Parallel Planar Subgraph Isomorphism and Vertex Connectivity

Lukas Gianinazzi
Department of Computer Science
ETH Zurich
lukas.gianinazzi@inf.ethz.ch

Torsten Hoefler
Department of Computer Science
ETH Zurich
htor@inf.ethz.ch

ABSTRACT

We present the first parallel fixed-parameter algorithm for subgraph isomorphism in planar graphs, bounded-genus graphs, and, more generally, all minor-closed graphs of locally bounded treewidth. Our randomized low depth algorithm has a near-*linear* work dependency on the size of the target graph. Existing low depth algorithms do not guarantee that the work remains asymptotically the same for any constant-sized pattern. By using a connection to certain *separating cycles*, our subgraph isomorphism algorithm can decide the vertex connectivity of a planar graph (with high probability) in asymptotically near-linear work and poly-logarithmic depth. Previously, no sub-quadratic work and poly-logarithmic depth bound was known in planar graphs (in particular for distinguishing between four-connected and five-connected planar graphs).

CCS CONCEPTS

• **Theory of computation → Parallel algorithms**.

KEYWORDS

graph algorithms; parallel algorithms; subgraph isomorphism; planar graphs; vertex connectivity; parameterized complexity

1 INTRODUCTION

Subgraph Isomorphism has applications for pattern discovery in biological networks [3, 40, 46], graph databases [33], and electronic circuit design [44]. It is also powerful subroutine to solve edge connectivity and vertex connectivity of planar graphs [20]. The subgraph isomorphism problem is to look for occurrences of a *pattern graph* H as a subgraph of a *target graph* G. Subgraph isomorphism is a generalization of many *NP*-complete problems (such as finding a *Maximum Clique*, *Longest Path*, or *Hamiltonian Cycle* [26]). The problem remains hard even in bounded degree graphs [25] and planar graphs [45].

Hence, it is natural to consider *parameterized* versions of the problem that are *tractable* when some parameter is small. We focus our attention to the case when the pattern graph H is relatively small, and give algorithms whose work grows slowly (i.e., close to linear) with the size of the target graph G, but is allowed to grow quickly (i.e., exponential) in terms of the size of the pattern graph H. This continues the development of *fixed-parameter tractable* (FPT) algorithms for NP-hard problems [18].

We present a parallel *fixed-parameter tractable* algorithm with low depth for subgraph isomorphism in *planar graphs*. Planar graphs are an important class of graphs which arise naturally from problems in geometry [36], when trying to lay out electronic circuits without crossings [1], and in image segmentation [49].

Drawing on existing FPT techniques [4, 19, 49], our algorithm exploits that local neighborhoods of a planar graph are well-behaved and can be efficiently decomposed. We overcome two fundamental challenges: The first challenge is the reliance on a breadth-first-search (of unbounded depth) to construct the local neighborhoods. We avoid this issue by applying a randomized clustering [37] into low-diameter parts. This decomposition works because we can bound the probability that an occurrence of the pattern is not in a single cluster by a constant. The second challenge is the work-efficient solution of a high depth dynamic program. We transform the problem into a directed acyclic graph and exploit the properties of the parametrized subgraph isomorphism problem to show that introducing shortcuts for only a small subset of nodes suffices to reduce the depth of the graph to poly-logarithmic in the target graph's size (and linear in the pattern graph's size).

1.1 Preliminaries

Subgraph isomorphism is interested in *occurrences* of a *graph pattern* H (with k vertices and diameter d) as a subgraph of a *target graph* G (with n vertices). Formally, a subgraph isomorphism is an injective map ϕ from the vertices of H to the vertices of G such that if two vertices u and v are adjacent in H, then $\phi(u)$ and $\phi(v)$ are adjacent in G. The simplest variant of the subgraph isomorphism problem is to *decide* if any occurrence of the pattern exists in the target graph, but we can also consider *counting* the occurrences or *listing* them.

For any graph G', we denote its vertex set as $V(G')$, its edge set as $E(G')$, and the subgraph of G' induced by a subset X of its vertices by $G'[X]$. A graph that is formed from the graph G by contracting edges, deleting vertices, and deleting edges is a *minor* of G. A family of graphs is *minor-closed* if every minor of every graph in the family is also in the family.

Vertex Connectivity. A graph with at least $c+1$ vertices is c-vertex-connected if removing any $c-1$ vertices does not disconnect the graph. The vertex connectivity c of a graph is the largest number c for which the graph is c-vertex-connected.

Tree Decomposition and Treewidth. A *tree decomposition* provides a recursive subdivision of a graph into overlapping subgraphs such that each subgraph is disconnected from the rest of the graph after removing few vertices. The *decomposition tree* records the recursive subdivision in a tree and labels the nodes of the tree with the vertices used to subdivide the graph (in a way that every edge occurs in *at least* one of the tree nodes). See Figure 1 for an example of how a decomposition tree represents a recursive subdivision of a graph.

The advantage of the tree decomposition is that it gives a way to describe a divide-and-conquer approach (along some graph decomposition) as a dynamic program on the decomposition tree instead. The dynamic program maintains *partial results* that correspond to the subgraphs of the current node in the decomposition tree and combines the partial results in a bottom-up fashion on this tree.

Formally, a *tree decomposition* [8–11, 23, 34] of a graph G consists of a nonempty *decomposition tree* $\mathcal{T}$ where each node X_i of the tree $\mathcal{T}$ is a subset of the vertices $X_i \subseteq V$ of G, such that:

- Every vertex u of G is contained in a contiguous nonempty subtree of the decomposition tree $\mathcal{T}$.
- For every edge (u, v) of the graph G, there is a node X_i of the tree $\mathcal{T}$ where both endpoints u and v are in the node X_i.

The maximum of $|X_i| - 1$ over all nodes X_i of the tree $\mathcal{T}$ is the *width* of the tree decomposition. The smallest width of any tree decomposition of G is the *treewidth* τ of G.

We can assume for simplicity that every interior node in the decomposition tree has exactly two children, as we can split high-degree nodes and add empty leaf nodes without changing the width of the decomposition. Moreover, a minimum width tree decomposition of a graph with m edges has $O(m)$ nodes.

Model of Computation. We consider a synchronous *shared memory* parallel machine with concurrent reads and exclusive writes (CREW PRAM). We express our bounds in terms of the total number of operations performed by any execution of the algorithm by all processors (called *work*) and the length of the critical path in the computation (called *depth*) [6]. By Brent's scheduling algorithm [6, 47], an algorithm with work W and depth D can be executed with P processors in time $O(W/P + D)$ on a CREW PRAM.

Randomization. Numerous efficient parallel algorithms make use of some form of randomness [28]. For some graph problems (such as minimum cuts [28] and minimum spanning trees [15]), a randomized algorithm has the lowest known bounds.

We assume each processor has access to an independent and uniformly distributed random word in each time step. If an event occurs with probability at least $1 - n^{-a}$ for all constants $a > 1$, we say it occurs *with high probability* (w.h.p.). An algorithm that returns the correct result with high probability is *Monte Carlo*.

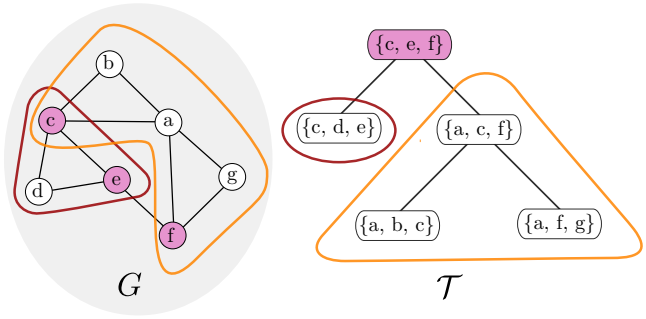

Figure 1: Illustration of a graph G and one of its tree decompositions $\mathcal{T}$ of width 2. The highlighted subtrees in the tree $\mathcal{T}$ correspond to the subgraphs highlighted in the graph G of the same color. The root node $\{c, e, f\}$ separates the two highlighted subtrees, meaning that every path from the subgraph induced by the left subtree to the subgraph induced by the right subtree contains a vertex that is in the root node $\{c, e, f\}$.

Table 1: Bounds for deciding planar subgraph isomorphism. (⋆) The algorithm is Monte Carlo, and its bounds hold w.h.p..

	Work	Depth
Alon et al.⋆ [2]	$e^k\, n^{\Theta(\sqrt{k})} \log n$	$\Theta(k \log n)$
Eppstein [19]	$O(2^{3k \log_2(3k+1)} n)$	$\Theta(kn)$
Dorn [17]	$O(2^{18.81k} n)$	$O(2^{18.81k} n)$
Fomin et al. ⋆ [22]	$2^{O(k/\log k)} n^{O(1)}$	$2^{O(k/\log k)} n^{O(1)}$
This Paper ⋆	$O(2^{3k \log_2(3k+1)}\ n \log n)$	$O(k \log^2 n)$

1.2 Related Work

For the general case of subgraph isomorphism, no algorithm with less work than the naive n^k is known. Ullmann presents an algorithm that uses a backtracking search [51].

Tree patterns of bounded size can be found efficiently in general graphs [2]. Much attention has been put on subgraph isomorphism in special *families of target graphs*, which require some form of sparsity and additional structure [2, 14, 19, 21].

Parameterized Complexity. The idea behind *parameterized complexity* [18] is to identify (one or more) fundamental parameters p of an NP-hard problem that characterize the difficult part of the problem. Then, a *fixed-parameter tractable algorithm*'s runtime separable into $f(p)g(n)$ (or $f(p) + g(n)$) where f is allowed to be any function of p and g has to be polynomial in n [18].

FPT Algorithms with low depth exist for several NP-complete problems [5, 7, 13]. Refer to Table 1 (excl. row 1) for an overview of the FPT algorithms for subgraph isomorphism in planar graphs.

Color Coding. Using a Monte Carlo technique called *Color Coding*, Alon et al. [2] obtain $O(e^k n^{\tau+1} \log n)$ work on a pattern of treewidth τ, which implies $e^k n^{\Theta(\sqrt{k})} \log n$ work for a planar pattern (as the treewidth of a planar graph with k vertices is $\Theta(\sqrt{k})$ [35, 48]). The algorithm's depth is poly-logarithmic in n and polynomial in k. Their key idea is to color the vertices in the target graph with k random colors, which allows a dynamic programming approach that needs to keep an exponentially smaller state. Note that this algorithm is *not* FPT for the size k of the pattern (nor the treewidth τ), because its runtime grows with $n^{\sqrt{k}}$ (or $n^{\tau+1}$).

Locally Bounded Treewidth. Eppstein presents the first FPT subgraph isomorphism algorithm for planar graphs that has a *linear* dependency on the size of the pattern graph [19]. It runs in polynomial time in n for patterns of size $O(\log n / \log\log n)$. The key insight is to exploit that local neighborhoods of planar graphs have bounded treewidth. Their algorithm generalizes to other minor-closed families with a relationship between diameter and treewidth, such as bounded-genus graphs [20]. They use a breadth-first-search (BFS) to decompose the graph into these local neighborhoods.

Sampling. Fomin et al. [22] present a randomized sampling approach that produces subgraphs of sub-linear treewidth in k. Then, they apply an existing FPT dynamic program.

1.3 Our Contributions

We present the first FPT work planar subgraph isomorphism algorithm with depth poly-logarithmic in n and polynomial in k. Our Monte Carlo algorithm has $k^{O(k)}n \log n$ work and $O(k \log^2 n)$ depth in planar graphs and has FPT work in all minor-closed families of graphs of locally bounded treewidth (see Section 4.3).

Table 1 contains the exact bounds and a comparison to the related works regarding planar graphs. Note that if the pattern graph occurs in the target graph, the expected work is $k^{O(k)}n$. Our algorithm can also *list* all x occurrences of a pattern with $O(xk\ (\log n + \log x)) + k^{O(k)}n\ (\log n + \log x)$ work and $O(k \log^2 n\ (\log n + \log x))$ depth.

We use a *low-diameter decomposition*, which can ensure that the occurrences of the pattern graph are in the same low-diameter part of the graph with sufficient probability. Then, we show how to exploit the special structure of a tree decomposition based algorithm to compute its results work-efficiently in parallel. Finally, we provide a randomized extension to the algorithm that also handles disconnected pattern graphs.

More generally, we can find isomorphic subgraphs that separate a set of marked vertices (leaving them in different components after removal of the subgraph). Because there is a relation between finding certain separating cycles as subgraphs and planar vertex connectivity, our subgraph isomorphism algorithm yields better parallel bounds for deciding vertex connectivity in planar graphs.

We show that planar vertex connectivity can be answered in $O(n \log n)$ work and $O(\log^2 n)$ depth. Previously, only 2-connectivity and 3-connectivity had sub-quadratic work and poly-logarithmic depth solutions [38, 50].

2 FROM PLANAR TO LOW TREEWIDTH

Planar graphs do not have bounded treewidth (it can be up to $\sqrt{n}$), which prevents a direct application of bounded treewidth techniques (as we use in Section 3). Fortunately, a planar graph of diameter d has treewidth at most $3d$ [19], and each occurrence of a pattern with diameter d is contained in a subgraph of diameter d of the target graph.

Hence, a simple (but work-inefficient) approach to solve subgraph isomorphism in planar graphs would consist of building for every vertex in the target graph the subgraph induced by nodes at a distance at most d, and then invoking an algorithm for bounded treewidth graphs on each of those subgraphs. This approach of *covering* the graph is inefficient because many vertices of the target graph could be in multiple (even all) of these subgraphs, leading to a total size of these subgraphs of $\Theta(n^2)$.

Instead, Eppstein [19] proposed (based on an idea by Baker [4]) a covering approach based on a single BFS to cover all subgraphs of diameter at most d with graphs of total size only $O(dn)$. It is easy to see that naive BFS takes linear work and $O(D)$ depth on a diameter D graph, but we care exactly about the situation when the diameter D is not bounded. Even on planar graphs, performing work-efficient and low-depth BFS is a challenging problem. An approach by Klein [32] achieves $O(n \log^9 n)$ work and poly-logarithmic depth.

To avoid the issue of low-depth BFS, in our approach, we first decompose the graph into *randomized* clusters of small diameter (as illustrated in Figure 2 and Figure 3). This allows us to then run

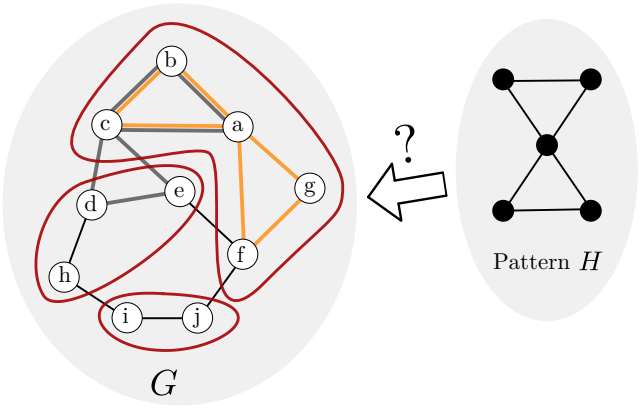

Figure 2: A randomized procedure splits the target graph G into clusters. Each occurrence of the pattern H is contained inside a single cluster with constant probability. In the example, the occurrence of the pattern H with vertices f, g, a, b, c is contained in a single cluster, but the occurrence d, e, a, b, c crosses the clusters. Hence, the former is found with this clustering, but the latter is not.

a simple parallel BFS on those low diameter graphs and construct a covering for each of those clusters. In summary, one run of our subgraph isomorphism algorithm works as follows:

(1) Cover the target graph with subgraphs $G_0, \ldots, G_i$ of bounded treewidth (they might overlap, as detailed in Section 2.1).
(2) Solve subgraph isomorphism for each such bounded treewidth subgraph in parallel (as described in Section 3).

Since our covering algorithm is randomized, an occurrence of a pattern may not be contained in any single subgraph in the cover. However, in expectation $O(1)$ repetitions suffice to find an occurrence of the pattern if it exists. At most $O(\log n)$ runs suffice to certify that no occurrence of a pattern exists with high probability. Our main result for planar graphs is the following:

Theorem 2.1. *Deciding (with high probability) if a connected pattern graph H occurs as a subgraph of a planar target graph G takes $O((3k)^{3k+1}n \log n)$ work and $O(k \log^2 n)$ depth.*

For a pattern of small diameter d, we obtain better bounds:

Corollary 2.2. *Deciding (with high probability) if a connected pattern graph H of diameter d occurs as a subgraph of a planar graph G takes $O((3d+3)^{3k+1}n \log n)$ work and $O(k \log^2 n)$ depth.*

To simplify the exposition, we assume (for now) that the pattern graph is connected and focus on the decision version of the problem. We then discuss how to remove the assumption of connectedness in Section 4.1 and show how to modify the algorithm to list all occurrences of a pattern graph in Section 4.2. Moreover, we generalize the approach from planar graphs to a class of graphs that contains all bounded-genus graphs in Section 4.3.

2.1 Parallel Low-Treewidth Cover

We show how to construct (in parallel) a set of subgraphs of low treewidth such that each occurrence of a connected pattern H is in at least one of the subgraphs with constant probability. The first step is to use a *low-diameter decomposition*. The goal of a low-diameter decomposition is to partition the vertices of the graph into (vertex-disjoint) clusters of low diameter such that few edges of the graph connect vertices in different clusters.

Exponential Start Time Clustering [37] is especially well-suited for our purposes because it bounds the *probability* that an edge connects two different clusters. This observation allows us to bound the probability that a connected subgraph is split into multiple clusters, and thus the clustering preserves the occurrences of a graph pattern with nontrivial probability, as needed for our purposes.

A *clustering* of G is a set of vertex-disjoint induced subgraphs called *clusters* that together contain all vertices. We say an edge *crosses the clusters* if it has endpoints in the vertex sets of two distinct clusters.

Lemma 2.3 (Exponential Start Time Clustering [37]). *With $O(n)$ work and $O(\beta \log n)$ depth, Exponential Start Time β-Clustering produces, w.h.p., clusters of diameter $O(\beta \log n)$ where each edge crosses the clusters with probability at most $1/\beta$.*

Note that Exponential Start-Time Clustering does not allow us to fix the number of clusters a priori. Instead, the number of clusters depends on the structure of the graph. For example, a clique will most likely end up as a single low-diameter cluster.

Because every edge crosses the clusters with small probability, the probability that a fixed occurrence of the pattern contains an edge that crosses the clusters is also relatively small (for an appropriate choice of parameter β). See Figure 2 for an illustration.

Observation 1. *The probability that no edge of a connected subgraph H of the graph G crosses a cluster of an Exponential Start Time $2k$-Clustering of G is at least $1/2$*

Proof. The idea is that some spanning tree of the occurrence remains intact (i.e., no edge in the tree crosses a cluster) with the given probability, which implies the result. Consider an arbitrary spanning tree A of H. By Lemma 2.3, the probability that a particular edge of the spanning tree crosses the clusters is at most $\frac{1}{2k}$. By the union bound, the probability that any of the $k-1$ edges of the spanning tree A crosses the clusters is at most $\frac{k-1}{2k} < \frac{1}{2}$. Hence, the probability that no edge crosses the clusters is at least $1/2$. □

We combine the clustering idea with the approach from Eppstein [19] and Baker [4] for the *Parallel treewidth k-d cover* algorithm.

Parallel Treewidth k-d-Cover.

(1) Run Exponential Start Time $2k$-Clustering on G.
(2) For each cluster, choose an arbitrary root v and run a naive parallel BFS within the cluster.
(3) This yields a BFS tree for each cluster. For each level i of the tree, output the subgraph G_i induced by the vertices at distance i through $i+d$ from v (as illustrated in Figure 3).

The algorithm guarantees that each of the subgraphs has low treewidth and that every occurrence of the pattern graph is in at least one of the subgraphs with constant probability:

Theorem 2.4. *For a planar target graph G and a connected pattern graph H with k vertices and diameter d, a Parallel Treewidth k-d Cover produces a set of induced subgraphs G_i of G such that:*

- *Every graph G_i has treewidth at most $3d$.*
- *Every vertex of G is contained in at most d graphs G_i.*
- *Every fixed occurrence of H is contained in at least one of the graphs G_i with probability at least $1/2$.*

The algorithm takes, w.h.p., $O(nd)$ work and $O(k \log n)$ depth.

Proof. Each of the graphs G_i is a subgraph of a planar graph with diameter d. Hence, it has treewidth at most $3d$ [19]. By Observation 1, an occurrence H' of H is in the same cluster with probability at least $1/2$. If this is the case, consider the first vertex u of the pattern occurrence H' encountered during the BFS done for the cluster and let i be the distance of u from the root v of the BFS tree. Then, the occurrence H' is an induced subgraph of G_i.

The clusters have diameter $O(k \log n)$. Hence, the BFSes have $O(k \log n)$ depth. Each vertex and edge is part of at most d subgraphs by construction, which implies that the work is $O(nd)$. □

It remains to find (in parallel) occurrences of the pattern on each of the low treewidth subgraphs we constructed. The algorithm in Section 3 requires that a tree decomposition of the subgraph has already been computed. For a planar graph, constructing such a decomposition of width $3d$ takes $O(n)$ work and $O(d)$ depth given a planar embedding of the graph [4, 19]. Computing a planar embedding takes $O(n)$ work and $O(\log^2 n)$ depth [31].

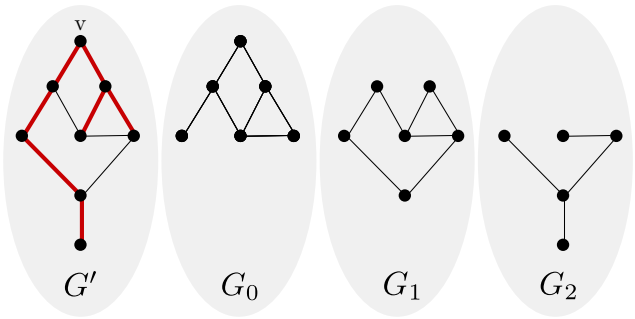

Figure 3: A cluster G' is covered by a set of subgraphs that are each induced by d consecutive levels in a BFS tree of the cluster. In the example, $d = 2$. The bold BFS tree of the graph G' rooted at v guides the covering of the graph with the induced subgraphs G_0, G_1, G_2. If the original graph contains a pattern of diameter d, then at least one of the subgraphs does as well. Note that the graphs G_3 and G_4 are not needed because their diameter is smaller than the pattern's diameter.

3 ALGORITHM FOR BOUNDED TREEWIDTH

The main result of this section is a parallel algorithm to solve subgraph isomorphism in parallel on graphs of bounded treewidth. It is based on a simplified version of the algorithm from Eppstein [19]. We transform the original problem into a graph search problem. Exploiting the particular structure of the resulting acyclic graph allows us a low depth and work-efficient solution.

Lemma 3.1. *Deciding if a connected pattern graph H is isomorphic to a subgraph of the target graph G of treewidth τ takes $O(k \log^2 n)$ depth and $O((\tau+3)^{3k+1} n)$ work. The bounds hold w.h.p..*

The overall idea of the sequential algorithm is to gradually compute the subgraph isomorphism while traversing the decomposition tree in a bottom-up fashion. We start by discussing the *partial matches* (partially completed subgraph isomorphisms) the algorithm employs, which are crucial for the parallel algorithm as well.

3.1 Partial Matches

Every node X in the decomposition tree corresponds to a subgraph $G[X]$ induced by X in the target graph G with only a small number of vertices $\tau+1$. Moreover, the descendants of the node X (together with X) induce a subgraph G_X of the graph G that is separated from the rest of the graph G by the vertices in the tree decomposition node X. The idea of *partial matches* is to find occurrences of subpatterns of the pattern H within these subgraphs and combining them in a bottom-up fashion in the tree decomposition.

Partial matches exist between subgraphs of the pattern graph H and these induced subgraphs G_X. Because vertices that are in the subgraph G_X but are not in the separating set X are not directly connected to the rest of the graph G, it is not necessary to explicitly store the mapping between pattern and target graph for these vertices in order to combine a partial match inside this subgraph with partial matches from the rest of the graph. Hence, when we build partial matches, only the τ^k different mappings for these vertices in the separating set X are important. The remaining vertices that have already been matched in a child are recorded as such. See Figure 4 for an example.

Formally, a partial match of X is a triple (ϕ, C, U), where C denotes the set of vertices *matched in a child*, U the set of vertices marked as *unmatched*, and a subgraph isomorphism function ϕ from the subgraph $H[V(H)\backslash(C \cup U)]$ to the subgraph $G[X]$. If a vertex v of H is *matched in a child*, the vertex v is mapped to a vertex in G_X which does not appear in X. If a vertex v of H is *unmatched*, then it is not matched to any vertex that appears in the subgraph G_X.

3.2 Eppstein's Sequential Algorithm

The idea is to extend the *partial matches* while traversing the decomposition tree $\mathcal{T}$ bottom-up. The goal is to construct a partial match of the root node where no vertex is *unmatched*. We focus on how to construct such a partial match for the root, from which a specific subgraph isomorphism can be recovered efficiently (by collecting appropriate partial isomorphisms in a top-down traversal of the tree; see also Section 4.2.1).

A partial match of a child node Y can be extended when going to a parent node X by matching some additional vertices that were

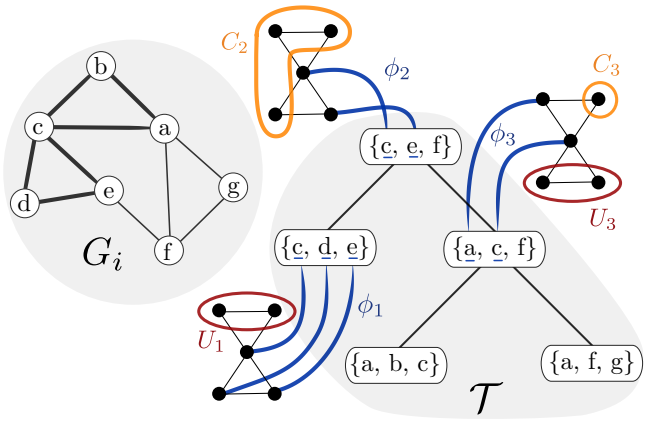

Figure 4: A valid partial match of the root $\{c, e, f\}$ of the decomposition tree is built from two compatible (and valid) partial matches of the left child $\{c, d, e\}$ and the right child $\{a, c, f\}$. Observe how every vertex in the pattern that is marked as '*matched in a child*' at the root is matched in exactly one of the two other partial matches. Also, note how the partial matches agree on the vertices that the decomposition nodes have in common: the partial match of the root agrees with the left child's match on c and e and with the right child's match on c.

unmatched by the child match to X, marking the vertices that have been matched by the child but are not in the parent as *matched in a child*, and leaving the rest of the partial isomorphism function the same (the vertices that were not newly matched in X remain unmatched).

A partial match that can be extended to a parent's partial match (possibly together with another child's partial match) is called *consistent* with a parent's partial match. The precise rules for being *consistent* follow. Consider a node X of the decomposition tree, one of its children Y, and the partial matches (ϕ_X, C_X, U_X) of X and (ϕ_Y, C_Y, U_Y) of Y. For all vertices v in H:

- If v is matched by ϕ_Y to a node in X *or* by ϕ_X to a node in Y, then they map to the same value: $\phi_X(v) = \phi_Y(v)$. This prevents the partial matches (ϕ_X, C_X, U_X) and (ϕ_Y, C_Y, U_Y) to map the same vertex in the pattern graph to different nodes in the target graph.
- If the child partial match ϕ_Y matches a vertex v to a vertex not in the parent label set X *or* marks the vertex v as *matched in a child* (i.e., in C_X), then the parent partial match marks the vertex v as *matched in a child* (i.e., in C_Y). In particular, we have $C_Y \subseteq C_X$.

Note that these rules imply that the child's partial match does not match any vertex that is unmatched by the parent, i.e., $U_Y \subseteq U_X$.

The point of the following combination rule is to ensure (on top of consistency) that a vertex that is marked as *matched in a child* in the parent is matched in exactly one of the children. A partial match M_X of node X is *compatible* with a partial match M_L of the left child L of X and partial match M_R of the right child R of N if the following conditions hold:

- The partial matches M_L and M_R are both consistent with the partial match M_X.
- If a vertex is marked as *matched in a child* by M_X, then it is marked as *unmatched* in exactly one of the child matches M_L and M_R.

A partial match is *valid*, if it is compatible with two partial matches of its children, or if it does not mark any vertices as *matched in a child*. Note that the trivial partial match that marks everything as *unmatched* is always valid. A valid partial match of the root node that does not mark any vertex as unmatched certifies the existence of a subgraph isomorphism.

The sequential algorithm traverses the decomposition tree bottom-up and enumerates all possible partial matches for the current node, then checks which are valid (given the valid matches for the children). For a tree decomposition of width τ and a pattern of size k, there are at most $(\tau+3)^k$ possible partial matches per node. There are at most $(\tau+3)^{3k}$ combinations of partial matches of the parent and its two children and validating a combination takes $O(\tau)$ time. Hence, the overall runtime is $O((\tau+3)^{3k+1}m)$.

3.3 Parallel Algorithm

The issue is that even a low-diameter planar graph might have a decomposition tree that has a large height of $\Omega(n)$. Therefore, parallelizing the computation at each node of the decomposition tree is not enough. It is possible to transform any tree decomposition into a decomposition of height $O(\log n)$ with *three times* the treewidth [10], which increases the work by a factor of $\Omega(9^k)$.

To avoid this, we parallelize across the height of the decomposition tree. In order to obtain a simpler problem, we partition the tree into paths. Then, we solve the problem on each of the paths. A path can be solved once all paths that start at a child of a node in the path have been solved. We avoid the sequential bottleneck by transforming the problem of finding valid partial matches in these subpaths of the tree decomposition into a reachability question in an acyclic directed graph with special structure. The reachability question can be solved work-efficiently with a low depth on this acyclic graph by introducing shortcuts of exponentially increasing distance to a carefully selected subset of the vertices.

3.3.1 Decomposition into Paths. Let us start by discussing how to decompose the tree into suitable subpaths. Walk from every leaf towards the root until reaching a branching node (i.e., a node with at least 2 children). Remove the visited paths from the tree, and proceed recursively. This decomposition can be implemented efficiently using parallel expression tree evaluation (tree contraction) [39, 47]:

Lemma 3.2 (Appendix A). *A tree T can be decomposed into a set of paths P where the paths are grouped into $O(\log n)$ layers with the property that vertices in the i-th layer have no children in a layer larger than i. This decomposition takes $O(n)$ work and $O(\log n)$ depth.*

3.3.2 The Graph of Partial Matches. We can reason about how to construct the valid partial matches for a subpath $\mathcal{P}$ of the tree decomposition, assuming we already solved all paths descending from a child of $\mathcal{P}$. Specifically, we derive locally at each node in the subpath $\mathcal{P}$ a set of partial matches that are valid partial matches *if at least* one of the partial matches of a child node of $\mathcal{P}$ is also a valid partial match. At the leaf of the path, we know which partial matches are valid (because both children have already been solved). This observation leads to the idea to construct a directed acyclic graph of partial matches where reachability models the validity of the partial matches, as follows.

Let $\mathcal{P}$ be a subpath of the tree decomposition $\mathcal{T}$. Consider a node X in the path $\mathcal{P}$ and assume we already computed the partial matches for the left child L of X (the other child is the right child R, where $R \in \mathcal{P}$). Then, we can check which partial matches of X and the right child of X are compatible with a partial match of L. This yields for every partial match M_X of X a set of partial matches of R that would validate the partial match M_X.

We construct a directed acyclic graph G' based on this idea. For the leaf node of $\mathcal{P}$, there is a vertex in G' for every valid partial match. For every other node X in $\mathcal{P}$, there is a vertex for every partial match of that node X. Then, there is an edge from a partial match M_R of the child R of X to a partial match M_X if there is a valid partial match M_L of the other child L of X such that M_X is compatible with M_L and M_R.

Reachability in the graph G' can model which partial matches are valid: A partial match is tagged as valid if it does not mark any vertices as *mapped by a child*. The partial matches from the leaf node of $\mathcal{P}$ are also tagged as valid. Then, the valid partial matches are those that are reachable from a partial match tagged as valid in the directed acyclic graph G'.

3.3.3 Finding Valid Partial Matches Via Reachability. Next, we discuss how to compute all the valid partial matches using the directed acyclic graph G'. Note that this graph G' still has a diameter equal to the length of the path $\mathcal{P}$, so we cannot directly use BFS. Hence, we introduce *shortcuts* of exponentially increasing distance to reduce the diameter to $O(k \log n)$. After introducing the shortcuts, we use naive parallel BFS to determine all the reachable vertices. The details follow.

A simple (but a factor $\log n$ work-inefficient) way to solve reachability is to introduce shortcuts for every vertex (similarly to some list ranking and connected components algorithms [47]):

(1) Introduce shortcuts in $\log n$ rounds $0, 1, \ldots, \log n$.
(2) Round i creates shortcuts of length 2^i. The edges of the graph are shortcuts of length 1.
(3) For round $i > 0$, for every vertex u, look at all its outgoing edges of length 2^{i-1}. For each such edge (u, v), look at all edges (v, w) of equal length 2^{i-1} and add an edge (u, w) of length 2^i to u.

This would result in $O(\log n)$ depth, but also be work inefficient by up to a factor $\Theta(\log n)$ (when $k = O(1)$) because *every* vertex in the graph G' does $\Omega(\log n)$ work.

The crucial observation to overcome this limitation is that any valid partial match is constructed by matching a "new" vertex at most k times. Thus, there are at most k edges in G' that match new vertices along any path in G' towards a valid partial match. The rest of the edges in G' do not introduce any new matches, but instead, translate from the partial match of a child to an equivalent partial match of the root. Since there is only one way not to introduce any new matches (see Figure 5), the subgraph of edges that do not introduce new edges is a directed forest (where edges are directed

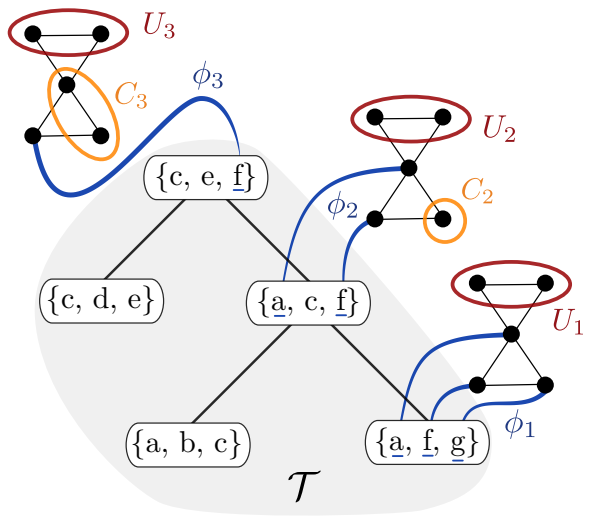

Figure 5: The valid partial match (ϕ_1, C_1, U_1) at node $\{a, f, g\}$ in the decomposition tree $\mathcal{T}$ can be turned into a valid partial match (ϕ_2, C_2, U_2) of the parent $\{a, c, f\}$ without matching a new vertex in exactly one way: The partial match has the same set of unmatched vertices $U_2 = U_1$. The set of children vertices contains the vertex that was matched to g, because g is not in $\{a, c, f\}$. The isomorphism function ϕ_2 is the same as ϕ_1 on all the vertices in $\{a, f, g\} \cap \{a, c, f\}$ and undefined elsewhere. The partial match (ϕ_2, C_2, U_2) is turned into (ϕ_3, C_3, U_3) similarly, except that now the set C_3 of vertices *matched in a child* also includes those in C_2

towards the roots). Hence, it suffices to introduce shortcuts in this forest F.

Because the subgraph F is a forest, shortcuts can be introduced work-efficiently in parallel: In each tree of F, decompose the tree into paths using Lemma 3.2. In each path, choose every $\log n$-*th* vertex as a vertex where shortcuts are introduced. Add a shortcut from every such vertex to the next, then add shortcuts of exponentially increasing distance between them (within the path). Moreover, add a shortcut from every vertex to the first vertex in a lower layer.

Lemma 3.3. *Computing the valid partial matches of the graph pattern H in a subpath $\mathcal{P}$ of a decomposition tree $\mathcal{T}$ of width τ takes $O(|\mathcal{P}|((\tau+3)^{3k+1}))$ work and $O(k \log n)$ depth.*

Proof. The work is linear in the number of vertices because we add the edges of exponentially increasing distances to a forest F of $O(|\mathcal{P}|/\log n)$ vertices.

After introducing the shortcuts, the distance from a valid leaf node to any other valid node is $O(k \log n)$: Consider any path p in the original graph G'. It contains at most k edges that are not in the forest F. Therefore, it consists of at most k subpaths $p_1, \ldots, p_k$ where each p_i is a subgraph of the forest F. Each subpath p_i is contained in a maximal tree F_i of F. By Lemma 3.2, p_i intersects at most $O(\log n)$ subpaths of F_i. It takes $O(\log n)$ hops to move from the first such subpath to the last (because of the shortcuts to a vertex in a lower layer). Then, it takes an additional $O(\log n)$ hops to traverse the first and last subpath using the shortcuts within each subpath. We conclude that the overall number of hops to traverse the path p is $O(k \log n)$.

Together with the depth of constructing the shortcut graph, this means that the depth of the algorithm is $O(k \log n)$. □

4 EXTENSIONS

We generalize our algorithm to disconnected patterns, show how to list all occurrences of a graph pattern, and characterize the family of graphs for which the algorithm is still FPT.

4.1 Disconnected Patterns

We extend our algorithm so that it can handle arbitrary disconnected patterns. These patterns are challenging because (in particular) the algorithm for treewidth k-cover cannot guarantee that every component of the pattern graph is in the same cluster.

Consider a pattern graph H consisting of l connected components. Number the components arbitrarily from 1 to l. A naive approach is to try out all l^n possible ways to split the target graph into l components. A randomized approach (inspired by *color coding* [2]) allows us to remove the exponential dependency on the number of vertices n. It works as follows:

(1) Color each vertex in G independently and uniformly at random with a number between 1 and c.
(2) For each color i, let G^i be the subgraph induced by the vertices that have color i.
(3) Search for occurrences of the i-th component of H in the subgraph G^i of color i vertices.
(4) Return true if and only if each search is successful.

Lemma 4.1. *Finding (with high probability) an occurrence of a disconnected pattern with l components and k vertices takes $O(l^k \log n)$ more work than finding an occurrence of a connected pattern.*

Proof. Consider a fixed occurrence of the pattern H. The probability that all of its vertices are assigned to the correct component of H is l^{-k}. Hence, $O(l^k)$ repetitions suffice to find a particular occurrence of H with constant probability, and $O(l^k \log n)$ repetitions suffice to certify that no occurrence exists with high probability. □

Note that this technique of finding disconnected patterns by reduction to the connected case is completely general and can be used in conjunction with any subgraph isomorphism algorithm.

4.2 Listing all Occurrences

We describe the modifications necessary to make our algorithm list all occurrences of a pattern. The first step is to modify the algorithm such that it returns a particular occurrence of a pattern with probability at least 1/2. Then, we can repeatedly generate a new set of occurrences, remove duplicates (by hashing), until we are confident enough that we have found all occurrences. The main difficulty is that the number of iterations necessary to find all the occurrences depends on the number of occurrences, which we do not know in advance.

However, since every *particular* occurrence is found with probability at least 1/2 in each iteration, if there is an occurrence that has not yet been found, at least one new occurrence is found with probability at least 1/2. This argument shows that the process is

related to getting many heads in a row when flipping coins: it is unlikely that many iterations in a row do not find a new occurrence.

Observation 2. *For all $j \leq i$, the probability that in a sequence of j independent coin flips i heads occur in a row is at most $j2^{-i}$.*

Proof. The probability that i heads occur in a row starting from the y-th coin flip is at most 2^{-i}. By a union bound over the j possible start positions, the bound follows. □

This observation still holds even for biased coins, as long as the probability that heads comes up is *at most* $1/2$.

Therefore, we iterate until after j iterations we have seen no new occurrence for $\log_2 j + \Theta(\log n)$ iterations in a row to guarantee that we have found all occurrences with high probability in n.

Theorem 4.2. *Listing w.h.p. all x occurrences of a connected pattern graph in a planar target graph takes $O(k \log^2 n\ (\log(x)+\log n)))$ depth and $O((xk + (3k+3)^{3k+1}n)\ (\log n + \log x))$ work.*

Proof. Every iteration finds a specific occurrence with probability at least $1/2$. Hence, after $\log_2 x + \Theta(\log n)$ iterations, the probability that we have *not* found a specific occurrence is at most $x^{-1}n^{-\Omega(1)}$. By a union bound over the x occurrences, the probability that we have not found *all* occurrences is at most $n^{-\Omega(1)}$. Hence, after $i = \log_2 x + \Theta(\log n)$ iterations, the algorithm will, with high probability, not find any new occurrences (because there are none) and by construction terminate after an additional $O(\log i + \log n)$ iterations. Overall, the algorithm takes at most $O(\log x + \log n)$ iterations to terminate with high probability. Together with the bounds from Section 4.2.1 this implies the work and depth bounds.

We show that the probability that the algorithm terminates before all occurrences have been found is at most $n^{-\Omega(1)}$. Consider the longest prefix of iterations of the algorithm where it has not found all occurrences. Model these iterations as coin flips, where the coin of an iteration turns up heads if this iteration finds no new occurrence. Heads comes up with probability *at most* $1/2$ because each such iteration finds a new occurrence with probability at least $1/2$. By Observation 2, the probability that (for any j in this sequence) after j coin flips heads comes up $\log_2 j + \Theta(\log n)$ times in a row is at most $n^{-\Omega(1)}$. This situation is the only one in which the algorithm terminates before finding all occurrences.

□

Hence, if we can find every occurrence that does not cross a cluster, we can find all occurrences with high probability. It remains to describe how to find these occurrences.

4.2.1 Recovering All Occurrences for a Cluster. Every valid partial match of the root of the tree decomposition that does not map any vertex as *unmatched* can be attributed to one or more subgraph isomorphisms. We construct these subgraph isomorphisms *top down* while traversing the shortcut graph of valid partial matches in reverse order (only following edges that lead to a valid partial match). The algorithm keeps a set of current subgraph isomorphisms at every vertex in the graph and does a parallel BFS of limited depth. When visiting a new vertex of the shortcut graph (which contains a partial mapping ϕ), every subgraph isomorphism in the list from the predecessor node is extended by ϕ and stored in the new vertex.

As for the decision problem, we observe that only k edges introduce a new vertex to the mapping. The other edges are shortcut so that overall at most $O(\log n)$ edges need to be traversed in between those k edges. However, we now need to construct the possible subgraph isomorphism even through those shortcuts explicitly. Fortunately, as illustrated in Figure 5, there is a unique way to extend a partial match through these shortcut edges, namely, do not change the current mapping at all. Hence, the overall depth of the reconstruction is $O(k \log^2 n)$.

By considering only occurrences that contain at least one vertex that is closest to the root of the BFS tree of the k-d cover, every traversed path leads to at least one subgraph isomorphism, and the work is bounded by the size of all the subgraph isomorphisms.

4.3 Bounded Genus & Apex-Minor-Free Graphs

Our results generalize to all (minor-closed) families of graphs where a bounded diameter graph has bounded treewidth. Observe that our treewidth k-cover algorithm from Section 2.1 does not use anything specific to planar graphs. It outputs subgraphs of diameter d that cover all occurrences of the pattern with constant probability. Moreover, our algorithm for bounded treewidth in Section 3 only requires a treewidth decomposition of low width. We start by giving the characterization of the graphs where our results hold and then discuss the few necessary changes.

4.3.1 Locally Bounded Treewidth. A family of graphs has *locally bounded treewidth* [20] if every graph of diameter D has treewidth at most $f(D)$, for some function f. Surprisingly, all minor-closed families of graphs that have locally bounded treewidth have *locally linear treewidth* [16], meaning that a graph of diameter D has treewidth $O(D)$.

The graphs of locally bounded treewidth have been characterized with respect to having certain *excluded minors*. A graph G that has a vertex v that is connected to all other vertices in G that becomes planar after removing v is an *apex-graph*. Such graphs do not have locally bounded treewidth. For example, consider the $n \times n$ grid with an additional vertex connected to all other vertices. This graph has diameter 2, but because the grid has treewidth n [48] this apex graph has treewidth at least n. Note that some apex graphs are planar (like the clique K_4) while others are not (like the clique K_5).

Interestingly, a minor-closed family of graphs of locally bounded treewidth must have an *apex graph as an excluded minor* [20] . For example, planar graphs exclude the apex graph K_5 as a minor (by Kuratowski's theorem [52]). Examples of apex-minor-free graphs include bounded-genus-graphs.

4.3.2 Parallel Tree Decomposition. The missing piece to our parallel subgraph isomorphism algorithm on apex-minor-free graphs is a parallel tree decomposition algorithm. The algorithm from Lagergren [34] achieves poly-logarithmic depth for constant treewidth, but the depth of the algorithm is not polynomial in τ. It becomes the bottleneck in our subgraph isomorphism algorithm.

Theorem 4.3 (Lagergren [34]). *For a graph with treewidth τ, computing a tree decomposition of width $8\tau + 7$ takes $\tau^{O(\tau)}m$ work and $\tau^{O(\tau)} \log^3 n$ depth.*

Together with the results from Section 2.1 and Section 3 this proves the generalized bounds. Similar results hold for disconnected patterns and listing all occurrences of the pattern.

Theorem 4.4. *Deciding (with high probability) if a connected pattern graph H occurs as a subgraph of an apex-minor-free graph G takes $k^{O(k)}n\log^3 n$ work and $k^{O(k)}\log^3 n$ depth.*

5 PLANAR VERTEX CONNECTIVITY

Vertex connectivity is a classic graph problem with applications in networking [12] and operations research [41]. Sequentially, c-vertex connectivity can be solved in linear time for planar graphs [19] and, more generally, in $O(c^2n^2\log n)$ time deterministically [30] and $O(m + c^{7/3}n^{4/3})$ time with high probability [42]. Recently, a sub-quadratic time deterministic algorithm [24] and a near-linear work [43] algorithm have been announced.

Two-connectivity and 3-connectivity have long been solved (optimally) for general graphs with linear work and logarithmic depth [38, 50]. In contrast, no sub-quadratic work poly-logarithmic depth 4-connectivity algorithm was available even for planar graphs prior to our work.

We show that vertex connectivity can be solved with $O(n\log n)$ work and $O(\log^2 n)$ depth in planar graphs. This result is possible because the vertex connectivity is closely related to certain separating cycles in a target graph that is constructed based on a planar embedding of the original graph (details below). Moreover, we use that the work of our subgraph isomorphism algorithm is $O(n\log n)$ *for any constant size pattern*. Eppstein [19] uses this idea (attributed to Nishizeki) for his *sequential* linear work vertex connectivity algorithm. We describe the approach and the necessary changes to our parallel algorithm.

5.1 From Connectivity to Separating Cycles

We show how to construct the target graph that we use to solve vertex connectivity, based on an idea attributed to Nishizeki [19]. See Figure 6 for an illustration.

Embed the graph G in the plane. Use the embedding to construct a *bipartite* target graph G' from G as follows. One side of the bipartite graph consists of the vertices from G. The vertices on this side are the *original vertices*. The other side has a vertex f for each face f in the original graph G. The vertices on this side are the *face vertices*. A face vertex f of G' and an original vertex v of G' are connected if and only if the face f contains the vertex v in the graph G. Observe that because the graph G' is bipartite, all its cycles have even length.

Separating Subgraphs. A subgraph H' of a graph G *separates the vertex set* $S \subseteq V(G)$ if the graph $G[V(G)\backslash V(H')]$ we get from removing all vertices of H' from G contains at least two vertices from S in two different connected components.

Lemma 5.1 (Nishizeki / Eppstein [19]). *If G is 2-connected and the shortest cycle in the bipartite graph G' that separates the set of original vertices has length $2c$, then G has vertex connectivity c.*

This leads us to our algorithm to decide planar vertex connectivity in parallel. First, check if the graph is 2-connected and if it is 3-connected using existing algorithms [38, 50]. If the graph is 3-connected, check if there is a cycle of length 8 in G' that separates the original vertices of G'. If so, the graph G has vertex connectivity 4. Otherwise, the graph G has vertex connectivity 5.

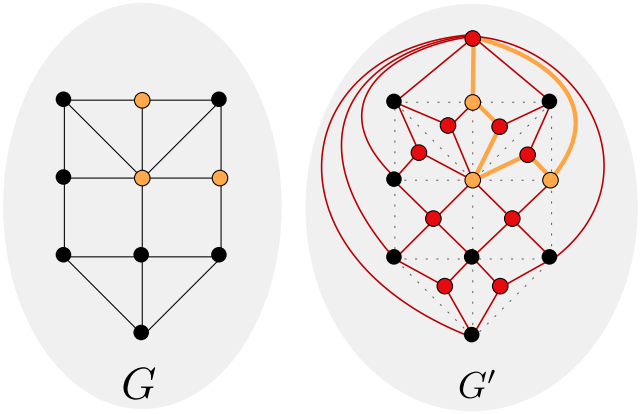

Figure 6: To construct the target graph G' from the embedding of the graph G, place a vertex v inside every face f of G and connect this vertex v to all the vertices of the face f (remove the original edges). Since there is a 6-cycle (highlighted and bold) in G' that separates the original vertices (black), but no smaller such cycle, the graph G is 3-connected. This cycle contains three original vertices (highlighted) whose removal disconnects the graph G.

Lemma 5.2. *Deciding Planar Vertex Connectivity (w.h.p) takes $O(n\log n)$ work and $O(\log^2 n)$ depth.*

Proof. The algorithm is correct by Lemma 5.1 and the fact that the vertex connectivity of a planar graph is at most 5. This follows from Euler's formula, which implies that every planar graph has a vertex of degree at most 5 [52]. Removing the neighbors of this vertex disconnects the graph, hence the graph is not 6-connected.

Constructing a planar embedding takes $O(n)$ work and $O(\log^2 n)$ depth [31]. Together with the modifications described in Section 5.2 (Lemma 5.3) this implies the work and depth bounds. □

Hence, we need to augment our subgraph isomorphism algorithm so that it can find a *subgraph that separates* a set of vertices (the original vertices in the case of the graph G').

A simple approach to find all *separating* cycles of a given length would be to enumerate *all* cycles of a given length using the algorithm from Section 4.2.1 and check which are separating. However, there can be $\Theta(n^4)$ many length 8 cycles in a planar graph [29], so this would be too much work.

5.2 Separating Subgraph Isomorphism

We generalize our parallel subgraph isomorphism algorithm so that it can find *subgraphs that separate* a given set of vertices. Two modifications are necessary. These are similar to what was necessary for the sequential algorithm [19] for cycles. The first modification is to the parallel treewidth cover algorithm from Section 2.1. This modification ensures that a subgraph that is separating in the original graph is also separating in each of the graphs in the cover. The

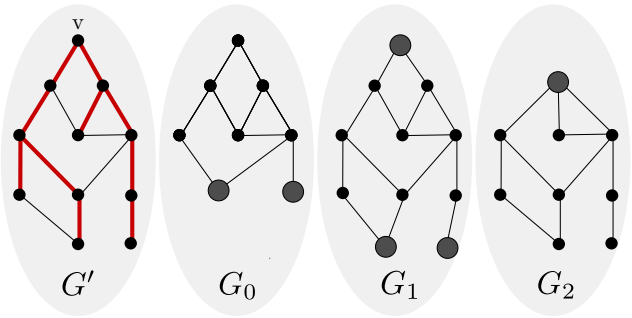

Figure 7: In addition to the vertices from the k-d cover, some vertices correspond to merged subgraphs (these are drawn larger in the picture). A subgraph of diameter d (here $d = 2$) that is separating in G' is separating in at least one of the minors G_0, G_1, G_2 using only the original (small) vertices.

second modification concerns the algorithm for bounded treewidth subgraph isomorphism from Section 3. It extends the state space of the recursion to keep track of which vertices are separated by the subgraph and which can be in the same component after removing the subgraph.

S-Separating Subgraph Isomorphism asks if there exists an occurrence H' of the pattern graph H in the target graph G that separates the vertex set $S \subseteq V(G)$. If the pattern graph is a cycle, the problem is called *S-Separating Cycle*.

5.2.1 How to Modify the k-d-Cover. Start by clustering the graph G as usual. Then, for each cluster, merge all neighboring clusters into a single vertex each (do not choose these as the source for the BFS). Then, in each cluster, instead of returning the graph G_i (which is an induced subgraph of the cluster), merge all connected components of the cluster that result after removing $V(G_i)$ into a single vertex each. This produces a set of minors of the graph (instead of a set of induced subgraphs), as shown in Figure 7.

When proceeding to find an S-separating subgraph in these minors, consider each merged vertex that contains at least one vertex of the set S to be in the set S. Moreover, do not allow the occurrence of the pattern to contain any of the merged vertices (the other vertices are in a set of allowed vertices A).

5.2.2 How to Modify the Bounded Treewidth Algorithm. The generalized algorithm must separate S and only contain vertices from the set of allowed vertices A. To restrict the found occurrences to only contain vertices from the set of allowed vertices, it suffices to restrict the mapping at each tree decomposition node to A.

The idea to find an occurrence that separates S is that we record which vertices are separated by the occurrence. Removing such an occurrence creates at least two connected components. We call one of these components the *inside* vertices and the rest of the vertices the *outside vertices*. Observe that after removing a separating occurrence from the graph, every resulting connected component must either consist of only inside or consist of only outside vertices.

We extend the construction of partial matches. A partial match for node X has an additional set $I_X \subseteq X$ of *on the inside* vertices and a set $O_X \subseteq X$ of *on the outside* vertices. Moreover, it has a boolean i_x to keep track if any of the vertices in S that occur in the subgraph induced by the current tree decomposition node are on the inside (and a boolean o_x to store if any of those vertices are on the outside). This bookkeeping ensures that at least one vertex is on both sides – otherwise, the subgraph would not be separating.

We adapt the semantics of the combination rules accordingly to reflect the intuition that partial matches keep track of which vertices are on the inside or outside. Consider a node X of the decomposition tree, one of its children Y, and the (extended) partial matches $(\phi_X, C_X, U_X, I_X, O_X, i_x, o_x)$ of X and $(\phi_Y, C_Y, U_Y, I_Y, O_Y, i_y, o_y)$ of Y. Then, for the partial matches to be valid, ensure the following:

- Every connected component of the subgraph of G induced by the vertices in X that are not mapped onto by the function ϕ_X is either fully in O_X or fully in I_X. Similarly for Y.
- The inside and outside of X and Y have to be consistent: For any vertex u, if $u \in X \cap Y$ then $u \in I_X$ if and only if $u \in I_Y$ and $u \in O_Y$ if and only if $u \in O_X$.
- The parent match has to 'remember' if any vertex is in S and on the inside or outside. Specifically, for a vertex $u \in S$, $u \in I_X$ implies i_x and $u \in O_X$ implies o_x. Moreover, i_y implies i_x and o_y implies o_x.

Finally, a valid partial match at the root must separate S (which means i_x and o_x are both true at the root).

Lemma 5.3. *Deciding* Planar S-Separating Subgraph Isomorphism *(w.h.p.) for a connected pattern graph with k vertices takes $O(k \log^2 n)$ depth and $O(2^{9k}(3k+1)^{3k+1} n \log n)$ work.*

Proof. Computing connected components and contracting the edges takes $O(n)$ work and $O(\log n)$ depth [27]. The number of states for the recursion increases by at most 2^{3k+3}. Hence, the number of considered combinations with the children increases by at most $O(2^{9k})$ at every node. □

When k is a constant, the algorithm takes $O(n \log n)$ work and $O(\log^2 n)$ depth. In Section 5.1, the only missing piece to solve planar vertex connectivity in $O(n \log n)$ work and $O(\log^2 n)$ depth is to find S-Separating 8-cycles, which we have just described how to solve in the stated bounds.

6 CONCLUSION AND FUTURE WORK

We presented a randomized algorithm to decide planar subgraph isomorphism in $O(n \log n)$ work and $O(\log^2 n)$ depth for constant size patterns. We used this result for deciding planar vertex connectivity in the same parallel bounds.

There are many interesting avenues for future work. Although we could use our subgraph listing algorithm to *count* the number of occurrences, this is not work-efficient as the runtime grows with the number of occurrences. The difficulty comes from the randomized way in which we cluster the graph to construct a k-d cover. A deterministic parallel k-d cover would solve this issue and yield a deterministic algorithm overall.

Reducing the work dependency on the size of the pattern k could be an essential step in improving the practicality of the approach. There are indications that $2^{\Omega(k/\log k)}$ is a lower bound for the dependency on k for any planar subgraph isomorphism algorithm with polynomial dependency in n [22], but there remains room for improvement regarding the exponential dependency on k. Moreover,

faster parallel algorithms for tree decomposition would directly improve our bounds for apex-minor-free graphs.

For planar vertex connectivity, we reduced the gap between the work of our algorithm and the best sequential algorithm to $O(\log n)$. It is natural to ask if it is possible to solve planar vertex connectivity in $O(n)$ work and poly-logarithmic depth. More generally, in light of the recently announced sequential near-linear time vertex connectivity algorithm for sparse graphs [43], it might be interesting to see if we can solve vertex connectivity in sparse graphs in near-linear work and low depth.

REFERENCES

[1] Alok Aggarwal, Maria Klawe, and Peter Shor. Multilayer grid embeddings for vlsi. *Algorithmica*, 6(1-6):129–151, 1991.

[2] Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995.

[3] Peter J. Artymiuk, Peter A. Bath, Helen M. Grindley, Catherine A. Pepperrell, Andrew R. Poirrette, David W. Rice, David A. Thorner, David J. Wild, and Peter Willett. Similarity searching in databases of three-dimensional molecules and macromolecules. *Journal of Chemical Information and Computer Sciences*, 32(6):617–630, 1992.

[4] Brenda S. Baker. Approximation algorithms for np-complete problems on planar graphs. *J. ACM*, 41(1):153–180, 1994.

[5] Max Bannach, Christoph Stockhusen, and Till Tantau. Fast parallel fixed-parameter algorithms via color coding. In *10th International Symposium on Parameterized and Exact Computation, IPEC 2015, September 16-18, 2015, Patras, Greece*, pages 224–235, 2015.

[6] Guy E. Blelloch. Programming parallel algorithms. *Commun. ACM*, 39(3):85–97, March 1996.

[7] Hans L. Bodlaender. Nc-algorithms for graphs with small treewidth. In *Graph-Theoretic Concepts in Computer Science, 14th International Workshop, WG '88, Amsterdam, The Netherlands, June 15-17, 1988, Proceedings*, pages 1–10, 1988.

[8] Hans L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993, San Diego, CA, USA*, pages 226–234, 1993.

[9] Hans L. Bodlaender, Pål Grønås Drange, Markus S. Dregi, Fedor V. Fomin, Daniel Lokshtanov, and Michal Pilipczuk. An o(cˆk n) 5-approximation algorithm for treewidth. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 499–508, 2013.

[10] Hans L. Bodlaender and Torben Hagerup. Parallel algorithms with optimal speedup for bounded treewidth. In *Automata, Languages and Programming, 22nd International Colloquium, ICALP95, Szeged, Hungary, July 10-14, 1995, Proceedings*, pages 268–279, 1995.

[11] Hans L. Bodlaender and Ton Kloks. Efficient and constructive algorithms for the pathwidth and treewidth of graphs. *J. Algorithms*, 21(2):358–402, 1996.

[12] Keren Censor-Hillel, Mohsen Ghaffari, and Fabian Kuhn. A new perspective on vertex connectivity. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 546–561, 2014.

[13] Zhi-Zhong Chen and Xin He. NC algorithms for partitioning planar graphs into induced forests and approximating np-hard problems. In *Graph-Theoretic Concepts in Computer Science, 21st International Workshop, WG '95, Aachen, Germany, June 20-22, 1995, Proceedings*, pages 275–289, 1995.

[14] Norishige Chiba and Takao Nishizeki. Arboricity and subgraph listing algorithms. *SIAM J. Comput.*, 14(1):210–223, 1985.

[15] Richard Cole, Philip N. Klein, and Robert Endre Tarjan. Finding minimum spanning forests in logarithmic time and linear work using random sampling. In *Proceedings of the 8th Annual ACM Symposium on Parallel Algorithms and Architectures, SPAA '96, Padua, Italy, June 24-26, 1996*, pages 243–250, 1996.

[16] Erik D. Demaine and Mohammad Taghi Hajiaghayi. Equivalence of local treewidth and linear local treewidth and its algorithmic applications. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2004, New Orleans, Louisiana, USA, January 11-14, 2004*, pages 840–849, 2004.

[17] Frederic Dorn. Planar subgraph isomorphism revisited. In *27th International Symposium on Theoretical Aspects of Computer Science, STACS 2010, March 4-6, 2010, Nancy, France*, pages 263–274, 2010.

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

[19] David Eppstein. Subgraph isomorphism in planar graphs and related problems. In *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, 22-24 January 1995. San Francisco, California, USA*, pages 632–640, 1995.

[20] David Eppstein. Diameter and treewidth in minor-closed graph families. *Algorithmica*, 27(3):275–291, 2000.

[21] David Eppstein, Maarten Löffler, and Darren Strash. Listing all maximal cliques in sparse graphs in near-optimal time. In *Algorithms and Computation - 21st International Symposium, ISAAC 2010, Jeju Island, Korea, December 15-17, 2010, Proceedings, Part I*, pages 403–414, 2010.

[22] Fedor V. Fomin, Daniel Lokshtanov, Daniel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. Subexponential parameterized algorithms for planar and apex-minor-free graphs via low treewidth pattern covering. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 515–524, 2016.

[23] Fedor V. Fomin, Daniel Lokshtanov, Michal Pilipczuk, Saket Saurabh, and Marcin Wrochna. Fully polynomial-time parameterized computations for graphs and matrices of low treewidth. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 1419–1432, 2017.

[24] Yu Gao, Jason Li, Danupon Nanongkai, Richard Peng, Thatchaphol Saranurak, and Sorrachai Yingchareonthawornchai. Deterministic graph cuts in subquadratic time: Sparse, balanced, and k-vertex. *CoRR*, abs/1910.07950, 2019.

[25] M. R. Garey, David S. Johnson, and Larry J. Stockmeyer. Some simplified npcomplete graph problems. *Theor. Comput. Sci.*, 1(3):237–267, 1976.

[26] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., USA, 1990.

[27] Hillel Gazit. Optimal EREW parallel algorithms for connectivity, ear decomposition and st-numbering of planar graphs. In *The Fifth International Parallel Processing Symposium, Proceedings, Anaheim, California, USA, April 30 - May 2, 1991*, pages 84–91, 1991.

[28] Barbara Geissmann and Lukas Gianinazzi. Parallel minimum cuts in near-linear work and low depth. In *Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures, SPAA 2018, Vienna, Austria, July 16-18, 2018*, pages 1–11, 2018.

[29] Seifollah Louis Hakimi and Edward F. Schmeichel. On the number of cycles of length k in a maximal planar graph. *Journal of Graph Theory*, 3(1):69–86, 1979.

[30] Monika Rauch Henzinger, Satish Rao, and Harold N. Gabow. Computing vertex connectivity: New bounds from old techniques. In *37th Annual Symposium on Foundations of Computer Science, FOCS '96, Burlington, Vermont, USA, 14-16 October, 1996*, pages 462–471, 1996.

[31] Philip N. Klein and John H. Reif. An efficient parallel algorithm for planarity. In *27th Annual Symposium on Foundations of Computer Science, Toronto, Canada, 27-29 October 1986*, pages 465–477, 1986.

[32] Philip N. Klein and Sairam Subramanian. A linear-processor polylog-time algorithm for shortest paths in planar graphs. In *34th Annual Symposium on Foundations of Computer Science, Palo Alto, California, USA, 3-5 November 1993*, pages 259–270, 1993.

[33] Michihiro Kuramochi and George Karypis. Frequent subgraph discovery. In *Proceedings of the 2001 IEEE International Conference on Data Mining, 29 November - 2 December 2001, San Jose, California, USA*, pages 313–320, 2001.

[34] Jens Lagergren. Efficient parallel algorithms for graphs of bounded tree-width. *J. Algorithms*, 20(1):20–44, 1996.

[35] Richard J. Lipton and Robert Endre. Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36(2):177–189, 1979.

[36] Jesús A. De Loera, Jrg Rambau, and Francisco Santos. *Triangulations*. Springer Berlin Heidelberg, 2010.

[37] Gary L. Miller, Richard Peng, Adrian Vladu, and Shen Chen Xu. Improved parallel algorithms for spanners and hopsets. In *Proceedings of the 27th ACM on Symposium on Parallelism in Algorithms and Architectures, SPAA 2015, Portland, OR, USA, June 13-15, 2015*, pages 192–201, 2015.

[38] Gary L. Miller and Vijaya Ramachandran. A new graph triconnectivity algorithm and its parallelization. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, pages 335–344, 1987.

[39] Gary L. Miller and John H. Reif. Parallel tree contraction and its application. In *26th Annual Symposium on Foundations of Computer Science, Portland, Oregon, USA, 21-23 October 1985*, pages 478–489, 1985.

[40] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon. Network motifs: Simple building blocks of complex networks. *Science*, 298(5594):824–827, 2002.

[41] Hiroshi Nagamochi, Toshimasa Ishii, and Hiro Ito. Minimum cost source location problem with vertex-connectivity requirements in digraphs. *Inf. Process. Lett.*, 80(6):287–293, 2001.

[42] Danupon Nanongkai, Thatchaphol Saranurak, and Sorrachai Yingchareonthawornchai. Breaking quadratic time for small vertex connectivity and an approximation scheme. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 241–252, 2019.

[43] Danupon Nanongkai, Thatchaphol Saranurak, and Sorrachai Yingchareonthawornchai. Computing and testing small vertex connectivity in near-linear time and queries. *CoRR*, abs/1905.05329, 2019.

[44] Miles Ohlrich, Carl Ebeling, Eka Ginting, and Lisa Sather. Subgemini: Identifying subcircuits using a fast subgraph isomorphism algorithm. In *Proceedings of the*

30th Design Automation Conference. Dallas, Texas, USA, June 14-18, 1993, pages 31–37, 1993.
[45] Ján Plesník. The np-completeness of the hamiltonian cycle problem in planar digraphs with degree bound two. *Inf. Process. Lett.*, 8(4):199–201, 1979.
[46] N. Przulj, D. G. Corneil, and I. Jurisica. Modeling interactome: scale-free or geometric? *Bioinformatics*, 20(18):3508–3515, 07 2004.
[47] John H. Reif. *Synthesis of Parallel Algorithms*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 1993.
[48] Neil Robertson and Paul D. Seymour. Graph minors. v. excluding a planar graph. *J. Comb. Theory, Ser. B*, 41(1):92–114, 1986.
[49] Frank R. Schmidt, Eno Töppe, and Daniel Cremers. Efficient planar graph cuts with applications in computer vision. In *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2009), 20-25 June 2009, Miami, Florida, USA*, pages 351–356, 2009.
[50] Robert Endre Tarjan and Uzi Vishkin. An efficient parallel biconnectivity algorithm. *SIAM J. Comput.*, 14(4):862–874, 1985.
[51] Julian R. Ullmann. An algorithm for subgraph isomorphism. *J. ACM*, 23(1):31–42, 1976.
[52] R.J. Wilson. *Introduction to Graph Theory*. Longman, 1996.

A DECOMPOSING A TREE INTO PATHS

We prove Lemma 3.2 using expression tree evaluation techniques. This means that we transform the problem into a problem of evaluating an expression tree of suitable operations. To evaluate this expression tree efficiently, we need to decompose the operations into unary functions satisfying certain properties, as described below.

Recall that the Lemma requires the tree to be split into $O(\log n)$ layers each consisting of disjoint paths. The idea is to compute for each vertex in the tree the layer in which the vertex occurs. This computes for each node a *layer number*, where the layer number of the leafs is zero and the layer number of nodes closer to the root is monotonically increasing (as detailed below).

Each layer (i.e. subgraph induced by vertices with the same layer number) consists of a forest where each connected component is a path. Hence, it is easy to find and order these paths (using list ranking) once we have the layer numbers.

Next, we describe the recursive function L that computes the layer numbers. In a general rooted tree, the parent b has the same layer number $l(b)$ as the maximum layer number of any of its children $a_1, \dots a_k$ if this maximum is unique (i.e., only one child has this layer number). Otherwise, the layer number of the parent is one larger than that maximum. In summary, the layer number $l(b)$ of node b with children $a_1, \dots a_k$ with layer numbers $l_1, \dots l_k$ is given recursively:

$$L(l_1, \dots l_k) = \begin{cases} \max(l_1, \dots l_k) & \text{if the maximum is unique ;} \\ \max(l_1, \dots l_k) + 1 & \text{otherwise .} \end{cases}$$

The layer number of a leaf is 0. This recursive description works because the case where the maximum is unique corresponds to when the parent is part of the same path as the child that obtains this maximum. If two children have the same layer number, the parent must start its own path and a new layer.

Moreover, observe that it becomes clear why there are $O(\log n)$ layers: For a parent to have a larger layer number than one of its children, there need to be at least two children of the same maximal layer number. This means that the number of nodes in a layer decreases by at least a factor 2 when going to a higher layer.

We proceed to describe the conditions for applying the efficient tree contraction based expression tree evaluation techniques, as summarized in Lemma A.1. A family of unary functions is *closed under composition* if the composition of any two functions in the family is also in the family. A family of unary functions $\mathcal{F}$ over the domain $\mathcal{D}$ is *closed under projection* with respect to a k-ary function $h : \mathcal{D}^k \to \mathcal{D}$ if for all tuples $a_1, \dots, a_{k-1} \in \mathcal{D}^{k-1}$ and all indexes i (between 1 and k) the function $h(a_1, \dots, a_{i-1}, x, a_{i+i} \dots, a_{k-1}) : \mathcal{D} \to \mathcal{D}$ (a unary function of x) is in the family $\mathcal{F}$.

Lemma A.1. *If there is a family of $O(1)$-computable functions that is closed under composition and closed under projection with respect to all the operations in an expression tree of n nodes, then evaluating the expression tree takes $O(n)$ work and $O(\log n)$ depth [47].*

The intuition is that the expression tree evaluation repeatedly contracts the expression tree. For this procedure to be well-defined, the algorithm needs to express partially evaluated subtrees using these unary functions. Next, we exhibit such a suitable family of unary functions for the function L that maps the layer number of the children to the layer number of the parent.

We define a set of unary functions over the domain of natural numbers, where for each natural number i, there are two functions: a function $f_i^{\neq}(x)$ and a function $g_i^{=}(x)$. Intuitively, the functions $f_i^{\neq}(x)$ record a state where the maximum (so far) is unique and equal to i. The functions $g_i^{=}(x)$ record the state where the maximum is not unique and equal to i. Formally, we set:

$$f_i^{\neq}(x) = \begin{cases} i+1 & \text{if } i = x \text{ ,} \\ \max(i, x) & \text{otherwise .} \end{cases}$$

$$g_i^{=}(x) = \begin{cases} i+1 & \text{if } i \geq x \text{ ,} \\ x & \text{if } i < x \text{ .} \end{cases}$$

We check that the function class is closed under composition. For any natural numbers i and j, the following holds:

$$g_j^{=}(\, f_i^{\neq}(x)\,) = f_i^{\neq}(\, g_j^{=}(x)\,) = \begin{cases} g_i^{=}(x) & \text{if } i = j \text{ ,} \\ f_i^{\neq}(x) & \text{if } i > j \text{ ,} \\ g_j^{=}(x) & \text{if } j > i \text{ .} \end{cases}$$

$$f_i^{\neq}(\, f_j^{\neq}(x)\,) = \begin{cases} g_i^{=}(x) & \text{if } i = j \text{ ,} \\ f^{\neq}_{\max(i,j)}(x) & \text{otherwise .} \end{cases}$$

$$g_i^{=}(\, g_j^{=}(x)\,) = g^{=}_{\max(i,j)}(x)$$

To check that the function class is closed under projection with respect to L, consider a sequence of layer values $\mathcal{L} = l_1, \dots, l_{k-1}$. Let $l_{\max}$ be the maximum of $\mathcal{L}$. For any valid index i we have that:

$$L(l_1, \dots, l_{i-1}, x, l_{i+1}, \dots, l_{k-1}) = \begin{cases} f^{\neq}_{l_{\max}}(x) & \text{if } l_{\max} \text{ is unique in } \mathcal{L}, \\ g^{=}_{l_{\max}}(x) & \text{otherwise.} \end{cases}$$