Derandomizing Isolation Lemma for $K_{3,3}$-free and K_5-free Bipartite Graphs

Rahul Arora[1], Ashu Gupta[2], Rohit Gurjar*[3], and Raghunath Tewari[4]

1 University of Toronto, arorar@iitk.ac.in
2 University of Illinois at Urbana-Champaign, ashug@iitk.ac.in
3 Aalen University, rgurjar@iitk.ac.in
4 Indian Institute of Technology Kanpur, rtewari@iitk.ac.in

Abstract

The perfect matching problem has a randomized NC algorithm, using the celebrated Isolation Lemma of Mulmuley, Vazirani and Vazirani. The Isolation Lemma states that giving a random weight assignment to the edges of a graph ensures that it has a unique minimum weight perfect matching, with a good probability. We derandomize this lemma for $K_{3,3}$-free and K_5-free bipartite graphs. That is, we give a deterministic log-space construction of such a weight assignment for these graphs. Such a construction was known previously for planar bipartite graphs. Our result implies that the perfect matching problem for $K_{3,3}$-free and K_5-free bipartite graphs is in SPL. It also gives an alternate proof for an already known result – reachability for $K_{3,3}$-free and K_5-free graphs is in UL.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes, G.2.2 Graph Theory

Keywords and phrases bipartite matching, derandomization, isolation lemma, SPL, minor-free graph

1 Introduction

The perfect matching problem is one of the most extensively studied problem in combinatorics, algorithms and complexity. In complexity theory, the problem plays a crucial role in the study of parallelization and derandomization. In a graph $G(V, E)$, a *matching* is a set of disjoint edges and a matching is called *perfect* if it covers all the vertices of the graph. Edmonds [12] gave the first polynomial time algorithm for the matching problem. Since then, there have been improvements in its sequential complexity [26], but an NC (efficient parallel) algorithm for it is not known. The perfect matching problem has various versions:
- Decision-PM: Decide if there exists a perfect matching in the given graph.
- Search-PM: Construct a perfect matching in the given graph, if it exists.

A randomized NC (RNC) algorithm for Decision-PM was given by [25]. Subsequently, Search-PM was also shown to be in RNC [19, 27]. The solution of Mulmuley et al. [27] was based on the powerful idea of *Isolation Lemma*. They defined a notion of an isolating weight assignment on the edges of a graph. Given a weight assignment on the edges, weight of a matching is defined to be the sum of the weights of all the edges in it.

▶ **Definition 1** ([27]). For a graph $G(V, E)$, a weight assignment $\mathbf{w}: E \to \mathbb{N}$ is isolating if G either has a unique minimum weight perfect matching according to w or has no perfect matchings.

* Supported by TCS Research Fellowship

ISSN 1433-8092

The Isolation Lemma states that a random integer weight assignment (polynomially bounded) is isolating with a good probability. Other parts of the algorithm in [27] are deterministic. They showed that if we are given an isolating weight assignment (with polynomially bounded weights) for a graph G, then a perfect matching in G can be constructed in NC^2. Later, Allender et al. [3] showed that the Decision-PM would be in SPL, which is in NC^2, if an isolating weight assignment can be constructed in L (see also [9]). A language L is in the class SPL if its characteristic function $\chi_L \colon \Sigma^* \rightarrow \{0,1\}$ can be (log-space) reduced to computing determinant of an integer matrix.

Derandomizing the Isolation Lemma remains a challenging open question. A general version of Isolation Lemma has also been studied, where one has to ensure a unique minimum weight set in a (non-explicitly) given family of sets (or multisets). Arvind and Mukhopadhyay [4] have shown that derandomizing this version of Isolation Lemma would imply circuit size lower bounds. While Reinhardt and Allender [30] have shown that derandomizing Isolation Lemma for some specific families of paths in a graph would imply $\mathsf{NL} = \mathsf{UL}$.

With regard to matchings, Isolation Lemma has been derandomized for some special classes of graphs: planar bipartite graphs [9, 33], constant genus bipartite graphs [10], graphs with small number of matchings [15, 1] and graphs with small number of nice cycles [16]. In a result subsequent to this work, Fenner et al. [13] achieved an almost complete derandomization of the isolation lemma for bipartite graphs. They gave a deterministic construction but with quasi-polynomially large weights. A graph G is bipartite if its vertex set can be partitioned into two parts V_1, V_2 such that any edge is only between a vertex in V_1 and a vertex in V_2. A graph is planar if it can be drawn on a plane without any edge crossings.

It is well known that a graph is planar if and only if it is both $K_{3,3}$-free and K_5-free [37]. For a graph H, G is an H-free graph if H is not a minor of G. $K_{3,3}$ is the complete bipartite graph with $(3,3)$ nodes and K_5 is the complete graph with 5 nodes. A natural generalization of planar bipartite graphs would be $K_{3,3}$-free bipartite graphs or K_5-free bipartite graphs. We make a further step towards the derandomization of Isolation Lemma by derandomizing it for these two graph classes. Note that these graphs are not captured by the classes of graphs mentioned above. In particular, a $K_{3,3}$-free or K_5-free graph can have arbitrarily high genus, exponentially many matchings or exponentially many nice cycles.

▶ **Theorem 2.** *Given a $K_{3,3}$-free or K_5-free bipartite graph, an isolating weight assignment (polynomially bounded) for it can be constructed in log-space.*

Another motivation to study these graphs came from the fact that Count-PM (counting the number of perfect matchings) is in NC^2 for $K_{3,3}$-free graphs [35] and in TC^2 ($\subseteq \mathsf{NC}^3$) for K_5-free graphs [32]. These were the best known results for Decision-PM too. The counting results, together with the known NC-reduction from Search-PM to Count-PM (for bipartite graphs) [22], implied an NC algorithm for Search-PM. Thus, a natural question was to find a direct algorithm for Search-PM via isolation, which we do here. One limitation of the earlier approach is that Count-PM is $\#\mathcal{P}$-hard for general bipartite graphs. Thus, there is no hope of generalizing this approach to work for all graphs. While the isolation approach can potentially lead to a solution for general/bipartite graphs.

Theorem 2 together with the results of Allender et al. [3] and Datta et al. [9] gives us the following results about matching.

▶ **Corollary 3.** *For a $K_{3,3}$-free or K_5-free bipartite graph,*

- Decision-PM *is in* SPL.
- Search-PM *is in* $\mathsf{FL}^{\mathsf{SPL}}$.

– Min-Weight-PM *is in* $\mathsf{FL}^{\mathsf{SPL}}$.

$\mathsf{FL}^{\mathsf{SPL}}$ is the set of function problems which can be solved by a log-space Turing machine with access to an SPL oracle. Like SPL, $\mathsf{FL}^{\mathsf{SPL}}$ also lies in NC^2. The problem Min-Weight-PM asks to construct a minimum weight perfect matching in a given graph with polynomially bounded weights on its edges.

The crucial property of these graphs, which we use, is that their 4-connected components are either planar or small sized. This property has been used to reduce various other problems on $K_{3,3}$-free or K_5-free graphs to their planar version, e.g. graph isomorphism [11], reachability [34]. However, their techniques do not directly work for the matching problem. There has been an extensive study on more general minor-free graphs by Robertson and Seymour. In a long series of works, they gave similar decomposition properties for these graphs [31]. Our approach for matching can possibly be generalized to H-free graphs for a larger/general graph H.

Our techniques: We start with the idea of Datta et al. [9] which showed that a skew-symmetric weight function on the edges ($w(u,v) = -w(v,u)$) such that every cycle has a nonzero circulation (weight in a fixed orientation) implies isolation of a perfect matching in bipartite graphs. To achieve nonzero circulation in a $K_{3,3}$-free or K_5-free graph, we work with its 3-connected or 4-connected component decomposition given by [37, 5], which can be constructed in log-space [34, 32]. The components are either planar or constant-sized and share a pair/triplet of vertices. These components form a *tree structure*, when each component is viewed as a node and there is an edge between two components if they share a pair/triplet. For any cycle C in the graph, we break it into its fragments contained within each of these components, which we call *projections* of C. Any such projection can be made into a cycle by adding virtual edges for separating pairs/triplets in the corresponding component.

Circulation of any cycle can be seen as a sum of circulations of its projections. The projections of a cycle can have circulations with opposite signs and thus, can cancel each other. To avoid this cancellation, we observe that the components, where a cycle has a non-empty projection form a subtree of the component tree. The idea is to assign edge weights using a different scale for each level of nodes in the tree. This ensures that for any subtree, its root node will contribute a weight higher than the total weight from all its other nodes. To avoid any cancellations within a component, weights in a component are given by modifying some known techniques for planar graphs [9, 21] and constant sized graphs.

This idea would work only if the component tree has a small depth, which might not be true in general. Thus, we create an $O(\log n)$-depth *working tree* by finding 'centers' for the component tree and its subtrees recursively. The construction of such a balanced working tree has been studied in context of evaluating arithmetic expressions [7]. In the literature, this construction is also known as 'centroid decomposition' or 'recursive balanced separators'. Its log-space implementation is more involved.

As the working tree has $O(\log n)$ depth, the straightforward way of using a different scale for each level will lead to edge weights being $n^{O(\log n)}$. So instead, in a component node, we assign weights to only those edges which surround a separating pair/triplet. The weighting scheme ensures that the total weight grows only by a constant multiple, when we move one step higher in the working tree.

Achieving non-zero circulation in log-space also puts directed reachability in UL [30, 6, 33]. Thus, we get an alternate proof for the result – directed reachability for $K_{3,3}$-free and K_5-free graphs is in UL [34].

In Section 2, we introduce the concepts of nonzero circulation, clique-sum, graph decomposition and the corresponding component tree. In Section 3, we give a log-space construction of a weight assignment with nonzero circulation for every cycle, for a class of graphs defined via clique-sum operations on planar and constant-sized graphs. In Appendix C, we argue that $K_{3,3}$-free and K_5-free graphs fall into this class.

2 Preliminaries

Let us first define a skew-symmetric weight function on the edges of a graph. For this, we consider the edges of the graph directed in both directions. We call this directed set of edges $\vec{E}$. A weight function $w\colon \vec{E} \to \mathbb{Z}$ is called skew-symmetric if for any edge (u, v), $w(u, v) = -w(v, u)$.

▶ **Definition 4** (Circulation). For a cycle C, whose edges are given by $\{(v_1, v_2), (v_2, v_3), \ldots, (v_{k-1}, v_k), (v_k, v_1)\}$, its circulation is defined to be $w(v_1, v_2) + w(v_2, v_3) + \cdots + w(v_k, v_1)$.

Clearly, as our weight function is skew-symmetric, changing the orientation of the cycle only changes the sign of the circulation. The following lemma [33, Theorem 6] gives the connection between nonzero circulations and isolation of a matching. For a bipartite (undirected) graph $G(V_1, V_2, E)$, a skew-symmetric weight function $w\colon \vec{E} \to \mathbb{Z}$ on its edges has a natural interpretation on the undirected edges as $\mathbf{w}\colon E \to \mathbb{Z}$ such that $\mathbf{w}(u, v) = w(u, v)$, where $u \in V_1$ and $v \in V_2$.

▶ **Lemma 5** ([33]). *Let $w\colon \vec{E} \to \mathbb{Z}$ be a skew-symmetric weight function on the edges of a bipartite graph G such that every cycle has a non-zero circulation. Then, $\mathbf{w}\colon E \to \mathbb{Z}$ is an isolating weight assignment for G.*

The bipartiteness assumption is needed only in the above lemma. We will construct a skew-symmetric weight function that guarantees nonzero circulation for every cycle, for a given $K_{3,3}$-free or K_5-free graph, i.e. without assuming bipartiteness.

2.1 Clique-sum

First, we will construct a nonzero circulation weight assignment for a special class of graphs, defined via a graph operation called *clique-sum*.

▶ **Definition 6** (Clique-sum). Let G_1 and G_2 be two graphs each containing a clique (of the same size). A clique-sum of graphs G_1 and G_2 is obtained from their disjoint union by identifying pairs of vertices in these two cliques to form a single shared clique, and by possibly deleting some of the edges in the clique. It is called a k-clique-sum if the cliques involved have at most k vertices.

One can form clique-sums of more than two graphs by a repeated application of clique-sum operation on two graphs (see Figure 2 in Appendix A). Using this, we define a new class of graphs. Let $\mathcal{P}_c$ be the class of all planar graphs together with all graphs of size at most c, where c is a constant. Define $\langle \mathcal{P}_c \rangle_k$ to be the class of graphs constructed by repeatedly taking k-clique-sums, starting from the graphs which belong to the class $\mathcal{P}_c$. In other words, it is the closure of $\mathcal{P}_c$ under k-clique sums. The starting graphs are called the component graphs. We will construct a nonzero circulation weight assignment for the graphs which belong to the class $\langle \mathcal{P}_c \rangle_3$.

Taking 1-clique-sum of two graphs will result in a graph which is not biconnected. As we are interested in perfect matchings, we only deal with biconnected graphs (see Appendix C.1). Thus, we assume that every clique-sum operation involves either 2-cliques or 3-cliques. A 2-clique which is involved in a clique-sum operation is called a separating pair. Similarly, a 3-clique is called a separating triplet. In general, they are called separating sets. Note that deletion of any separating pair/triplet will make the graph disconnected. We emphasize here that there can be other pairs/triplets in the graph which are not involved in a clique-sum operation, but whose deletion will make the graph disconnected. In this work, the term separating pair/triplet does not refer to such pairs/triplets.

2.2 Component Tree

In general, clique-sum operation can be performed many times using the same separating set. In other words, many components can share a separating set. In Appendix C, we show that any graph in $\langle \mathcal{P}_c \rangle_3$ can be modified via some matching preserving operations such that on decomposition, any separating set is shared by only two components. Henceforth, in this section we assume this property.

Using this assumption, we can define a component graph for any graph $G \in \langle \mathcal{P}_c \rangle_3$ as follows: each component is represented by a node and two such nodes are connected by an edge if the corresponding components share a separating set. Observe that this component graph is actually a tree. This is because when we take repeated clique-sums, a new component can be attached with only one of the already existing components, as a clique will be contained within one component. In literature [17, 34], the component tree also contains a node for each separating set and it is connected by all the components which share this separating set. But, here we can ignore this node as we have only two sharers for each separating set.

In the component tree, each component is shown with all the separating sets it shares with other components. Thus, a copy of a separating set is present in both its sharer components. Moreover, in each component, a separating set is shown with a virtual clique, i.e., a virtual edge for a separating pair and a virtual triangle for a separating triplet. These virtual cliques represent the paths between the nodes via other components (see Figure 3 in Appendix A). If any two vertices in a separating set have a real edge in G, then that real edge is drawn in one of the sharing components, parallel to the virtual edge. Note that while a vertex can have its copy in two components, any real edge is present in exactly one component.

In literature [17, 34], for any real edge in a separating set, the component tree contains a new node called "3-bond" (two vertices with a real edge and two parallel virtual edges). But, here we do not have this node and represent the real edge as mentioned above.

3 Nonzero Circulation

In this section, we construct a nonzero circulation weight assignment for a given graph in the class $\langle \mathcal{P}_c \rangle_3$, provided that the component tree and the planar embeddings of the planar components are given. Moreover, to construct this weight assignment we will make some assumptions about the given graph and its component tree.

1. In any component, a vertex is a part of at most one separating set.
2. Each separating set is shared by at most two components.
3. Any virtual triangle in a planar component is always a face.

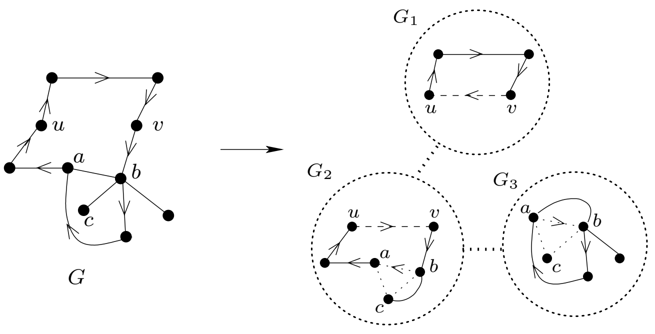

Figure 1 Breaking a cycle into its component cycles (projections) in the component tree. Notice that the original cycle and its components share the same set of *real* edges.

In Appendix C, we show how to construct a component tree for a given $K_{3,3}$-free or K_5-free graph and then to modify it to have these properties. The third property comes naturally, as the inside and outside parts of any virtual triangle can be considered as different components sharing this separating triplet. All these constructions are in log-space.

3.1 Components of a cycle

We look at a cycle in the graph as *sum* of many cycles, one from each component the cycle passes through. Intuitively, the original cycle is *broken* at the separating set vertices which were part of the cycle, thereby generating fragments of the cycle in various nodes of the component tree. In all the component nodes containing these fragments, we include the virtual edges of the separating sets in question to complete the fragment into a cycle, thus resulting in component cycles in the component nodes (see Figure 1).

Consider a directed cycle $C = \{(v_0, v_1), (v_1, v_2), \dots, (v_{k-1}, v_0)\}$ in a graph $G = (V, E)$. Without loss of generality, consider that G is separated into two components G_1 and G_2 via a separating pair (v_i, v_0) or a separating triplet (v_i, v_0, u), where $1 \leq i < k$ and $u \in V$. Then, one of the components, say G_1, will contain the vertices $v_i, v_{i+1 \bmod k}, \dots, v_{k-1}, v_0$, and the other ($G_2$) will contain the vertices $v_0, v_1, \dots, v_{i-1}, v_i$. Then the cycles $C_1 = \{(v_i, v_{i+1 \bmod k}), \dots, (v_{k-1}, v_0), (v_0, v_i)\}$ and $C_2 = \{(v_0, v_1), \dots, (v_{i-1}, v_i), (v_i, v_0)\}$ in G_1 and G_2 respectively are the component cycles of C, and we say that C is the sum of C_1 and C_2. Observe that the edges (v_i, v_0) and (v_0, v_i) are virtual.

Repeat the processes recursively for C_1 and C_2 until no separating set breaks a cycle component, and we get the component cycles of the cycle C. Note that any edge in a cycle C is contained in exactly one of its component cycles. Moreover for any component cycle, all its edges, other than the virtual edges, are contained in C.

Observe that for any separating set in a component, a cycle can use one of its vertices to go out of the component and another vertex to come in (this transition is represented by a virtual edge in the component). As any separating set has size at most 3, a cycle can visit a node of the component tree only once. In other words, a cycle can have only one component cycle in any component tree node (this would not be true if we had separating sets of size 4). Also, a component cycle can take only one edge of any virtual triangle.

▶ **Definition 7** (Projection of a cycle). For a given component node N in the component tree, the component cycle of a cycle C in N is called the projection of C on N. If there is no component cycle of C in N, then C is said to have an empty projection on N.

It is easy to see that for any cycle C, the components on which C has a non-empty projection, form a subtree of the component tree. To construct the weight assignment (Section 3.2), we will work with the component nodes of the component tree. Within any component, weight of a virtual edge will always be set to zero. Along with the fact that each cycle has the same set of real edges as the union of the edges in all its projections, this leads to the following lemma.

▶ **Lemma 8.** *The circulation of a cycle is the sum of the circulations of its component cycles.*

Note that for a cycle, its component cycles can have circulations with different signs (positive or negative) as they can have different orientations (clockwise or anti-clockwise) in the planar components. Hence the total circulation can potentially be zero. Our idea is to ensure that one of the component cycles get a circulation greater than all the other component cycles put together. This will imply a nonzero circulation.

3.2 Weighting Scheme

The actual weight function we employ is a combination of two weight functions w_0 and w_1. They are combined with an appropriate scaling so that they do not interfere with each other. w_1 ensures that all the cycles which are within one component have a non-zero circulation and w_0 ensures that all the cycles which project on at least two components have a non-zero circulation. We first describe the construction of w_0.

Working Tree: The given component tree can have arbitrary depth, while our weight construction would need the tree-depth to be $O(\log n)$. Thus, we re-balance the tree to construct a new *working tree*. It is a rooted tree which has the same nodes as the component tree, but the edge relations are different. The working tree, in some sense, 'preserves' the subtree structure of the original tree.

For a tree S, its working tree $\mathsf{wt}(S)$ is constructed as follows: Find a 'center' node $c(S)$ in the tree S and mark it as the root of the working tree, $r(\mathsf{wt}(S))$. Deleting the node $c(S)$ from the tree S would give a set of disjoint trees, say $\{S_1, S_2, \dots, S_k\}$. Apply this procedure recursively on these trees to construct their working trees $\mathsf{wt}(S_1), \mathsf{wt}(S_2), \dots, \mathsf{wt}(S_k)$. Connect each $\mathsf{wt}(S_i)$ to the root $r(\mathsf{wt}(S))$, as a subtree. In other words, $r(\mathsf{wt}(S_i))$ is a child of $r(\mathsf{wt}(S))$. For the base case, when the tree is a node, its working tree is the node itself. This completes the construction. If the component $c(S)$ shares the separating set τ_i with S_i, then the subtree $\mathsf{wt}(S_i)$ is said to be attached to the root $r(\mathsf{wt}(S))$ at τ_i.

The 'center' nodes are chosen in a balanced way so that the working tree depth is $O(\log n)$. Von Braunmühl and Verbeek [36], and later Limaye et al. [23], gave a log-space construction of such a balanced tree, but in terms of well-matched strings (also see [8]). In Section 3.3, we present the log-space construction in terms of a tree, along with a precise definition of a 'center' node.

Note that for any two nodes $v_1 \in S_i$ and $v_2 \in S_j$ such that $i \neq j$, $\mathrm{path}(v_1, v_2)$ in S passes through the node $c(S) = r(\mathsf{wt}(S))$. Thus, we get the following property for the working tree.

▶ Claim 9. For any two nodes $u, v \in S$, let their least common ancestor in the working tree $\mathsf{wt}(S)$ be the node a. Then $\mathrm{path}(u, v)$ in the tree S passes through a.

The root $r(\mathsf{wt}(S))$ of the working tree $\mathsf{wt}(S)$ is said to be at depth 0. For any other node in $\mathsf{wt}(S)$, its depth is defined to be one more than the depth of its parent. Henceforth, depth of a node will always mean its depth in the working tree. From Claim 9, we get the following.

▶ Claim 10. Let S' be an arbitrary subtree of S, with its set of nodes being $\{v_1, v_2, \ldots, v_k\}$. There exists $i^* \in \{1, 2, \ldots, k\}$ such that for any $j \in [k]$ with $j \neq i^*$, v_j is a descendant of v_{i^*} in the working tree $\mathsf{wt}(S)$.

Proof. Let d^* be the minimum depth of any node in S', and let v_{i^*} be a node in S' with depth d^*. We claim that every other node in S' is a descendant of v_{i^*} in the working tree $\mathsf{wt}(S)$. For the sake of contradiction, let there be a node $v_j \in S'$ which is not a descendant of v_{i^*}. Then, the least common ancestor of v_j and v_{i^*} in $\mathsf{wt}(S)$ must have depth strictly smaller than d^*. By Claim 9, this least common ancestor must be present in the tree S'. But, we assumed d^* is the minimum depth value in S'. Thus, we get a contradiction. ◀

This claim plays a crucial role in our weight assignment construction, as for any cycle C the components with a non-empty projection of C form a subtree of the component tree. To assign weights in the graph, we work with the working tree of its component tree. Let the working tree be $\mathcal{T}$. We start by assigning weight to the nodes having the largest depth, and move up till we reach depth 0, that is, the root node $r(\mathcal{T})$. The idea is that for any cycle C, its unique least-depth projection should get a circulation higher than the total circulation of all its other projections.

Complementary to the depth, we also define *height* of every node in the working tree. Let the maximum depth of any node in the working tree be D. Then, the height of a node is defined to be the difference between its depth and $D + 1$.

Circulations of cycles spanning multiple components: For any subtree T of the working tree $\mathcal{T}$, the weights to the edges inside the component $r(T)$ will be given by two different schemes depending on whether the corresponding graph is planar or constant sized.

Let the maximum possible number of edges in a constant sized component be m. Then, let K be a constant such that $K > \max(2^{m+2}, 7)$. Also, suppose that the height of a node N is given by the function $h(N)$, and the number of leaves in subtree T is given by $l(T)$. Lastly, suppose the set of subtrees attached at $r(T)$ is $\{T_1, T_2, \ldots, T_k\}$.

Constant sized graph: Let the set of (real) edges of the graph be $\{e_1, e_2, \ldots, e_m\}$. The edge e_j will be given weight $2^j \times K^{h(r(T))-1} \times l(T)$ for an arbitrarily fixed direction. The intuition behind this scheme is that powers of 2 ensure that sum of weights for any non-empty subset of edges remain nonzero even when they contribute with different signs.

Planar graph: We work with a given planar embedding of the graph. For any weight assignment $w : \vec{E} \to \mathbb{Z}$ on the edges of the graph, we define the *circulation of a face* as the circulation of the corresponding cycle in the clockwise direction, i.e., traverse the boundary edges of the face in the clockwise direction and take the sum of their weights. Instead of directly assigning edge weights, we will fix circulations for the inner faces of the graph. As we will see later, fixing positive circulations for all inner faces will avoid any cancellations. Lemma 14 describes how to assign weights to the edges of a planar graph to get the desired circulation for each of the inner faces.

Assigning circulations to the faces: Here, only those inner faces are assigned nonzero circulations which are adjacent to some separating pair/triplet shared with a subtree. This is a crucial idea. As we will see in Lemma 11, this ensures that the maximum possible circulation of a cycle grows only by a constant multiple as we move one level higher up in the working tree.

If T is a singleton, i.e., there are no subtrees attached at T, we give a zero circulation to all the faces (and thus zero weight to all the edges) of $r(T)$. Otherwise, consider a separating pair $\{a, b\}$ where a subtree T_i is attached to $r(T)$. The two faces adjacent to the virtual edge

(a, b) will be assigned circulation $2 \times K^{h(r(T_i))} \times l(T_i)$. Similarly, consider a triplet $\{a, b, c\}$ where a subtree T_j is attached. Then all the faces (at most 3) adjacent to the virtual triangle $\{a, b, c\}$ get circulation $2 \times K^{h(r(T_j))} \times l(T_j)$. Repeat this procedure for all the faces adjacent to any pairs and/or triplets where subtrees are attached. If a face is adjacent to more than one virtual edge/triangle, then we just take the sum of different circulations due to each virtual edge/triangle.

Recall that by definition, each face has a positive circulation in the clockwise direction. The intuition behind this scheme is the following: circulation of any cycle in the planar component is just the sum of circulations of the faces inside it (Lemma 12). As all of them have the same sign, they cannot cancel each other. Moreover, it will be ensured that the contribution to the circulation from this planar component is higher than the total contribution from all its subtrees, and thus, cannot be canceled.

Now, we formally show that this weighting scheme ensures that all the cycles spanning multiple components in the tree get non-zero circulation.

Nonzero Circulation of a cycle: First, we derive an upper bound on the circulation of any cycle completely contained in a subtree T of the working tree.

▶ **Lemma 11.** *The upper bound on the circulation of any cycle contained in a subtree T of the working tree $\mathcal{T}$ is $U_T = K^{h(r(T))} \times l(T)$.*

Proof. We prove this using induction on the height of $r(T)$.

Base case: The height of $r(T)$ is 1. Notice that this means that $r(T)$ has the maximum depth amongst all the nodes in $\mathcal{T}$, and therefore, $r(T)$ is a leaf node, and T is a singleton. Consider the two cases: i) when $r(T)$ is a planar graph, ii) when it is a constant sized graph.

By our weight assignment, if $r(T)$ is planar, the total weight of all the edges is zero. On the other hand, if $r(T)$ is a constant sized graph, the maximum circulation of a cycle is the sum of weights of its edges, that is, $\sum_{i=1}^{m}(K^0 \times 1 \times 2^i) < 2^{m+1} \leq K$. Thus, the circulation is upper bounded by $K^{h(r(T))} \times l(T)$ (as $l(T) = 1$).

Induction hypothesis: For any tree T' with $h(r(T')) \leq j - 1$, the upper bound is $U_{T'} = K^{h(r(T'))} \times l(T')$.

Induction step: We will prove that for any tree T with $h(r(T)) = j$, the upper bound is $U_T = K^{h(r(T))} \times l(T)$.

Let the subtrees attached at $r(T)$ be $\{T_1, T_2, \ldots, T_k\}$. For any cycle in T, sum of the circulations of its projections on the subtrees $T_1, T_2, \ldots, T_k$ can be at most $\sum_{i=1}^{k} U_{T_i}$.

First, we handle the case when $r(T)$ is planar. For any subtree T_i, the total circulation of faces in $r(T)$ due to connection to T_i can be $6 \times K^{h(r(T_i))} \times l(T_i)$. This is because the circulation of each face adjacent to the separating set connecting with T_i is $2 \times K^{h(r(T_i))} \times l(T_i)$, and there can be at most 3 such faces. Thus,

$$\begin{aligned}
U_T &= \sum_{i=1}^{k} U_{T_i} + \sum_{i=1}^{k} \left(6 \times K^{h(r(T_i))} \times l(T_i)\right) \\
&= \sum_{i=1}^{k} \left(K^{h(r(T_i))} \times l(T_i)\right) + \sum_{i=1}^{k} \left(6 \times K^{h(r(T_i))} \times l(T_i)\right) \\
&= 7 \times K^{h(r(T))-1} \times \sum_{i=1}^{k} l(T_i) \qquad (\because \forall i,\ h(r(T_i)) = h(r(T)) - 1) \\
&< K^{h(r(T))} \times \sum_{i=1}^{k} l(T_i) \qquad (\because K > 7)
\end{aligned}$$

$= K^{h(r(T))} \times l(T)$

Now, consider the case when $r(T)$ is a small non-planar graph. The maximum possible contribution from edges of $r(T)$ to the circulation of a cycle in T is less than $2^{m+1} \times K^{h(r(T))-1} \times l(T)$. Similar to the case when $r(T)$ is planar, contribution from all subtrees is at most $K^{h(r(T))-1} \times l(T)$. The total circulation of a cycle in T can be at most the sum of these two bounds, and is thus bounded above by $(2^{m+1}+1) \times K^{h(r(T))-1} \times l(T)$. Since, $K > 2^{m+2}$, the total possible circulation is less than $K^{h(r(T))} \times l(T)$.

Therefore, the upper bound $U_T = K^{h(r(T))} \times l(T)$. ◀

To see that each cycle gets a nonzero circulation, recall Lemma 8, which says that the circulation of the cycle is the sum of circulations of its projections on different components. Consider a cycle C. Recall that components with a non-empty projection of C form a subtree S_C in the component tree. From Claim 10, we can find a node $v^* \in S_C$ such that all other nodes in S_C are its descendants in the working tree $\mathcal{T}$. Thus, v^* is the unique minimum depth component on which C has a non-empty projection. Now, we show two things: (i) the contribution to the circulation from this component is nonzero, and (ii) it is larger than sum of all the circulation contributions from all its subtrees in the working tree.

Let v^* be the root of a subtree T in the working tree. Let the subtrees attached at $r(T)$ $(= v^*)$ be $\{T_1, T_2, \ldots, T_k\}$ and the separating sets in $r(T)$ at which they are attached be $\{\tau_1, \tau_2, \ldots, \tau_k\}$ respectively.

Case 1: when $r(T)$ is a constant-sized component. It is easy to see that the circulation of any cycle in this component will be nonzero as long as it takes a real edge, because the weights given are powers of 2. Also, the minimum weight of any edge in $r(T)$ is $2 \times \sum_{i=1}^{k} U_{T_i}$. Thus, when a cycle takes a real edge, contribution to its circulation from $r(T)$ is larger than the contribution from higher depth components (components in the subtrees attached at $r(T)$). And any cycle has to take a real edge, as the virtual edges and triangles all have disjoint set of vertices. (Here, the virtual triangle does not count as a cycle).

Case 2: when $r(T)$ is a planar component. The crucial observation here is that in a planar graph, all the faces inside a cycle contribute to its circulation in the same orientation.

▶ **Lemma 12** ([6]). *In a planar graph, circulation of a cycle in clockwise orientation is the sum of circulations of the faces inside it (Proof given in Appendix B).*

Since C passes through at least one of the subtrees attached at $r(T)$, say T_i, it must go through the separating set τ_i. Hence, the projection of C in $r(T)$, say C', must use the virtual edge (or one of the edges in the virtual triangle) corresponding to τ_i. This would imply that at least one of the faces adjacent to τ_i is inside C'. This is true for any subtree T_i which C passes through. As the faces adjacent to separating sets have nonzero circulations and each face has a positive circulation in clockwise direction, the circulation of C' is nonzero.

Recall that circulation of any face adjacent to τ_i is $2U_{T_i}$, where U_{T_i} is the upper bound on circulation contribution from T_i. This implies that the circulation of C' will surpass the total circulation from all the subtrees which C passes through. Thus, we can conclude the following.

▶ **Lemma 13.** *Circulation of any cycle which passes through at least two components is nonzero.*

Face circulations using edge weights: Now, we come back to the question of assigning weights to the edges in a planar component such that the faces get the desired circulations. Lemma 14 describes this procedure for any planar graph.

▶ **Lemma 14** ([21]). *Let $G(V, E)$ be a planar graph with F being its set of inner faces in some planar embedding. For any given function on the inner faces $w' : F \to \mathbb{Z}$, a skew-symmetric weight function $w\colon \vec{E} \to \mathbb{Z}$ can be constructed in log-space such that each face $f \in F$ has a circulation $w'(f)$ (Proof is described in Appendix B).*

This scheme can assign weight to any edge in the given graph, while we are not allowed to give weights to virtual edges/triangles. So, we first collapse all the virtual triangles to one node and all the virtual edges to one node. As no two virtual triangles/edges are adjacent, after this operation, every face remains a non-trivial face (except the virtual triangle face). Now, we apply the procedure from Lemma 14. After undoing the collapse, the circulations of the faces will not change and we will have the desired circulations.

Circulation of cycles contained within a single component: To construct w_1 for planar components, we assign $+1$ circulation to every face using Lemma 14 (similar to the case of multiple components). This would ensure nonzero circulation for every cycle within the planar component. This construction has been used in [21] for bipartite planar graphs. [33] also gives a log-space construction which ensures nonzero circulation for all cycles in a planar graph, using Green's theorem.

For the non-planar components, w_0 already ensures that each cycle has non-zero circulation. Therefore, we set $w_1 = 0$. Use a linear combination of w_0 and w_1 such that they do not interfere with each other. Such a combination is easy to achieve by multiplying w_1 by n^2 or a higher power of n since since w_0 is $O(n)$. This together with Lemma 13 gives us the following.

▶ **Lemma 15.** *Circulation of any cycle is non-zero.*

Complexity: In Appendix B.1, we argue that the weights given by this scheme are polynomially bounded and the weight-construction procedure can be done in log-space.

3.3 Construction of the Working Tree

Now, we describe a log-space construction of the working tree. The idea is obtained from the construction of [23, Lemma 6], where they create a $O(\log n)$-depth tree of well-matched substrings of a given well-matched string. Recall that for a tree S, the working tree $\mathsf{wt}(S)$ is constructed by first choosing a center node $c(S)$ of S and marking it as the root of $\mathsf{wt}(S)$, and then recursively finding the working trees for each component obtained by removing the node $c(S)$ from S and connecting them to the root of $\mathsf{wt}(S)$, as subtrees.

First, consider the following possible definition of the center: for any tree S with n nodes, one can define its center to be a node whose removal would give disjoint components of size $\leq 1/2|S|$. Finding such a center is an easy task and can be done in log-space. Clearly, the depth of the working tree would be $O(\log n)$. It is not clear if the recursive procedure of finding centers for each resulting component can be done in log-space. Therefore, we give a more involved way of defining centers, so that the whole recursive procedure can be done in log-space.

First, we make the tree S rooted at an arbitrary node r. To find the child-parent relations of the rooted tree, one can do the standard log-space traversal of a tree.

Tree traversal [24]: for every node, give its edges an arbitrary cyclic ordering. Start traversing from the root r by taking an arbitrary edge. If you arrive at a node u using its edge e then leave node u using the right neighbor of e. This traversal ends at r with every edge being traversed exactly twice.

For any node v, let S_v denote the subtree of S, rooted at v. For any node v and one of its descendant nodes v' in S, let $S_{v,v'}$ denote the tree $S_v \setminus S_{v'}$. Moreover $S_{v,\epsilon}$ would just mean S_v, for any v. With our new definition of the center, at any stage of the recursive procedure, the component under consideration will always be of the form $S_{v,v'}$, for some nodes $v, v' \in S$. Now, we give a definition of the center for a rooted tree of the form $S_{v,v'}$.

Center $c(S_{v,v'})$: case (i) When $v' = \epsilon$, i.e. the given tree is S_v. Let c be a node in S_v, such that its removal gives components of size $\leq 1/2|S_v|$. If there are more than one such nodes then choose the lexicographically smallest one (there is at least one such center [18]). Define c as the center of $S_{v,v'}$.

Let the children of c in S_v be $\{c_1, c_2, \ldots, c_k\}$. Clearly, after removing c from S_v, the components we get are $S_{c_1}, S_{c_2}, \ldots, S_{c_k}$ and $S_{v,c}$. Thus, they are all of the form described above, and have size $\leq 1/2|S_v|$.

case (ii) When v' is an actual node in S_v. Let the node sequence on the path connecting v and v' be $(u_0, u_1, \ldots, u_p)$, with $u_0 = v$ and $u_p = v'$. Let $0 \leq i < p$ be the least index such that $|S_{u_{i+1},v'}| \leq 1/2|S_{v,v'}|$. This index exists because $|S_{u_p,v'}| = 0$. Define u_i as the center of $S_{v,v'}$.

Let the children of u_i, apart from u_{i+1}, be $\{c_1, c_2, \ldots, c_k\}$. After removal of u_i from $S_{v,v'}$, the components we get are $S_{c_1}, S_{c_2}, \ldots, S_{c_k}$, $S_{u_{i+1},v'}$ and S_{v,u_i}. By the choice of i, $|S_{u_i,v'}| > 1/2|S_{v,v'}|$. Thus, $|S_{v,u_i}| \leq 1/2|S_{v,v'}|$. So, the only components for which we do not have a guarantee on their sizes, are $S_{c_1}, S_{c_2}, \ldots, S_{c_k}$. Observe that when we find a center for the tree $S_{c_j,\epsilon}$ in the next recursive call, it will fall into case (i) and the components we get will have their sizes reduced by a factor of $1/2$.

Thus, we can conclude that in the recursive procedure for constructing the working tree, we reduce the size of the component by half in at most two recursive calls. Hence, the depth of working tree is $O(\log n)$. Now, we describe a log-space procedure for the working tree.

▶ **Lemma 16.** *For any tree S, its working tree* $\mathsf{wt}(S)$ *can be constructed in log-space.*

Proof. We just describe a log-space procedure for finding the parent of a given node x in the working tree. Running this procedure for every node will give us the working tree.

Find the center of the tree S. Removing the center would give many components. Find the component S_1, to which the node x belongs. Apply the same procedure recursively on S_1. Keep going to smaller components which contain x, till x becomes the center of some component. The center of the previous component in the recursion will be the parent of x in the working tree.

In this recursive procedure, to store the current component $S_{v,v'}$, we just need to store two nodes v and v'. Apart from these, we need to store center of the previous component and size of the current component.

To find the center of a given component $S_{v,v'}$, go over all possibilities of the center, depending on whether v' is ϵ or a node. For any candidate center c, find the sizes of the components generated if c is removed. Check if the sizes satisfy the specified requirements. Any of these components is also of the form $S_{u,u'}$ and thus can be stored with two nodes.

By the standard log-space traversal of a tree, for any given tree $S_{v,v'}$, one can count the number of nodes in it and test membership of a given node. Thus, the whole procedure works in log-space. ◀

4 Discussion

One of the open problems is to construct a polynomially bounded isolating weight assignment for a more general class of graphs, in particular, for all bipartite graphs. Note that *nonzero circulation* for every cycle is sufficient but not necessary for constructing an isolating weight assignment. Although existence of an isolating weight assignment can be shown by randomized arguments, no such arguments exist for showing the existence of a nonzero circulation weight assignment. It needs to be investigated whether it is possible to achieve a nonzero circulation for every cycle (with polynomially bounded weights) in a complete bipartite graph. Log-space construction of such a weight assignment would imply that Bipartite Perfect Matching is in NC and answer the NL=UL? question.

Our approach does not directly extend to more general minor-free graphs, because their decomposition can involve separating sets of size more than 3. For example, when we have a separating set of size 4, a cycle can have two different projections in a component, i.e., it enters the component twice and leaves the component twice. These two projections can contribute to the total circulation with opposite signs and can cancel each other.

Like $K_{3,3}$-free or K_5-free graphs, small genus graphs are another generalization of planar graphs for which Count-PM is in NC [14, 22]. Thus, Search-PM for small genus bipartite graphs is in NC [22]. Can we do a log-space isolation of a perfect matching for these graphs?

The isolation question is also open for general planar graphs. In fact, planar graphs do not have any NC algorithm for Search-PM, via isolation or otherwise. On the other hand, counting the number of perfect matchings in a planar graph is in NC. It is surprising, as counting seems to be a harder problem than isolation.

5 Acknowledgements

We thank Arpita Korwar for various helpful discussions.

References

1 Manindra Agrawal, Thanh Minh Hoang, and Thomas Thierauf. The polynomially bounded perfect matching problem is in NC^2. In Wolfgang Thomas and Pascal Weil, editors, *STACS 2007*, volume 4393 of *Lecture Notes in Computer Science*, pages 489–499. Springer Berlin Heidelberg, 2007.

2 Eric Allender and Meena Mahajan. The complexity of planarity testing. *Information and Computation*, 189(1):117 – 134, 2004.

3 Eric Allender, Klaus Reinhardt, and Shiyu Zhou. Isolation, matching, and counting uniform and nonuniform upper bounds. *J. Comput. Syst. Sci.*, 59(2):164–181, 1999.

4 V. Arvind and Partha Mukhopadhyay. Derandomizing the isolation lemma and lower bounds for circuit size. In Ashish Goel, Klaus Jansen, JoséD.P. Rolim, and Ronitt Rubinfeld, editors, *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques*, volume 5171 of *Lecture Notes in Computer Science*, pages 276–289. Springer Berlin Heidelberg, 2008.

5 Takao Asano. An approach to the subgraph homeomorphism problem. *Theoretical Computer Science*, 38(0):249 – 267, 1985.

6 Chris Bourke, Raghunath Tewari, and N. V. Vinodchandran. Directed planar reachability is in unambiguous log-space. *ACM Trans. Comput. Theory*, 1(1):4:1–4:17, February 2009.

7 Richard P. Brent. The parallel evaluation of general arithmetic expressions. *J. ACM*, 21(2):201–206, April 1974.

8 Bireswar Das, Samir Datta, and Prajakta Nimbhorkar. Log-space algorithms for paths and matchings in k-trees. *Theory of Computing Systems*, 53(4):669–689, 2013.

9 Samir Datta, Raghav Kulkarni, and Sambuddha Roy. Deterministically isolating a perfect matching in bipartite planar graphs. *Theory of Computing Systems*, 47:737–757, 2010.

10 Samir Datta, Raghav Kulkarni, Raghunath Tewari, and N. V. Vinodchandran. Space complexity of perfect matching in bounded genus bipartite graphs. *J. Comput. Syst. Sci.*, 78(3):765–779, 2012.

11 Samir Datta, Prajakta Nimbhorkar, Thomas Thierauf, and Fabian Wagner. Graph isomorphism for $K_{3,3}$-free and K_5-free graphs is in log-space. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2009, December 15-17, 2009, IIT Kanpur, India*, pages 145–156, 2009.

12 Jack Edmonds. Path, trees, and flowers. *Canadian J. Math.*, 17:449–467, 1965.

13 Stephen A. Fenner, Rohit Gurjar, and Thomas Thierauf. Bipartite perfect matching is in quasi-nc. Technical Report TR15-177, Electronic Colloquium on Computational Complexity (ECCC), 2015.

14 Anna Galluccio and Martin Loebl. On the theory of Pfaffian orientations. I. perfect matchings and permanents. *Electr. J. Comb.*, 6, 1999.

15 Dima Grigoriev and Marek Karpinski. The matching problem for bipartite graphs with polynomially bounded permanents is in NC (extended abstract). In *28th Annual Symposium on Foundations of Computer Science, Los Angeles, California, USA, 27-29 October 1987*, pages 166–172, 1987.

16 Thanh Minh Hoang. On the matching problem for special graph classes. In *IEEE Conference on Computational Complexity*, pages 139–150. IEEE Computer Society, 2010.

17 John E. Hopcroft and Robert Endre Tarjan. Dividing a graph into triconnected components. *SIAM J. Comput.*, 2(3):135–158, 1973.

18 Camille Jordan. Sur les assemblages de lignes. *Journal für die reine und angewandte Mathematik*, 70:185–190, 1869.

19 Richard M. Karp, Eli Upfal, and Avi Wigderson. Constructing a perfect matching is in random NC. *Combinatorica*, 6(1):35–48, 1986.

20 S. Khuller. *Parallel Algorithms for K_5-minor Free Graphs*. Cornell University, Department of Computer Science, 1988.

21 Arpita Korwar. Matching in planar graphs. Master's thesis, Indian Institute of Technology Kanpur, 2009.

22 Raghav Kulkarni, Meena Mahajan, and Kasturi R. Varadarajan. Some perfect matchings and perfect half-integral matchings in NC. *Chicago Journal of Theoretical Computer Science*, 2008(4), September 2008.

23 Nutan Limaye, Meena Mahajan, and B.V.Raghavendra Rao. Arithmetizing classes around NC_1 and l. In *STACS 2007*, volume 4393 of *Lecture Notes in Computer Science*, pages 477–488. Springer Berlin Heidelberg, 2007.

24 Steven Lindell. A logspace algorithm for tree canonization (extended abstract). In *Proceedings of the Twenty-fourth Annual ACM Symposium on Theory of Computing*, STOC '92, pages 400–404, New York, NY, USA, 1992. ACM.

25 László Lovász. On determinants, matchings, and random algorithms. In *FCT*, pages 565–574, 1979.

26 Silvio Micali and Vijay V. Vazirani. An $O(\sqrt{V}E)$ algorithm for finding maximum matching in general graphs. In *Proceedings of the 21st Annual Symposium on Foundations of Computer Science*, SFCS '80, pages 17–27, Washington, DC, USA, 1980. IEEE Computer Society.

27 Ketan Mulmuley, Umesh V. Vazirani, and Vijay V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7:105–113, 1987.

28 Noam Nisan and Amnon Ta-Shma. Symmetric logspace is closed under complement. In Frank Thomson Leighton and Allan Borodin, editors, *STOC*, pages 140–146. ACM, 1995.

29 Omer Reingold. Undirected connectivity in log-space. *J. ACM*, 55:17:1–17:24, September 2008.

30 Klaus Reinhardt and Eric Allender. Making nondeterminism unambiguous. *SIAM J. Comput.*, 29(4):1118–1131, 2000.

31 Neil Robertson and P.D Seymour. Graph minors. xvi. excluding a non-planar graph. *Journal of Combinatorial Theory, Series B*, 89(1):43 – 76, 2003.

32 Simon Straub, Thomas Thierauf, and Fabian Wagner. Counting the number of perfect matchings in K_5-free graphs. In *IEEE 29th Conference on Computational Complexity, CCC 2014, Vancouver, BC, Canada, June 11-13, 2014*, pages 66–77, 2014.

33 Raghunath Tewari and N. V. Vinodchandran. Green's theorem and isolation in planar graphs. *Inf. Comput.*, 215:1–7, 2012.

34 Thomas Thierauf and Fabian Wagner. Reachability in $K_{3,3}$-free and K_5-free graphs is in unambiguous logspace. *Chicago J. Theor. Comput. Sci.*, 2014, 2014.

35 Vijay V. Vazirani. NC algorithms for computing the number of perfect matchings in $K_{3,3}$-free graphs and related problems. *Information and Computing*, 80(2):152–164, 1989.

36 Burchard von Braunmühl and Rutger Verbeek. Input-driven languages are recognized in log n space. In Marek Karpinski, editor, *Foundations of Computation Theory*, volume 158 of *Lecture Notes in Computer Science*, pages 40–51. Springer Berlin Heidelberg, 1983.

37 Klaus Wagner. Über eine Eigenschaft der ebenen Komplexe. *Math. Ann.*, 114, 1937.

A Skipped Figures

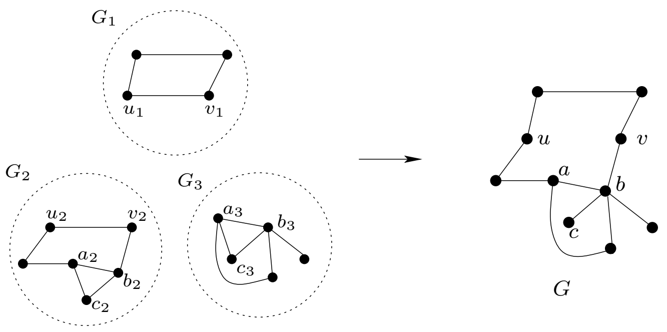

Figure 2 Graph G obtained by taking (i) 2-clique-sum of G_1 and G_2 by identifying $\langle u_1, v_1\rangle$ with $\langle u_2, v_2\rangle$ and (ii) 3-clique-sum of the resulting graph with G_3 by identifying $\langle a_2, b_2, c_2\rangle$ with $\langle a_3, b_3, c_3\rangle$.

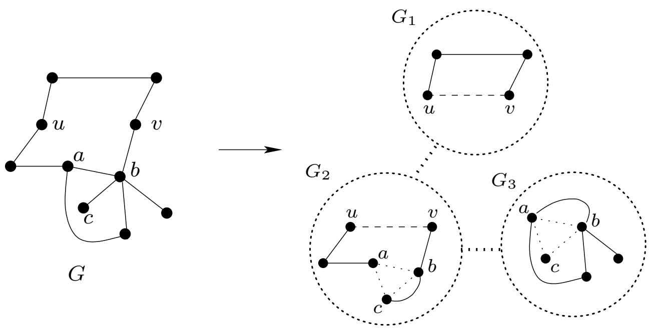

Figure 3 A graph $G \in \langle \mathcal{P}_c\rangle_3$ is shown with its component tree. Dotted circles show the nodes and dotted lines connecting them show the edges of the component tree. Dashed lines represent virtual edges and dotted triangles represent the virtual triangles, in the components.

B Skipped Proofs

Here we prove the lemmas whose proofs were skipped in the main part of the paper.

▶ Lemma 12. In a planar graph with a given planar embedding, circulation of a cycle in the clockwise orientation is the sum of circulations of the faces inside it.

Proof. We give the proof using mathematical induction on the number of faces inside the cycle.

Consider a planar graph $G = (V, E)$. For any cycle C, its circulation is denoted by $w(C)$.

Base case: The base case is a cycle containing only one face inside it. By definition of the circulation of a face, for a clockwise-oriented cycle, its circulation equals the circulation of the face.

Induction hypothesis: The circulation of a cycle having k faces is the sum of circulations of the faces inside it.

Induction step: Consider a clockwise-oriented cycle C having k faces, $f_1, f_2, \dots, f_k$, inside it. Now consider a cycle C' having the same orientation as C and with all but one face of C inside it. Without loss of generality, let this face be f_k.

We use the notation E_{ij} to show the set of edges shared between faces f_i and f_j, taken in a clockwise direction around f_i.

Denote by S_k the set of clockwise edges (w.r.t f_k) shared between f_k and other faces inside C, that is, $S_k = \cup_{i=0}^{k-1} E_{ki}$. Let S_{-k} denote the same set of edges taken in the opposite direction.

Also, we use the notation $E(C)$ to denote the set of edges taken by a cycle C. Similarly, E_k denotes the set of edges around a face f_k, taken in the clockwise direction. Similar to S_{-k}, define E_{-k}.

We can see that $E(C) \setminus E(C') = E_k \setminus S_k$, and $E(C') \setminus E(C) = S_{-k}$.

$$\begin{aligned}
w(C) &= w(C') + w(E(C) \setminus E(C')) - w(E(C') \setminus E(C)) \\
&= w(C') + w(E_k \setminus S_k) - w(S_{-k}) \\
&= w(C') + w(E_k) - w(S_k) - w(S_{-k}) \\
&= w(C') + w(E_k) - w(S_k) + w(S_k) && (w \text{ is skew-symmetric}) \\
&= \sum_{i=1}^{k-1} w(f_i) + w(E_k) && \text{(Induction hypothesis)} \\
&= \sum_{i=1}^{k-1} w(f_i) + w(f_k) && \text{(Lemma 14)}
\end{aligned}$$

Thus, the circulation of C is the sum of circulations of the faces contained in it. ◀

▶ Lemma 14. Let $G(V, E)$ be a planar graph with F being its set of inner faces in some planar embedding. For any given function on the inner faces $w' : F \to \mathbb{Z}$, a skew-symmetric weight function $w \colon \vec{E} \to \mathbb{Z}$ can be constructed in log-space such that each face $f \in F$ has circulation $w'(f)$.

Proof. The construction in [21] gives $+1$ circulation to every face of the graph and is in NC. We modify it to assign arbitrary circulations to the faces and argue that it works in log-space.

Let G^* be the dual graph of G and T^* be a spanning tree of G^*. The dual graph can be easily constructed in log-space from the planar embedding, one just needs to identify the edges present in each face. See [28, 29] for log-space construction of a spanning tree. Make the tree T^* rooted at the outer face of G. Let $E(T^*)$ denote the edges of the tree T^* (and also the corresponding edges in graph G). All the edges in $E \setminus E(T^*)$ will get weight 0. For any node f in G^* (a face in G), let T^*_f denote the subtree of T^* rooted at f. Let $w'(T^*_f)$ denote the total sum of the weights in the tree, i.e. $w'(T^*_f) = \sum_{f_1 \in T^*_f} w'(f_1)$. This function can be computed for every node in the tree T^* by the standard log-space tree traversal (see Section 3.3). For any inner face f, let e_f be the edge connecting f to its parent in the dual tree T^*. We assign the edge e_f, weight $w'(T^*_f)$ in clockwise direction (w.r.t. face f).

We claim that under this weight assignment, circulation of any inner face f is $w'(f)$. To see this, let us say $f_1, f_2, \dots, f_k$ are the children of f in the dual tree T^*. These nodes are connected with f using edges $e_{f_1}, e_{f_2}, \dots, e_{f_k}$ respectively. Now, consider the weights of these edges in the clockwise direction w.r.t. face f. For any $1 \leq i \leq k$, weight of e_{f_i} is $-w'(T^*_{f_i})$ and weight of e_f is $w'(T^*_f)$. Clearly, sum of all these weights is $w'(f)$. ◀

B.1 Complexity of the weight assignment

▶ **Lemma 17.** *The total weight given by the weighting scheme is polynomially bounded.*

Proof. The weight w_1 is polynomially bounded according to the procedure in Lemma 14.

Consider w_0. Observe that the upper bound $U_{\mathcal{T}}$ for the circulation of a cycle in $\mathcal{T}$ is actually just the sum of weights of all the edges in constant sized components, and of all the faces in planar components. By the construction given in the proof of Lemma 14, weight of any edge in the planar component is bounded by the sum of circulations of all the faces. Therefore, $U_{\mathcal{T}}$ gives the bound on the weight function w_0. Since the maximum depth of any node in $\mathcal{T}$ can be at most $O(\log|\mathcal{T}|)$, the height of $r(T)$, that is $h(r(T)) = O(\log|\mathcal{T}|)$. Also, the total number of leaves in $\mathcal{T}$ is at most $|\mathcal{T}|$.

$$U_{\mathcal{T}} = K^{h(r(\mathcal{T}))} \times l(\mathcal{T}) \leq K^{O(\log|\mathcal{T}|)} \times |\mathcal{T}| = |\mathcal{T}|^{O(\log K)}|\mathcal{T}| = |\mathcal{T}|^{O(\log K)}$$

If n is the size of the original graph G, then clearly $|\mathcal{T}| \leq n$. Therefore, $U_{\mathcal{T}} = O(n^{O(\log K)})$. Recall that K is a constant, and thus, w_0 is also polynomially bounded.

Since we use a linear combination of w_0 and w_1, the total weight function is polynomially bounded. ◀

Space complexity of the weight assignment: Section 3.3 gives a log-space construction of the working tree. We use simple log-space procedures in sequence to assign the weights in the working tree. After construction of the working tree, we use iterative log-space procedures to store the following for each node: i) the depth of the node, and ii) the number of leaves in the subtree rooted at it. Both just require a tree traversal while keeping a counter, and can be done in log-space (see Section 3.3). We use another straightforward log-space function to compute height of every node using the maximum depth amongst all the nodes. For each component node of the working tree, we store a list of all the separating sets in it and corresponding pointers for the subtrees attached at them.

Next, we iterate on the nodes of the working tree to assign the weights. For every non-planar component N, we assign a weight of $2^i \times K^{(h(N)-1)} \times l(T(N))$ to the i-th edge of component N, where $T(N)$ is the subtree rooted at N.

For every planar component N, we visit all its virtual edges/triangles. For a given virtual edge/triangle τ_i, let T_i be the subtree attached to N at τ_i. We add a circulation of $2 \times K^{h(r(T_i))} \times l(T_i)$ to all the faces adjacent to τ_i. Clearly, the procedure works in log-space. As the last step, we find the weights for the edges which would give the desired circulations of the faces. Lemma 14 shows that it can be done in log-space.

C $K_{3,3}$-free and K_5-free graphs

In this section, we will see that any $K_{3,3}$-free or K_5-free graph falls into the class $\langle \mathcal{P}_c \rangle_3$. We will show how to construct the desired component tree for any given $K_{3,3}$-free or K_5-free graph and modify it to satisfy the assumptions made in Section 3. All these constructions are in log-space.

C.1 Biconnected Graphs

If a graph G is disconnected then a perfect matching in G can be constructed by taking a union of perfect matchings in its different connected components. As connected components of a graph can be found in log-space [29], we will always assume that the given graph is connected.

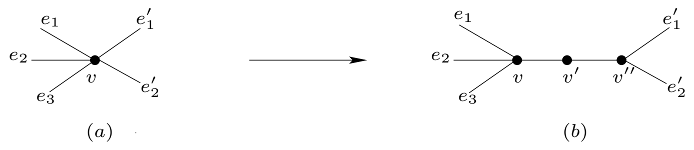

Figure 4 Vertex-split: A vertex v is split into three vertices v, v', v'', which are connected by a path. Some of the edges incident on v are transferred to v''.

Let G be a connected graph. A vertex a in G is called an *articulation point*, if its removal will make G disconnected. A graph without any articulation point is called biconnected. Let a be an articulation point in G such that its deletion creates connected components $G_1, G_2, \ldots, G_m$. It is easy to see that for G to have a perfect matching, exactly one of these components should have odd number of vertices, say G_1. Then, in any perfect matching of G, the vertex a will always be matched to a vertex in G_1. Thus, we can delete any edge connecting a to other components, and all the perfect matchings will still be preserved. It is easy to see that finding all the articulation points and for each articulation point, performing the above mentioned reduction can be done in log-space, via reachability queries [29, 34]. Thus, we will always assume that the given graph is biconnected.

C.2 Matching Preserving Operation

Vertex-split: For a graph G, we define an operation called *vertex-split*, which *preserves matchings*, as follows: Let v be a vertex and let X be the set of all the edges incident on v. Let $X_1 \sqcup X_2$ be an arbitrary partition of X. Create two new vertices v' and v'' (see Figure 4). Make the edges (v, v') and (v', v''). We call these two edges as *auxiliary edges*. For all the edges in X_2, change their endpoint v to v''. We denote this operation by vertex-split(v, X_1, X_2).

Let the modified graph be G'. One can go back to the graph G by identifying vertices v, v' and v'' and deleting auxiliary edges. This operation is *matching preserving* in the following sense.

▶ **Lemma 18.** *There is a one-one correspondence between perfect matchings of G and G'.*

Proof. Consider a perfect matching M in G, where v is matched with a vertex in X_1. It is easy to see that the matching $M' := M \cup \{(v', v'')\}$ is a perfect matching in G'. The other case when v is matched with a vertex in X_2 is similar.

Consider a perfect matching M' in G'. Removing the auxiliary edge from M' and identifying the vertices v, v' and v'' will give us a perfect matching in G. ◀

It is also easy to see that the vertex-split operation preserves biconnectivity, since v' and v'' are articulation points if and only if v is an articulation point.

C.3 Component Tree

Wagner [37] and Asano [5] gave exact characterizations of K_5-free graphs and $K_{3,3}$-free graphs, respectively. These characterizations essentially mean that any graph in these two classes can be constructed by taking 3-clique-sums of graphs which are either planar or have size bounded by 8. Recall that $\langle \mathcal{C} \rangle_k$ denotes the class of graphs obtained by taking repeated k-clique-sums of graphs starting from the graphs in class $\mathcal{C}$.

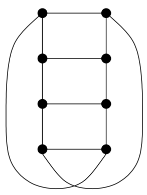

Figure 5 The four-rung Möbius ladder V_8.

▶ **Theorem 19** ([5]). *Let $\mathcal{C}$ be the class of all planar graphs together with the 5-vertex clique K_5. Then $\langle \mathcal{C} \rangle_2$ is the class of $K_{3,3}$-free graphs.*

▶ **Theorem 20** ([37, 20]). *Let $\mathcal{C}$ be the class of all planar graphs together with the four-rung Möbius ladder V_8 (Figure 5). Then $\langle \mathcal{C} \rangle_3$ is the class of K_5-free graphs.*

As mentioned in Section C.1, we can assume that the given graph is biconnected. It is known that for any given biconnected $K_{3,3}$-free graph G, its component tree can be constructed in log-space [34, Lemma 3.8]. The components here are all planar or K_5, which share separating pairs. Also, for any given biconnected K_5-free graph G, its component tree can be constructed in log-space [32, Definition 5.2, Lemma 5.3]. The components here are all planar or V_8. They can share a separating pair or a separating triplet. The planar embedding of a planar component can be computed in log-space [2, 29].

The component tree defined in [34, 32] slightly differs from our definition in Section 2.2. They have an extra component for each separating set. This component is connected to all the components which share this separating set. Moreover, whenever there is a real edge between two nodes of a separating set, it is represented by a 3-bond component (one real edge and two parallel virtual edges). The 3-bond component is also connected to the corresponding separating set node. For our purposes, these two kinds of components are not needed.

For any given biconnected $K_{3,3}$-free graph or K_5-free graph G, we start with the component tree which is constructed by [34, 32]. We show how to modify the component tree, in log-space, to have the assumptions made in Section 3.

Applying the clique-sum operations on the modified component tree will give us the actual modified graph G'. We will argue that all these modifications in G are just repeated application of the vertex-split operation (Lemma 18) in G. Thus, these are matching preserving. As mentioned earlier, from a perfect matching in G', one can get a perfect matching in G by just deleting the auxiliary vertices and edges created in the vertex-split operations.

We emphasize here that these operations might create some new pairs/triplets in the graph G' such that their removal will make the graph disconnected. But, we do not consider these new pairs/triplets as separating sets. By a separating pair/triplet we only mean those pairs/triplets which are shared by different components in the original decomposition of G.

(i) Removing "3-bond" components: For all the 3-bond components we do the following: Remove the 3-bond component. Let τ be the separating set and C_τ be the corresponding node in the component tree, where this 3-bond component is attached (a 3-bond component is always a leaf). Take an arbitrary component attached to C_τ. This component will have a virtual clique for τ. Make an appropriate real edge parallel to the existing virtual edge, in this virtual clique corresponding to τ. Note that if this component was planar, it will remain so. Moreover, it is easy to adjust the planar embedding. Clearly, this operation can be done in log-space. This does not change the actual graph G in any way.

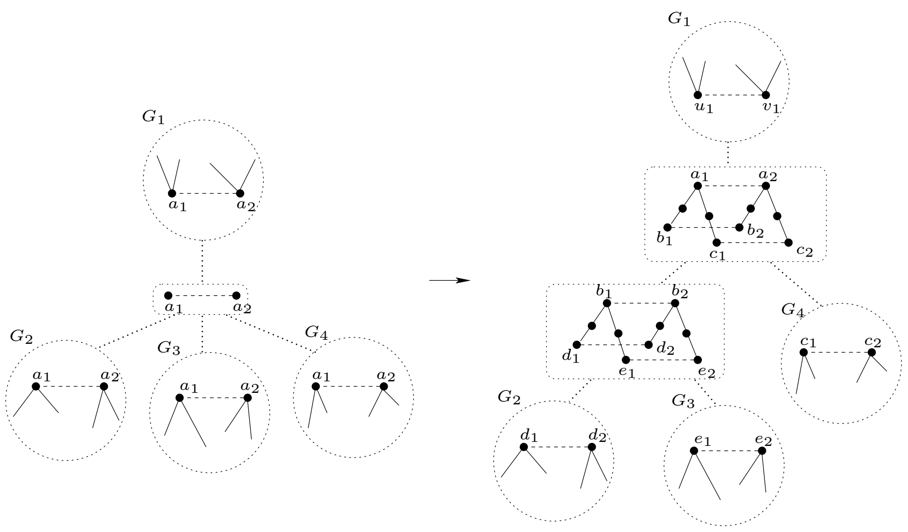

Figure 6 (a) A separating pair $\langle a_1, a_2\rangle$ is shared by four components G_1, G_2, G_3, G_4. (b) $\{\langle b_1, b_2\rangle, \langle c_1, c_2\rangle, \langle d_1, d_2\rangle, \langle e_1, e_2\rangle\}$ are copies of $\langle a_1, a_2\rangle$, which are connected by length-2 paths to form a binary tree. Different copies are shared by different components.

(ii) Any separating set is shared by at most two components: Let τ be a separating set shared by m components $G_1, G_2, \ldots, G_m$. Let the cardinality of τ be t (t can be 2 or 3). Let us define a gadget M as follows: it has three sets of nodes $\{a_i \mid 1 \leq i \leq t\}$, $\{b_i \mid 1 \leq i \leq t\}$, $\{c_i \mid 1 \leq i \leq t\}$. For each i, connect a_i with b_i by a length-2 path and also connect a_i with c_i by a length-2 path. Make 3 virtual cliques each of size t, one each for nodes $\{a_i\}_i$, $\{b_i\}_i$ and $\{c_i\}_i$. Thus, three components can be attached with M.

Now, we construct a binary tree T which has exactly $m-1$ leaves. Replace leaves of T with components $G_2, G_3, \ldots, G_m$. Replace all other nodes of T with copies of the gadget M. Further, make an edge between component G_1 and the root of T (see Figure 6). In this binary tree, any node of type M (gadget) shares its separating set $\{a_i\}_i$ with its parent node, $\{b_i\}_i$ with its left child node and $\{c_i\}_i$ with its right child node. The components $G_2, G_3, \ldots, G_m$ share their copy of τ with their respective parent nodes in the tree T. The component G_1 shares its copy of τ with the root node of T.

Doing this procedure for every separating set will ensure that every separating set is shared between at most two components. Moreover, now there is no extra component for the separating set, and the components which share a separating set are joined directly by an edge. A binary tree with $m-1$ leaves can be easily constructed in log-space (Take nodes $\{x_1, x_2, \ldots, x_{2m-3}\}$, x_i has children x_{2i} and x_{2i+1}). All the other operations here are local like deleting and creating edges and changing vertex labels. Thus it can be done in log-space.

Now, we want to argue that this operation is matching preserving for the actual graph G. Let us view this operation as a repeated application of the following operation: Partition the set of components $\{G_2, G_3, \ldots G_m\}$ in two parts, say G_1' and G_1''. Now, take a copy of the gadget M and connect it to all three components G_1, G_1' and G_1''. M shares its separating sets $\{a_i\}_i$, $\{b_i\}_i$ and $\{c_i\}_i$ with G_1, G_1' and G_1'' respectively. In the actual graph G, this operation separates the edges incident on a vertex in τ into three parts: edges from G_1, G_1' and G_1'' respectively. These three sets of edges are now incident on three different copies of the vertex. Moreover, the first copy is connected to each of the other two copies via a length-2 path. Hence, it is easy to see this as applying the vertex-split operation

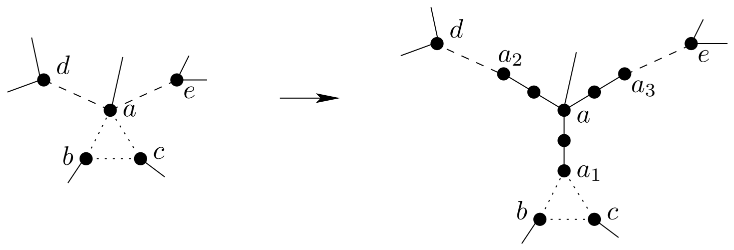

Figure 7 (a) Vertex a is a part of two separating pairs $\langle a, d\rangle$ and $\langle a, e\rangle$ and a separating triplet $\langle a, b, c\rangle$. (b) Vertex-split is applied on vertex a, 3 times, to split it into a star. The new separating sets are $\langle a_1, b, c\rangle$, $\langle a_2, d\rangle$ and $\langle a_3, e\rangle$.

(Lemma 18) twice. Now, we recursively do the same operation after partitioning the set of components G'_1 and G''_1 further. Thus, the whole operation can be seen as a vertex-split operation applied many times in the actual graph G.

Instead of a binary tree we could have also taken a tree with one root and $m-1$ leaves. This operation would also be matching preserving but the component size will depend on m. On the other hand, in our construction the new components created have size at most 15 (number of real edges is bounded by 12). Thus, the graph G' remains in class $\langle \mathcal{P}_c\rangle_3$.

(iii) Any vertex is a part of at most one separating set: Let a be vertex in a component C, where it is a part of separating sets $\tau_1, \tau_2, \dots, \tau_m$. We apply the vertex-split operation (Lemma 18) on a, m times, to split a into a star. Formally, create a set of m new nodes $a_1, a_2, \dots, a_m$. Connect each a_i with a by a path of length 2. For each i, replace a with a_i in the separating set τ_i. Let the updated separating set be τ'_i. The edge in the component tree which corresponds to τ_i, should now correspond to τ'_i. Any real edge in the component C which is incident on a, remains that way (see Figure 7). Clearly, doing this for every vertex in all the components will ensure that every vertex is a part of at most one separating set.

It is easy to see that a planar component will remain planar after this operation. The modification of the planar embedding and other changes here are local and can be done in log-space.

Now, we want to argue that this operation is matching preserving. Let us see how this operation modifies the actual graph G. Let C_i be the component which shares τ_i with C. Removal of τ_i would split the graph G into two components, say G'_i and G''_i, where G'_i is the one containing C. The above operation means that any edge in G''_i which was incident on a, is now incident on a_i instead of a. As each a_i is connected to a by a length-2 path, this operation can be seen as a repeated application of the vertex-split operation (Lemma 18). Thus, this operation is matching preserving.

Increase in the size of non-planar components: After this operation, the size of each component will grow. Let us find out the new bound on the size of constant-sized graphs. For a $K_{3,3}$-free graph, all non-planar components are of type K_5. Moreover, they are only involved in a 2-clique-sum. Hence, it can have at most $\binom{5}{2} = 10$ separating pairs. In this case, each vertex is a part of four separating pairs. Thus, each vertex will be split into a 4-star, creating 8 new vertices and 8 new edges. Totally, there will be 45 vertices and 40 real edges. Additionally, there can be some already existing real edges, at most 10. Thus, the total number of edges is bounded by 50.

For a K_5-free graph, all non-planar components are of type V_8. Note that they do not have a 3-clique, thus, can only be involved in a 2-clique-sum. In the worst case, it will have

12 separating pairs. Each vertex will be a part of 3 separating pairs. Hence, each vertex will be split into a 3-star, creating 6 new vertices and 6 new edges. Totally, there will be 56 vertices and 48 edges. Thus, together with already existing real edges, total number of real edges is bounded by 60.

(iv) A separating triplet in a planar component already forms a face: If a separating triplet does not form a face in a planar component, then the two parts of the graph – one inside the triplet and the other outside – can be considered as different components sharing this triplet. In fact, the construction in [32] already does this. When they decompose a graph with respect to a triplet, the different components one gets by deleting this triplet are all considered different components in the component tree.

ECCC ISSN 1433-8092

http://eccc.hpi-web.de