_2 of a planar graph G. A *homeomorphism* of G_1 onto G_2 is a homeomorphism of Σ onto itself which maps vertices, edges, and faces of G_1 onto vertices, edges, and faces of G_2, respectively. We call two planar drawings *equivalent*, if there is a homeomorphism from one onto the other.

Theorem 1.1. *e.g.* [9] *Every 3-connected planar graph has a unique drawing in a sphere Σ up to homeomorphism.*

Proposition 1.2. *e.g.* [22] *Every planar n-vertex graph has $2^{O(n)}$ non-equivalent drawings.*

Remark 1.3. Let G and H be two plane graphs. If their drawings are equivalent, then G is isomorphic to H. On the contrary, if G is isomorphic to H and neither graphs are 3-connected, then their drawings are not necessarily equivalent.

Triangulations. We call a plane graph G a *planar triangulation* or simply a *triangulation* if every face in $F(G)$ is bounded by a triangle (a cycle of length three). If H is a subdrawing of a triangulation G, we call G a *triangulation of H*.

Nooses and combinatorial nooses. A *noose* of a Σ-plane graph G is a simple closed curve in Σ that meets G only in vertices. From the Jordan Curve Theorem, it then follows that nooses separate Σ into two regions. Let $V(N) = N \cap V(G)$ be the vertices and $F(N)$ be the faces intersected by a noose N. The *length* of N is the number $|V(N)|$ of vertices in $V(N)$. The clockwise order in which N meets the vertices of $V(N)$ is a cyclic permutation π on the set $V(N)$.

Remark 1.4. Let a plane graph H be a subdrawing of a plane graph G. Every noose N in G is also a noose in H and $N \cap V(H) \subseteq N \cap V(G)$.

A *combinatorial noose* $N_C = [v_0, f_0, v_1, f_1, \dots, f_{\ell-1}, v_\ell]$ in a plane graph G is an alternating sequence of vertices and faces of G, such that

- f_i is a face incident to both v_i, v_{i+1} for all $i < \ell$,
- $v_0 = v_\ell$ and the vertices $v_1, \dots, v_\ell$ are mutually distinct and
- if $f_i = f_j$ for any $i \neq j$ and $i, j = 0, \dots, \ell - 1$, then the vertices v_i, v_{i+1}, v_j, and v_{j+1} do not appear in the order $(v_i, v_j, v_{i+1}, v_{j+1})$ on the bound of face $f_i = f_j$.

The *length* of a combinatorial noose $[v_0, f_0, v_1, f_1, \dots, f_{\ell-1}, v_\ell]$ is ℓ.

Remark 1.5. The order in which a noose N intersects the faces $F(N)$ and the vertices $V(N)$ of a plane graph G gives a unique alternating face-vertex sequence of $F(N) \cup V(N)$ which is a combinatorial noose N_C. Conversely, for every combinatorial noose N_C there exists a noose N with face-vertex sequence N_C.

We may view combinatorial nooses as equivalence classes of nooses, that can be represented by the same face-vertex sequence.

Sphere cut decompositions. For a Σ-plane graph G, we define a *sphere cut decomposition* or *sc-decomposition* $\langle T, \mu, \pi \rangle$ as a branch decomposition which for every edge e of T has a noose N_e that cuts Σ into two regions Δ_1 and Δ_2 such that $G_i \subseteq \Delta_i \cup N_e$, where G_i is the graph induced by the edge set $\{\mu(f) : f \in L \cap V(T_i)\}$ for $i \in \{1, 2\}$ and $T_1 \dot\cup T_2 = T \setminus \{e\}$. Thus N_e meets G only in $V(N_e) = \mathrm{mid}(e)$ and its length is $|\mathrm{mid}(e)|$. The vertices of $\mathrm{mid}(e) = V(G_1) \cap V(G_2)$ are enumerated according to a cyclic permutation π on $\mathrm{mid}(e)$.

The following two propositions will be crucial in that they give us upper bounds on the number of partial solutions we will compute in our dynamic programming approach. With both propositions, we will bound the number of combinatorial nooses in a plane graph by the number of cycles in the triangulation of some auxiliary graph.

Proposition 1.6. *([4]) No planar n-vertex graph has more than $2^{1.53n}$ simple cycles.*

Proposition 1.7. *([27]) The number of non-isomorphic maximal planar graphs on n vertices is approximately $2^{3.24n}$.*

Proposition 1.7 also gives a bound on the number of non-isomorphic triangulations. Any drawing of a maximal planar graph G must be a triangulation, otherwise G would not be maximal. With Theorem 1.1, every maximal planar graph has a unique drawing which is a triangulation. On the other hand, every triangulated graph is maximal planar.

2. Computing sphere-cut decompositions in linear time

In this section we sketch an algorithm for computing sc-decompositions of bounded width. Let H be a connected subgraph of G with $|V(H)| = k$, and let $v \in V(H)$. Then H is a subgraph of the induced subgraph G^v of G, where $G^v = G[S]$ with $S = \{w \in S \mid \text{dist}(v, w) \leq k\}$ ($\text{dist}(v, w)$ denotes the length of a shortest path between v and w in G). This observation helps us to shrink the search space of our algorithm by cutting out chunks of G of bounded width and solve subgraph isomorphism separately on each chunk. With the algorithm of Tamaki [26], one can compute a branch decomposition of G^v of width $\leq 2k + 1$, following similar ideas as in the approach of Baker [2] for tree decompositions. With some simple modifications, we achieve the same result for sc-decompositions. In an extended version of this paper [10], we prove the following lemma and give an algorithm that computes a sc-decomposition of bounded width in linear time.

Lemma 2.1. *([2],[26],[10]) Let G be a plane graph with a rooted spanning tree whose root-leaf-paths have length $\leq k$. We can find an sc-decomposition of width $2k+1$ in time $O(kn)$.*

3. Plane Subdrawing Equivalence

In this section, we study the variant of the subgraph isomorphism problem on patterns and host graphs drawn in the unit sphere. In Plane Subdrawing Equivalence, the question is to find a subdrawing of a plane host graph G that is equivalent to the drawing of a plane pattern H. By Remark 1.3, the problem is equivalent to Planar Subgraph Isomorphism for 3-connected planar graphs. In Section 4 we carry over our results to all planar graphs. We first introduce some topological tools that we need for embedded dynamic programming. At every step of the dynamic programming, we compute every way how a combinatorial noose N corresponding to a middle set of the sc-decomposition $\langle T, \mu, \pi\rangle$ of G can intersect a subdrawing equivalent to the drawing of pattern H. Each intersection gives rise to a combinatorial noose of H. See Figure 1 for an illustration.

The running time of the algorithm crucially depends on the number of combinatorial nooses in H. The aim of this section is to prove the following:

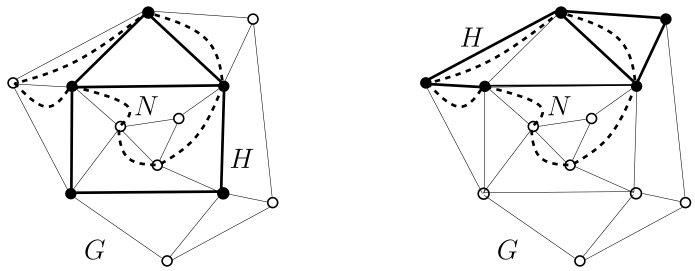

Figure 1: On the left, we draw graph G with an emphasized subdrawing H intersected by a combinatorial noose N indicated by dashed lines. On the right, we have the same graph G with a different copy of H intersected by N.

Theorem 3.1. *Let G be a plane graph on n vertices and H be a plane graph on $k \leq n$ vertices. We can decide if there is a subdrawing of G that is equivalent to the drawing of H in time $2^{O(k)}n$. We can find and count subdrawings equivalent to the drawing of H in time $2^{O(k)}n$, and enumerate ω subdrawings in time $2^{O(k)}n + O(\omega k)$.*

3.1. Combinatorial nooses in plane graphs

For a refined algorithm analysis we now take a close look at combinatorial nooses of plane graphs. In particular we are interested in counting the number of combinatorial nooses. In this subsection, we will prove the following proposition:

Proposition 3.2. *Every plane k-vertex graph has $2^{O(k)}$ combinatorial nooses.*

Before proving this proposition, we state that every combinatorial noose of a plane graph on k vertices corresponds to a cycle in some other plane graph on at most $O(k)$ vertices. The proofs of the following lemmas can be found in [10]. First we relate combinatorial nooses in a planar triangulation H to the cycles of H. Then we state that for any plane graph H there is an auxiliary graph H^*, such that the combinatorial nooses of H can be injectively mapped to the cycles of the triangulations of H^*. From Proposition 1.6 we know an upper bound on the number of cycles in planar graphs, which we employ to prove Proposition 3.2.

Lemma 3.3. *Let H be a planar triangulation and $N_C = [v_0, f_0, v_1, f_1, \dots, f_{\ell-1}, v_\ell]$ a combinatorial noose of H. Then for every pair of consecutive vertices v_i, v_{i+1} in N_C, there is a unique edge $\{v_i, v_{i+1}\}$ in $E(H)$. That is, the sequence $[v_0, v_1, \dots, v_\ell]$ is a simple cycle in H if $|V(N_C)| > 2$, and if $|V(N_C)| = 2$, it corresponds to a single edge in H.*

For an edge $e = \{v, w\}$ of a graph H we *subdivide* e by adding a vertex u to $V(H)$ and replacing e by two new edges $e_1 = \{v, u\}$ and $e_2 = \{u, w\}$. In a drawing of H, we place point u in the middle of the drawing of e partitioning e into e_1 and e_2.

Lemma 3.4. *Let H be plane graph and $N_C = [v_0, f_0, v_1, f_1, \dots, f_{\ell-1}, v_\ell]$ a combinatorial noose of H with $|V(N_C)| > 2$. Let H^* be obtained by subdividing every edge in $E(H)$. There exists a planar triangulation H' of H^* such that $[v_0, v_1, \dots, v_\ell]$ is a cycle in H'.*

Proof of Proposition 3.2. If H is triangulated, we have with Lemma 3.3 that every combinatorial noose corresponds to a unique cycle in H. By Proposition 1.6, the number of cycles in H is bounded by $2^{1.53k}$. Since for every edge of a cycle in H, we have two choices for a combinatorial noose to visit an incident face, we get the overall upper bound of $2^{2.53k}$ on the number of combinatorial nooses. If H is plane, we have to count the triangulations of H^* (Lemma 3.4). By Proposition 1.7 and the comments below it, there are at most $2^{3.24n}$ non-isomorphic triangulations on n vertices. Let us denote this set of triangulated graphs by Φ. We note that H^* is a subgraph of some graph of Φ, say of all graphs in $\Phi_H \subseteq \Phi$ with $|\Phi_H| \geq 1$. Since every triangulated graph is 3-connected, we have with Theorem 1.1 that every graph H' in Φ_H has a unique drawing in Σ up to homeomorphism. The plane graph H^* is then a subdrawing of a drawing equivalent to an arbitrary planar drawing of H' in Σ. The number of triangulations times the number of combinatorial nooses in each triangulation is an upper bound on the number of combinatorial nooses in H^*. ∎

For embedded dynamic programming on a sc-decomposition $\langle T, \mu, \pi \rangle$, we can argue with Remark 1.4 that if H is a subdrawing of G, then noose N formed by the middle set $\mathrm{mid}(e)$ is a noose of H, too. Recalling Remark 1.5, the alternating sequence of vertices and faces of H visited by N forms a combinatorial noose N_C in H. This observation allows us to discuss the results from a combinatorial point of view without the underlying topological arguments. Instead of nooses we will refer to combinatorial nooses in the remaining section.

3.2. Embedded dynamic programming

In embedded dynamic programming, the basic difference to usual dynamic programming is that we do not check for every partial solution for a given problem if or how it lies in the graph processed so far. Instead, we check how the graph that we have processed so far is intersecting the entire solution, that is how the graph is *embedded* into our solution. For subdrawing equivalence, we are interested in how G is drawn in the plane pattern H up to homeomorphism. Each edge of an sc-decomposition tree T corresponds to a noose N of G. We will associate to N the list of all possible subgraphs of H that appear in the part of G bounded by N. Therefore, we will describe all possible ways H is intersected by N. The number of solutions we get is bounded by the number of combinatorial nooses in H we can map N onto. We describe the algorithm in what follows.

Dynamic programming. We root sc-decomposition $\langle T, \mu, \pi \rangle$ at some node $r \in V(T)$. For each edge $e \in T$, let L_e be the set of leaves of the subtree rooted at e. The subgraph G_e of G is induced by the edge set $\{\mu(v) \mid v \in L_e\}$. The vertices of $\mathrm{mid}(e)$ form a combinatorial noose N that separates G_e from the residual graph.

Assuming H is a subgraph of G, the basic idea of embedded dynamic programming is that we are interested in how the vertices of the combinatorial noose N are intersecting faces and vertices of H. Since every noose in G is a noose in H, we can map N to a combinatorial noose N^H of H, bounding (clockwise) a unique subgraph H_{sub} of H.

In each step of the algorithm, all solutions for a sub-problem in G_e are computed, namely all possibilities of how N is mapped onto a combinatorial noose N^H in H that separates H_{sub} from the rest of H, where $H_{sub} \subseteq H$ is isomorphic to subgraphs of G_e. For every middle set, we store this information in an array. It is updated in a bottom-up process starting at the leaves of $\langle T, \mu, \pi \rangle$. During this updating process it is guaranteed that the 'local' solutions for each subgraph associated with a middle set of the sc-decomposition are combined into a 'global' solution for the overall graph G.

Valid mappings. Let G be a plane graph with a rooted sc-decomposition $\langle T, \mu, \pi\rangle$ and let H be a plane pattern. For every middle set mid(e) of $\langle T, \mu, \pi\rangle$ let N be the associated combinatorial noose in G with face-vertex sequence of $F(N) \cup V(N)$. Let $\mathfrak{L}$ denote the set of all combinatorial nooses of H whose length is at most the length of N. We now want to map N order preserving to each $N^H \in \mathfrak{L}$. We map vertices of N to both vertices and faces of H. Therefore, we consider partitions of $V(N) = V_1(N)\dot{\cup}V_2(N)$ where vertices in $V_1(N)$ are mapped to vertices of $V(H)$ and vertices in $V_2(N)$ to faces of $F(H)$. We define a mapping $\gamma : V(N) \cup F(N) \to V(H) \cup F(H)$ relating N to the combinatorial nooses in $\mathfrak{L}$. For every $N^H \in \mathfrak{L}$ on faces and vertices of set $F(N^H) \cup V(N^H)$ and for every partition $V_1(N)\dot{\cup}V_2(N)$ of $V(N)$ mapping γ *is valid* if

a) γ restricted to $V_1(N)$ is a bijection to $V(N^H)$;

b) for every $v \in V_2(N)$ and $f \in F(N)$ we have $\gamma(v)$ and $\gamma(f)$ in $F(N^H)$;

c) for every $v_i \in V(N)$ and subsequence $[f_{i-1}, v_i, f_i]$ of N, face $\gamma(v_i)$ is equal to both $\gamma(f_{i-1})$ and $\gamma(f_i)$, and vertex $\gamma(v_i)$ is incident to both $\gamma(f_{i-1})$ and $\gamma(f_i)$;

d) for every pair $w_i, w_j \in V(N^H)$: if $\{w_i, w_j\} \in E(H)$ then $\{\gamma^{-1}(w_i), \gamma^{-1}(w_j)\} \in E(G)$.

Items *a*) and *b*) say where to map the faces and vertices of N to. Item *c*) (with *a*)) makes sure that if two vertices v_h, v_j in sequence $N = [\ldots, v_h, \underline{\ldots}, v_j, \ldots]$ are mapped to two vertices w_i, w_{i+1} that appear in sequence N^H as $[\ldots, w_i, f_i, w_{i+1}, \ldots]$ then every face and vertex inbetween v_h, v_j in sequence N (here underlined) is mapped to face f_i. Item *d*) rules out the invalid solutions, that is, we do not map a pair of vertices in G that have no edge in common to the endpoints of an edge in H. We do so because if H is a subgraph of G then an edge in H is an edge in G, too. For an illustration, see Figure 2.

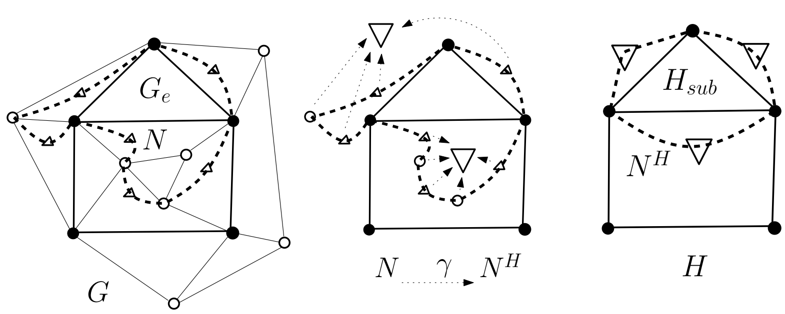

Figure 2: On the left, we have a plane graph G with a subgraph H emphasized. A combinatorial noose N separating subgraph G_e is indicated by dashed lines. The vertices of N are full and empty circles and the faces triangles. In the middle, we have H and indicate to which faces (big triangles) of H vertices and faces of N are mapped by γ. This gives us combinatorial noose N^H on the right, separating subgraph H_{sub}.

We assign an array A_e to each mid(e) consisting of all tuples $\langle N^H, \gamma_e\rangle$ each representing a valid mapping γ_e from combinatorial noose N corresponding to mid(e) to a combinatorial noose $N^H \in \mathfrak{L}$. The vertices and faces of N are oriented clockwise around the drawing of G_e. Without loss of generality, we assume for every $\langle N^H, \gamma_e\rangle \in A_e$ the orientation of N^H to be clockwise around the subdrawing H_{sub} of H equivalent to a subdrawing of G_e.

Step 0: Initializing the leaf edges. For each parent edge e_ℓ of a leaf ℓ of T we initialize the valid mappings from the combinatorial noose bounding the edge $\mu(\ell)$ of G to every combinatorial noose in H of length at most two.

Step 1: Update process. We update the arrays of the middle sets in post-order manner from the leaves of T to root r. In each dynamic programming step, we compare the arrays of two middle sets $\mathrm{mid}(e), \mathrm{mid}(f)$ in order to create a new array assigned to the middle set $\mathrm{mid}(g)$, where e, f and g have a vertex of T in common. From [12] we know about a special property of sc-decompositions: namely that the combinatorial noose N_g is formed by the symmetric difference of the combinatorial nooses N_e, N_f and that $G_g = G_e \cup G_f$. In other words, we are ensured that if two solutions on G_e and G_f bounded by N_e and N_f *fit together*, then they form a new solution on G_g bounded by N_g. We now determine when two solutions represented as tuples in the arrays A_e and A_f fit together. We update two tuples $\langle N_e^H, \gamma_e \rangle \in A_e$ and $\langle N_f^H, \gamma_f \rangle \in A_f$ to a new tuple in A_g if

- for every $x \in (\, V(N_e) \cup F(N_e)\,) \cap (\, V(N_f) \cup F(N_f)\,)$, we have $\gamma_e(x) = \gamma_f(x)$;
- for the subgraph H_e of H separated by N_e^H and the subgraph H_f of H separated by N_f^H, we have that $E(H_e) \cap E(H_f) = \emptyset$ and $V(H_e) \cap V(H_f) \subseteq \{\gamma(v) \mid v \in V(N_e) \cap V(N_f)\}$.

If N_e and N_f fit together, we get a valid mapping $\gamma_g : N_g \to N_g^H$ as follows:

- for every $x \in (\, V(N_e) \cup F(N_e)\,) \cap (\, V(N_f) \cup F(N_f)\,) \cap (\, V(N_g) \cup F(N_g),\,)$ we have $\gamma_e(x) = \gamma_f(x) = \gamma_g(x)$;
- for every $y \in (\, V(N_e) \cup F(N_e)\,) \setminus (\, V(N_f) \cup F(N_f)\,)$ we have $\gamma_e(y) = \gamma_g(y)$;
- for every $z \in (\, V(N_f) \cup F(N_f)\,) \setminus (\, V(N_e) \cup F(N_e)\,)$ we have $\gamma_f(z) = \gamma_g(z)$.

We have that γ_g is a valid mapping from N_g to the combinatorial noose N_g^H that bounds subgraph $H_g = H_e \cup H_f$. Thus, we add tuple $\langle N_g^H, \gamma_g \rangle$ to array A_g.

Step 2: End of DP. If, at some step, we have a solution where the entire subgraph H is formed, we exit the algorithm confirming. That is, if $H = H_e \cup H_f$ and H_i is bounded by N_i (for both $i \in \{e, f\}$) then the combinatorial noose N_g is bounding the subdrawing of G equivalent to the drawing of H. We output this subdrawing by reconstructing the solution top-down in $\langle T, \mu, \pi \rangle$. If at root r no subdrawing equivalent to the drawing of H has been found, we output 'FALSE'.

Correctness of DP. Let plane graph H be a subdrawing of G. We have already seen how to map every combinatorial noose of G that identifies a separation of G via a valid mapping γ to a combinatorial noose of H determining a separation of H. Step 0 ensures that every edge of H is bounded by a combinatorial noose N^H of length two, which is determined by tuple $\langle N^H, \gamma \rangle$ in an array assigned to a leaf edge of T. We need to show that Step 1 computes a valid solution for N_g from N_e and N_f for incident edges e, f, g. We note that the property that the symmetric difference of the combinatorial nooses N_e and N_f forms a new combinatorial noose N_g is passed on to the combinatorial nooses N_e^H, N_f^H and N_g^H of H, too. If the two solutions fit together, then H_e of H separated by N_e^H and subgraph H_f of H separated by N_f^H only intersect in the image of $V(N_e) \cap V(N_f)$. We may observe that N_e^H and N_f^H intersect in a continuous alternating subsequence with order reversed to each other, i.e., $N_e^H \mid_{N_e \cap N_f} = \overline{N_f^H} \mid_{N_e \cap N_f}$, where $\overline{N^H}$ means the reversed sequence N^H. Since every oriented N^H identifies uniquely a separation of $E(H)$, we can easily determine if two tuples $\langle N_e^H, \gamma_e \rangle \in A_e$ and $\langle N_f^H, \gamma_f \rangle \in A_f$ fit together and form a new subgraph of H. If H is a subdrawing of G, then at some step we will enter Step 2 and produce the entire H.

Running time analysis. We first give an upper bound on the size of each array. The number of combinatorial nooses in $\mathfrak{L}$ we are considering is bounded by the total number of combinatorial nooses in H, which is $2^{O(|V(H)|)}$ by Proposition 3.2. The number of partitions of vertices of any combinatorial noose N is bounded by $2^{|V(N)|}$. Since the order of both N^H and N is given we only have $2|V(H)|$ possibilities to map vertices of N to N^H, once the vertices of N are partitioned. Thus, in an array A_e we may have up to $2^{O(|V(H)|)} \cdot 2^{|V(N)|} \cdot |V(H)|$ tuples $\langle N_e^H, \gamma_e \rangle$. We first create all tuples in the arrays assigned to the leaves. Since middle sets of leaves only consist of an edge in G, we get arrays of size $O(|V(H)|^2)$ which we compute in the same asymptotic running time. When updating middle sets $\mathrm{mid}(e), \mathrm{mid}(f)$, we compare every tuple of one array A_e to every tuple in array A_f to check if two tuples fit together. We can compute the unique subgraph H_e (resp. H_f) described by a tuple in A_e (resp. A_f), compare two tuples in A_e, A_f and create a new tuple in A_g in time linear in the order of $V(N)$ and $V(H)$. Since the size of A_g is bounded by $2^{O(|V(H)|)} \cdot 2^{O(|V(N)|)}$, the update process for two middle sets takes the same asymptotic time. Assuming sc-decomposition $\langle T, \mu, \pi \rangle$ of G has width ω and $|V(H)| \leq \omega$, we get the following result.

Lemma 3.5. *For a plane graph G with a given sc-decomposition $\langle T, \mu, \pi \rangle$ of G of width w and a plane pattern H on $k \leq w$ vertices we can search for a subdrawing of G equivalent to H in time $2^{O(w)} \cdot n$.*

3.3. The algorithm

We present the overall algorithm for solving Plane Subdrawing Equivalence with running time stated in Theorem 3.1.

Algorithm 3.1: Plane Subdrawing Equivalence: PLSE.

Input : Plane graph G; Plane pattern H of order k.

1 Choose an arbitrary vertex v in G.
2 Partition $V(G)$ into $S_0 \cup S_1 \cup \ldots \cup S_\ell$ with $S_i = \{w \in V(G) : \mathrm{dist}(v, w) = i\}$
3 **for** *every* $G_i = G[S_i \cup \ldots \cup S_{i+k}]$ *with* $0 \leq i \leq \ell - k$ **do**
4   Compute sc-decomposition $\langle T, \mu, \pi \rangle$ of G_i.
5   Do embedded dynamic programming on $\langle T, \mu, \pi \rangle$ to find a subdrawing of G_i equivalent to the drawing of H and intersecting S_i.

Partitioning the vertex set in Line 2 of Algorithm 3.1 PLSE, is a similar approach to the well-known Baker-approach [2]. Every vertex set S_i contains the vertices of distance i to the chosen vertex v. $S_0 = \{v\}$ and ℓ is the maximum distance in G from v. The graph G_i in Line 3 is induced by the sets $S_i, \ldots, S_{i+k}$. As in [14], we may argue that every vertex in G appears in at most k subgraphs G_i. This keeps our running time linear in n. We can apply Lemma 2.1 to each G_i in Line 4 to a compute sc-decomposition $\langle T, \mu, \pi \rangle$ of width $\leq 2k + 1$, by adding a root vertex r for the BFS tree and make r adjacent to every vertex in S_i. The dynamic programming approach can easily be turned into an algorithm counting subdrawing equivalences (similar to [14]), by using a counter in the dynamic programming. Using an inductive argument, for every subgraphs G_i in Line 5 we only compute subgraphs intersecting with vertices in S_i and thus omit double-counting. We can adopt our technique to list the subdrawings of G equivalent to the drawing of H.

4. Planar subgraph isomorphism

Now we consider the case when both pattern H and host graph G are planar but not plane. From Remark 1.3 we know that two isomorphic planar graphs must not need to come with equivalent drawings. However, we observe that if H is isomorphic to a subgraph of G, then for every planar drawing of G there exists a drawing of H that is equivalent to a subdrawing of G. Hence, we may simply draw G planarly, and run the algorithm of the previous section for all non-equivalent drawings of H.

Algorithm 4.1: Planar subgraph isomorphism.

Input : Planar graph G, Planar pattern H of size k.
Compute a planar drawing of G.
if *H 3-connected* **then** Return PLSE(G, H).
for *every non-equivalent drawing I of H* **do**
  Return PLSE(G, I).

The whole algorithm. We compute in Algorithm 4.1 every non-equivalent drawing of H as follows. First, we compute the set $\mathcal{H}$ of non-isomorphic maximal planar graphs in time proportional to its size using the algorithm in [20]. For every graph $H' \in \mathcal{H}$ and every subdrawing I of H' we check whether I is isomorphic to H by using the linear time algorithm for planar graph isomorphism in [19][1] . By Proposition 1.2, we then call Algorithm 3.1 $2^{O(k)}$ times, for each plane graph I isomorphic to H. This ensures us that Algorithm 3.1 has running time as stated in Theorem 0.1 [2].

Conclusion

We have shown how to use topological graph theory to improve the results on the already mentioned variations of Planar Subgraph Isomorphism, solving the open problems posed in [14] and [12]. With the results of [15], [14] extends the feasible graph class from planar graphs to apex-minor-free graphs. This cannot be done with the tools presented here. However, the authors of [11] devise a truly subexponential algorithm for k-Longest Path in H-minor-free graphs and thus apex-minor-free graphs, employing the structural theorem of Robertson and Seymour [25] and the results of [8, 5]. Can the structure of H-minor-free graphs, be exploited for our purposes?

It seems unlikely that our work can be extended to obtain a subexponential algorithm. The first reason, mentioned in the introduction, is that Bidimensionality applies to subgraphs with minor properties rather than to general subgraphs. Secondly, our enumerative bounds are either tight or of lower bound $2^{\Omega(k)}$. We want to pose the open problem: Is Plane Subdrawing Equivalence solvable in time $2^{o(k)}n^{O(1)}$?

Acknowledgments. The author thanks Paul Bonsma, Holger Dell and Fedor Fomin for discussions and comments of great value to the presentation of these results.

[1] We get a list of drawings of H, from which we can delete equivalent drawings by a modification of the algorithm in [19]—namely isomorphism test for face-vertex graphs.

[2] It can be show that Algorithm 3.1 runs in time $O(2^{12.57k}n)$ and Algorithm 4.1 in $O(2^{18.81k}n)$

References

[1] N. Alon, R. Yuster, and U. Zwick, *Color-coding*, J. Assoc. Comput. Mach., 42 (1995), pp. 844–856.

[2] B. S. Baker, *Approximation algorithms for NP-complete problems on planar graphs*, J. Assoc. Comput. Mach., 41 (1994), pp. 153–180.

[3] H. L. Bodlaender, *A linear-time algorithm for finding tree-decompositions of small treewidth*, SIAM J. Comput., 25 (1996), pp. 1305–1317.

[4] K. Buchin, C. Knauer, K. Kriegel, A. Schulz, and R. Seidel, *On the number of cycles in planar graphs*, in Proc. of the 13th Annual International Conference on Computing and Combinatorics (COCOON'07), vol. 4598 of LNCS, Springer, 2007, pp. 97–107.

[5] A. Dawar, M. Grohe, and S. Kreutzer, *Locally excluding a minor*, in Proc. of the 22nd IEEE Symposium on Logic in Computer Science (LICS 2007), IEEE Computer Society, 2007, pp. 270–279.

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

[7] E. D. Demaine and M. T. Hajiaghayi, *The bidimensionality theory and its algorithmic applications*, Computer J., 51 (2008), pp. 292–302.

[8] E. D. Demaine, M. T. Hajiaghayi, and K. Kawarabayashi, *Algorithmic graph minor theory: Decomposition, approximation, and coloring*, in Proc. of the 46th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2005), IEEE Computer Society, 2005, pp. 637–646.

[9] R. Diestel, *Graph theory*, vol. 173 of Grad. Texts in Math., Springer, New York, third ed., 2000.

[10] F. Dorn, *Planar Subgraph Isomorphism Revisited*, http://arxiv.org/abs/0909.4692, 2009.

[11] F. Dorn, F. V. Fomin, and D. M. Thilikos, *Catalan structures and dynamic programming on H-minor-free graphs*, in Proc. of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2008), ACM, New York, 2008, pp. 631–640.

[12] F. Dorn, E. Penninkx, H. L. Bodlaender, and F. V. Fomin, *Efficient exact algorithms on planar graphs: Exploiting sphere cut decompositions*, Algorithmica, (2009, to appear).

[13] R. G. Downey and M. R. Fellows, *Parameterized complexity*, Springer, New York, 1999.

[14] D. Eppstein, *Subgraph isomorphism in planar graphs and related problems*, J. Graph Alg. and Appl., 3 (1999), pp. 1–27.

[15] D. Eppstein, *Diameter and treewidth in minor-closed graph families*, Algorithmica, (2009).

[16] J. Flum and M. Grohe, *Parameterized Complexity Theory*, Texts in Theoretical Computer Science. EATCS Series, Springer, Berlin, 2006.

[17] F. V. Fomin and D. M. Thilikos, *New upper bounds on the decomposability of planar graphs*, J. Graph Theory, 51 (2006), pp. 53–81.

[18] M. R. Garey and D. S. Johnson, *Computers and Intractability, A Guide to the Theory of NP-Completeness*, W.H. Freeman and Company, New York, 1979.

[19] J. E. Hopcroft and J. K. Wong, *Linear time algorithm for isomorphism of planar graphs (preliminary report)*, in Proc. of the Sixth Annual ACM Symposium on Theory of Computing (STOC'74), ACM, 1974, pp. 172–184.

[20] Z. Li and S.-I. Nakano, *Efficient generation of plane triangulations without repetitions*, in Proc. of the 28th International Colloquium on Automata, Languages and Programming (ICALP'01), vol. 2076 of LNCS, Springer, 2001, pp. 433–443.

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

[22] D. Osthus, H. J. Prömel, and A. Taraz, *On random planar graphs, the number of planar graphs and their triangulations*, J. Combin. Theory Ser. B, 88(1) (2003), pp. 119–134.

[23] J. Plehn and B. Voigt, *Finding minimally weighted subgraphs*, in Proc. of the 16th International Workshop on Graph-Theoretic Concepts in Computer Science (WG'90), vol. 484 of LNCS, Springer, 1990, pp. 18–29.

[24] N. Robertson, P. Seymour, and R. Thomas, *Quickly excluding a planar graph*, J. Combin. Theory Ser. B, 62 (1994), pp. 323–348.

[25] N. Robertson and P. D. Seymour, *Graph minors. XVI. Excluding a non-planar graph*, J. Combin. Theory Ser. B, 89 (2003), pp. 43–76.

[26] H. Tamaki, *A linear time heuristic for the branch-decomposition of planar graphs*, in Proc. of the 11th Annual European Symposium on Algorithms (ESA'03), vol. 2832 of LNCS, Springer, 2003, pp. 765–775.

[27] W. T. Tutte, *A census of planar triangulations*, Canad. J. Math., 14 (1962), pp. 21–38.