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@dgp.toronto.edu
[2]University of Illinois at Urbana-Champaign, ashug@iitk.ac.in
[3]Tel-Aviv University, rohitgurjar0@gmail.com
[4]Indian Institute of Technology Kanpur, rtewari@iitk.ac.in

March 21, 2017

Abstract

The perfect matching problem has a randomized NC algorithm, using the celebrated Isolation Lemma of Mulmuley, Vazirani and Vazirani. The Isolation Lemma implies 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.

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. 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.

Edmonds [Edm65] gave the first polynomial time algorithm for Decision-PM and Search-PM. Since then, there have been improvements in its sequential complexity [MV80], but an NC (efficient parallel) algorithm for it is not known.

A randomized NC (RNC) algorithm for Decision-PM was given by [Lov79]. Subsequently, Search-PM was also shown to be in RNC [KUW86, MVV87]. The solution of Mulmuley et al. [MVV87] was based on a beautiful lemma called the *Isolation Lemma*. They defined the notion of an isolating weight assignment on the edges of a graph.the 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 ([MVV87])**.** *For a graph $G(V, E)$, a weight assignment $\mathbf{w}\colon E \to \mathbb{N}$ is isolating if G either has a unique minimum weight perfect matching according to w or has no perfect matchings.*

The Isolation Lemma states that a random integer weight assignment (polynomially bounded) is isolating with a good probability. Other parts of the algorithm in [MVV87] 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. [ARZ99] 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 [DKR10]). A language L is in the class SPL if its characteristic function $\chi_L \colon \Sigma^* \to \{0,1\}$ can be (log-space) reduced to computing the determinant of an integer matrix.

Derandomizing the Isolation Lemma remains a challenging open question. The Isolation Lemma, as given by Mulmuley et al. [MVV87], actually works for a more general setting. It states that for any family of sets, a random weight assignment (with polynomially bounded integers) will achieve a unique minimum weight set in the family, with a good probability. As there are 2^{2^n} families of subsets possible for an n-element set, general derandomization is not possible. However, one can hope to derandomize the Isolation Lemma for families which have a succinct representation. For example, the family of perfect matchings in a graph can be represented by that graph. Arvind and Mukhopadhyay [AM08] have studied the case when the family of sets is given via a small circuit. They showed that derandomizing this version of the Isolation Lemma would imply circuit size lower bounds. While Reinhardt and Allender [RA00] 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 [DKR10, TV12], constant genus bipartite graphs [DKTV12], graphs with small number of matchings [GK87, AHT07] and graphs with small number of nice cycles [Hoa10]. 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. In a result subsequent to this work, Fenner et al. [FGT16] achieved an almost complete derandomization of the isolation lemma for bipartite graphs. They gave a deterministic construction but with quasi-polynomially large weights.

In this work, we derandomize the Isolation Lemma for a class of graphs which is a generalization of planar bipartite graphs. A forbidden graph characterization of planar graphs is well known due to Wagner [Wag37]. For a graph H, G is an H-free graph if H is not a minor of G. $K_{3,3}$ denotes the complete bipartite graph with $(3,3)$ nodes and K_5 denotes the complete graph with 5 nodes. Wagner [Wag37] showed that a graph is planar if and only if it is both $K_{3,3}$-free and K_5-free. Thus, a natural generalization of planar graphs would be a class of graphs which avoids only one of the two graphs $K_{3,3}$ and K_5 as a minor.

We derandomize the Isolation lemma for $K_{3,3}$-free bipartite graphs and K_5-free bipartite graphs. 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 1. *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 [Vaz89] and in TC^2 ($\subseteq \mathsf{NC}^3$) for K_5-free graphs [STW16]. These were the best known results for Decision-PM too. With an additional restriction of biparititeness, these counting results, together with the known NC-reduction from Search-PM to Count-PM [KMV08], implied an NC algorithm for Search-PM. To be precise, an NC^2-algorithm for $K_{3,3}$-free bipartite graphs and a TC^2-algorithm for K_5-free bipartite graphs.

A natural question was to find a direct algorithm for Search-PM via isolation, which is the main contribution of this paper. Recall that the isolation based algorithms for Decision-PM and Search-PM are in NC^2, which improves the previous bound of TC^2 for K_5-free bipartite graphs. Another limitation of the counting based 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, as has been demonstrated in the work of Fenner et al. [FGT16].

Theorem 1 together with the results of Allender et al. [ARZ99] and Datta et al. [DKR10] gives us the following complexity results about matching.

Corollary 2. *For a $K_{3,3}$-free bipartite graphs and K_5-free bipartite graphs,*

- 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 [DNTW09], reachability [TW14]. 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 [RS03]. 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. [DKR10] 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. Such a weight function ensuring nonzero circulation for every cycle has been designed by Datta et al. [DKR10] for planar graphs.

To achieve non-zero circulations in a $K_{3,3}$-free (or K_5-free) graph, we work with its 3-connected (or 4-connected) component decomposition given by [Wag37, Asa85], which can be constructed in log-space [TW14, STW16]. The crucial property of these graphs is that the components are either planar or constant-sized and two components are connected only by a shared pair/triplet of vertices. These pairs/triplets are called separating sets. The 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 separating set. Hereafter, the term vertex will mean the vertex of the graph, and the term node will mean a component in the component tree.

Any cycle can be viewed as traversing through a component and visiting other components via a vertex in a separating set and coming back to the component via another vertex in the separating set. Thus, any cycle C can be broken into its fragments contained within each of these components, which we call *projections* of C. Any such projection can be made into a cycle itself by adding virtual edges for separating pairs/triplets in the corresponding component.

Next, circulation of any cycle can be viewed 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 [DKR10, Kor09] 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* from the component tree. We first find a *center* node of the component tree which divides into balanced components, i.e., if we remove this node then each connected part has only a constant fraction of nodes of the original tree. We make this center node the root of the working tree. We apply this procedure recursively on the subtrees we get by removing the center. The construction of such a balanced working tree has been studied in context of evaluating arithmetic expressions [Bre74]. In the literature, this construction is also known as 'centroid decomposition' or 'recursive balanced separators'. Its log-space implementation is more involved.

Now, the scaling of weights in component node is done according to its level in the working tree. We use the convention that the root of the working tree has the highest level and leaves

have the lowest. To elaborate scaling, weights in a component node are multiplied by a number such that the contribution from this node surpasses the total contribution from lower level nodes. Note that a straightforward implementation of this idea can possibly multiply the total weight by a factor of $O(n)$, as we move one level higher in the tree. As the working tree has $O(\log n)$ depth, this will lead to edge weights being $n^{O(\log n)}$. So, here we need another trick. Instead of assigning weights to all the edges in a component node, we assign weights to only to edges incident on a separating set. These edges are chosen because if a cycle goes into another component via a separating set, it must use one of the edges incident on the separating set. This weighting scheme ensures that the total weight grows only by a constant multiple, when we move one level higher in the working tree.

In Section 2, we introduce the concepts of nonzero circulation, graph decomposition and the corresponding component tree. We define a class of graphs whose decomposition into 4-connected components will give either planar graphs or constant-sized graphs. To define this class, we use the notion of a clique-sum operation which can be viewed as the inverse of graph decomposition.

In Section 3, we give a log-space construction of a weight assignment with nonzero circulation for every cycle for the mentioned class of graphs, assuming that we are given a component tree of the graph. In Section 4, we argue that $K_{3,3}$-free and K_5-free graphs fall into this class of graphs, and that their component tree can be constructed in log-space.

Achieving non-zero circulation in log-space also puts directed reachability in UL [RA00, BTV09, TV12]. Thus, we get an alternate proof for the result – directed reachability for $K_{3,3}$-free and K_5-free graphs is in UL [TW14].

2 Preliminaries

For a graph $G(V, E)$, n will always denote the number of vertices in it. Let us first define the notion of 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), $w(u, v) = -w(v, u)$. Now, we define the circulation of a cycle. In this paper, a cycle would always mean a simple cycle.

Definition 3 (Circulation)**.** *Let $w\colon \vec{E} \to \mathbb{Z}$ be a skew-symmetric weight function. For any directed 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 $\mathrm{circ}_w(C)$ with respect to w 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 [TV12, 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 4 ([TV12])**.** *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

As mentioned before, clique-sum is a graph operation which can be viewed as the inverse of graph decomposition. We will define a class of graphs via the clique-sum operation which will contain $K_{3,3}$-free graphs and K_5-free graphs. We will construct a nonzero circulation weight assignment for this class of graphs, assuming that we are given a decomposition in a particular desired form.

Definition 5 (k-clique-sum)**.** *Let G_1 and G_2 be two graphs each containing a clique of size $k' \leq k$. A k-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.*

One can form clique-sums of more than two graphs by a repeated application of clique-sum operation on two graphs (see Figure 1). 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.

Definition 6. *$\langle \mathcal{P}_c \rangle_k$ is defined 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, $\langle \mathcal{P}_c \rangle_k$ is the closure of $\mathcal{P}_c$ under k-clique sums. We will construct a nonzero circulation weight assignment for the graphs which belong the class $\langle \mathcal{P}_c \rangle_3$ (with some further restrictions).

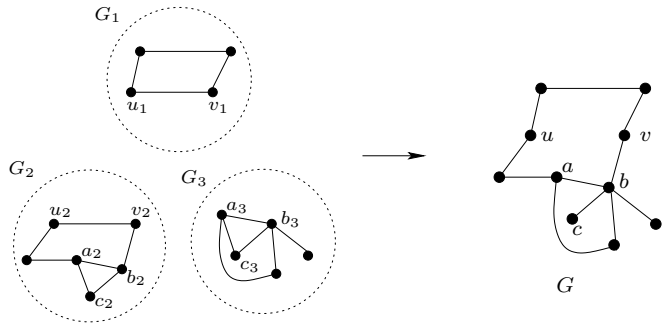

Figure 1: Graph G obtained by taking (i) 2-clique-sum of G_1 and G_2 by identifying $\langle u_1, v_1 \rangle$ with $\langle u_2, v_2 \rangle$ and (ii) 3-clique-sum of the resulting graph with G_3 by identifying $\langle a_2, b_2, c_2 \rangle$ with $\langle a_3, b_3, c_3 \rangle$.

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 Section 4.1). Thus, we will only consider clique-sum operations that involve either 2-cliques or 3-cliques. A 2-clique which is involved in a clique-sum operation will be called a separating pair. Similarly, such a 3-clique will be called a separating triplet. In general, they will be called separating sets.

In general, two clique-sums operations can be performed using two separating sets which are not completely disjoint. In particular, one can use the same separating set for many clique-sums. It will be convenient for us to assume that a clique-sum does not use a separating set which has common vertices with separating sets used in earlier clique-sums. Another assumption we take is that if a triplet is chosen from a planar component for the clique-sum We define a sub-class of $\langle \mathcal{P}_c \rangle_3$, which fulfils these assumptions.

Definition 7. *$\langle \mathcal{P}_c \rangle_3^*$ is defined to be the class of graphs constructed by taking a series of 2-clique-sums and 3-clique-sums starting from the graphs which belong to the class $\mathcal{P}_c$, where any clique-sum does not use a vertex which was involved in previous clique-sums.*

In Section 4, we will reduce the matching problem on graphs in class $\langle \mathcal{P}_c \rangle_3$ to the same on graphs in class $\langle \mathcal{P}_c \rangle_3^*$.

2.2 Component Tree

Let G be a graph in the class $\langle \mathcal{P}_c \rangle_3^*$, which has been constructed by taking a series of clique-sums on graphs $\{G_1, G_2, \ldots G_r\}$. One can define a component graph for G with r nodes – one node corresponding to each G_i. Here, G_is will be referred to as components of G. With an abuse of notation, G_i will also denote the corresponding node in the component graph. We define the

component graph $\mathtt{T}(G)$ of G recursively. Let G' and G'' be two graphs in $\langle \mathcal{P}_c \rangle_3^*$ constructed by clique-sums on $\{G'_1, G'_2, \ldots, G'_p\}$ and $\{G''_1, G''_2, \ldots, G''_q\}$, respectively. For $t = 2$ or 3, let τ' and τ'' be cliques of size t in G' and G'', respectively. Let G a t-clique-sum of G' and G'' obtained by identifying τ' and τ''. Observe that since τ' is a clique, all its vertices must be contained in G'_i for some $1 \leq i \leq p$. This is because clique-sum operation does add new edges. Moreover, by assumption, the vertices of τ' have not been used for any clique-sum operation in the construction of G'. Thus, G'_i is the unique component which contains τ'. Similarly, let G''_j be the unique component containing τ''. We define the component graph $\mathtt{T}(G)$ to be the disjoint union of $\mathtt{T}(G')$ and $\mathtt{T}(G'')$ with an edge connecting the nodes G'_i and G''_j. It is easy to see from the construction that $\mathtt{T}(G)$ is a tree. Henceforth, $\mathtt{T}(G)$ will be referred as the component tree.

Observe that if there is an edge between two nodes G_i and G_j of $\mathtt{T}(G)$ then there is a separating set τ whose copy is present in both G_i and G_j. This separating set is said to be shared by the two components. Moreover, since a separating set τ is used in a clique-sum operation only once, it is shared by only two components.

Note that the graph G can be obtained in many ways using clique-sums on different starting graphs. Each such construction gives us a different component tree. For our weight assignment construction, any such given component tree will suffice, provided the components belong to the class $\mathcal{P}_c$.

Recall that in the graph G, the vertices of a separating set τ need not form a clique, since its edges can be deleted in the clique-sum. Thus, in each component node of the component tree, 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 vertices via other components (see Figure 2). If any two vertices in a separating set have a real edge in G, then that real edge is kept in only one of the sharing components, parallel to the virtual edge. Note that while a vertex in a separating set can have its copy in two components, any real edge is present in exactly one component.

Our definition of the component tree slightly differs from the one used in the literature [HT73, TW14]. In particular, their 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 ignore this node as we have only two sharers for each separating set. We elaborate more on the differences in Section 4.3.

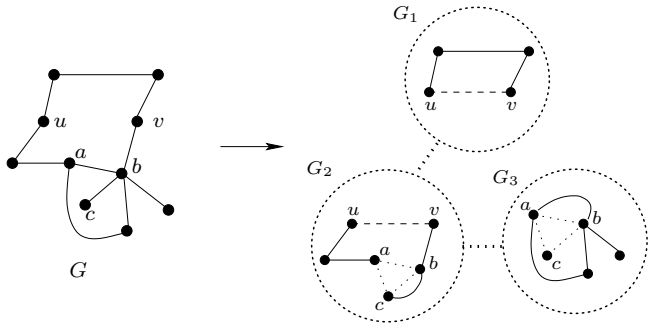

Figure 2: A graph $G \in \langle \mathcal{P}_c \rangle_3$ is shown with its component tree. Dotted circles show the nodes and dotted lines connecting them show the edges of the component tree. Dashed lines represent virtual edges and dotted triangles represent the virtual triangles, in the components.

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 a component tree is given with the components being in $\mathcal{P}_c$ and the planar embeddings of the planar components are given. We also assume that any virtual triangle in a

planar component is a face (in the given planar embedding). This assumption is without loss of generality, as the inside and outside parts of any virtual triangle can be considered as different components sharing this separating triplet. We elaborate more on this in Section 4.3. In Section 4, we reduce the matching problem for a $K_{3,3}$-free or K_5-free graph to a graph in class $\langle \mathcal{P}_c \rangle_3^*$ and argue that a component tree with the desired properties can be build in log-space.

3.1 Components of a cycle

To design the weight assignment, it will be useful to 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 component 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 3). We call these components cycles, *projections* of the original cycle.

Let us give a formal definition recursively. Let G be a graph in $\langle \mathcal{P}_c \rangle_3^*$ with $G_1, G_2, \ldots, G_r$ being its components from $\mathcal{P}_c$. Consider a directed cycle $C = \{(v_0, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_0)\}$ in the graph G. Let us say G is obtained by a clique-sum of two graphs G' and G'' via cliques τ' and τ''. Let τ be the separating set of G, which was obtained by identifying τ' and τ''. If C is completely contained in one of the graphs, say G', then we say the projection of C into G' is C itself and C has an empty projection into G''.

Now, consider the case that C has edges from both G' and G''. Then each of the graph G' and G'' must contain a continuous segment of C. This is true because τ has at most three vertices. Hence, cycle C can use one vertex of τ to go from G' to G'' and another vertex of τ to come back from G'' to G'. This argument would not work if we were dealing with 4-clique-sums and τ had 4 vertices. In that case, G' (and G'') could contain two disconnected segments of C.

Without loss of generality, for some $1 \leq i < k$, let G' contain the edges $\{(v'_i, v_{i+1}), (v_{i+1}, v_{i+2}), \ldots, (v_{k-1}, v'_0)\}$ and let G'' contain the edges $\{(v''_0, v_1), (v_1, v_2) \ldots, (v_{i-1}, v''_i)\}$, where the vertices $v'_0, v'_i \in \tau'$ and $v''_0, v''_i \in \tau''$ are the copies of $v_0, v_i \in \tau$ in G' and G'', respectively. Then consider the cycles $C' = \{(v'_i, v_{i+1}), \ldots, (v_{k-1}, v'_0), (v'_0, v'_i)\}$ and $C'' = \{(v''_0, v_1), \ldots, (v_{i-1}, v''_i), (v''_i, v''_0)\}$. Note that the edges (v'_0, v'_i) and (v''_i, v''_0) are virtual. We say that C' and C'' are the projections of C into G' and G'', respectively. And we say that C is the sum of C' and C''. Let w be any skew-symmetric weight function which assigns weight zero to all the virtual edges. Then it is easy to see that

$$\text{circ}_w(C) = \text{circ}_w(C') + \text{circ}_w(C''). \tag{1}$$

Now, we repeat the process recursively to get projections of C' and C'' into components of G' and G'', respectively. In the end, we get the projections $C_1, C_2, \ldots, C_r$ of the cycle C into components $G_1, G_2, \ldots, G_r$, respectively. Each projection is a simple cycle or possibly empty. Note that any edge in a cycle C is contained in exactly one of its projections. Moreover for any C_j, all its edges, other than the virtual edges, are contained in C. 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. The following observation is a generalization of Equation (1).

Observation 2. *Let w be any skew-symmetric weight function which assigns weight zero to all the virtual edges. Then* $\text{circ}_w(C) = \text{circ}_w(C_1) + \text{circ}_w(C_2) + \cdots + \text{circ}_w(C_r)$.

Note that for a cycle, its projections can have circulations with different signs (positive or negative). Hence, the total circulation can potentially be zero. Our idea is to ensure that one of the projections get a circulation greater than all the other projections put together. This will imply a nonzero circulation. The following observation, which follows from the construction of the cycle components, plays a crucial role in this scheme.

Observation 3. *For any cycle C in graph G, the component nodes of $\mathtt{T}(G)$ on which C has a non-empty projection form a subtree of $\mathtt{T}(G)$.*

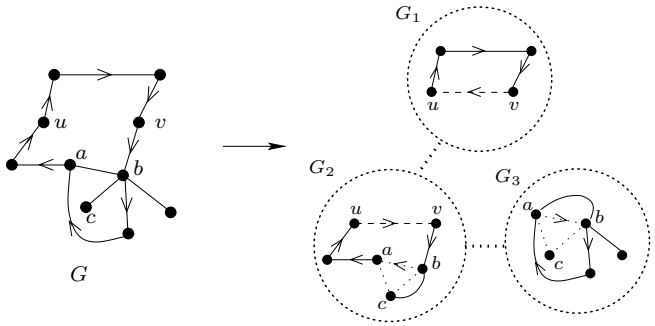

Figure 3: Breaking a cycle into its component cycles (projections) in the component tree. Notice that the original cycle and its components share the same set of *real* edges.

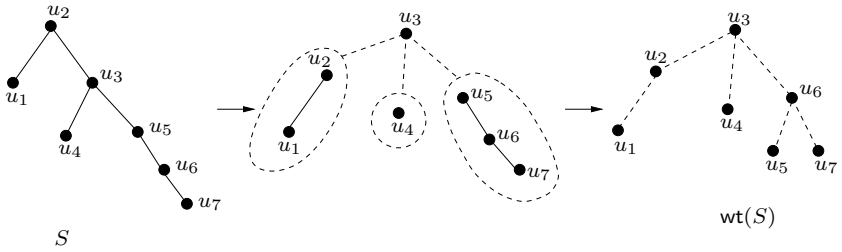

Figure 4: For a tree S, construction of the working tree $\mathsf{wt}(S)$ is shown in two steps. (i) u_3 as chosen as the center $c(S)$ and the three circles show the trees obtained after removal of u_3. (ii) u_2, u_4 and u_6 are chosen as centers of these three trees, respectively.

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 contained 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.

3.2.1 Working Tree

Our weight construction would need the component tree to have depth $O(\log n)$, when rooted at a chosen node. However, the given component tree can have large depth for all choices of the root. 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, we define its working tree $\mathsf{wt}(S)$ recursively. The definition requires the notion of a center $c(S)$ of a tree S, which can be any node of the tree S. To achieve $O(\log n)$ depth for the working tree, the center node has to be chosen in a balanced way. However, the definition of the working tree is valid for any choice of the center.

Definition 8. *For a tree S, let $\{S_1, S_2, \ldots, S_k\}$ be the set of disjoint trees obtained by deleting the node $c(S)$ from the tree S. The working tree $\mathsf{wt}(S)$ is defined to be a rooted tree whose root $r(\mathsf{wt}(S))$ is $c(S)$ and each tree $\mathsf{wt}(S_i)$ is attached to the root $c(S)$ as a subtree. In other words, each node $r(\mathsf{wt}(S_i))$ is a child of $r(\mathsf{wt}(S))$. For the base case, when S is just a node, its working tree $\mathsf{wt}(S)$ is the node itself.*

See Figure 4 for an example of the working tree. Von Braunmühl and Verbeek [vBV83], and later Limaye et al. [LMR10], gave a log-space construction of such a working tree with depth

$O(\log n)$, but in terms of well-matched strings (also see [DDN13]). In Section 3.4, 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 $i \neq j$, path(v_1, v_2) in S passes through the node $c(S) = r(\mathsf{wt}(S))$. This can be generalized to the following observation.

Observation 4. *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* path(u, v) *in the tree S passes through a.*

Depth: 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 Observation 4, we get the following.

Lemma 9. *Let S' be an arbitrary subtree of S, with its set of nodes being $\{v_1, v_2, \ldots, v_k\}$. Then there exists $i^* \in [k]$ 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 Observation 4, this least common ancestor must be present in the tree S'. But, we assumed d^* is the minimum depth value of any node in S'. Thus, we get a contradiction. □

This lemma plays a crucial role in our weight assignment construction, since from Observation 3, the component nodes of $\mathtt{T}(G)$ where cycle C has a non-empty projection form a subtree of the component tree.

3.2.2 Constructing w_0

Recall that w_0 is supposed to give nonzero circulation to all the cycles in G which have non-empty projections on at least two components. To assign weights, we work with the working tree of its component tree $\mathtt{T}(G)$. While assigning weights in a component node, we will ensure that virtual edges get zero weight. For ease of notation, let $\mathcal{T}$ denote working tree $\mathsf{wt}(\mathtt{T}(G))$. 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 $D + 1$. Let the height of a node N be given by the function $h(N)$.

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 $K > \max(2^{m+2}, 7)$. Let the number of leaves in a subtree T be given by $l(T)$. Lastly, suppose the set of subtrees attached at $r(T)$ is $\{T_1, T_2, \ldots, T_k\}$. If the component $r(T)$ shares the separating set τ_i with a component node in T_i, then the subtree T_i is said to be attached to the root $r(T)$ at τ_i.

When $r(T)$ is a constant sized graph: Let the set of (real) edges of the component $r(T)$ 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.

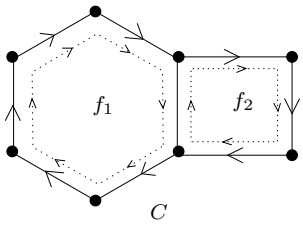

Figure 5: C is a cycle with faces f_1 and f_2 inside it. Circulation of C is equal to the sum of circulations (shown with dotted lines) of faces f_1 and f_2, since the common edge between f_1 and f_2 gets opposite orientations from f_1 and f_2.

When $r(T)$ is a planar graph: We work with the given planar embedding of the component $r(T)$. 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 13 describes how to assign weights to the edges of a planar graph to get the desired circulation for each of the inner faces.

The next lemma shows that in any planar graph, circulation of a cycle is the sum of circulations of the faces inside it.

Lemma 10 ([BTV09])**.** *Let w be a skew-symmetric weight function on the edges. In a planar component with a given planar embedding, let C be a directed cycle which is oriented clockwise. Let $f_1, f_2, \ldots, f_p$ be all the faces inside cycle C. Then* $\text{circ}_w(C) = \text{circ}_w(f_1)+\text{circ}_w(f_2)+\cdots+\text{circ}_w(f_p)$.

Proof. If an edge (u, v) belongs to cycle C, then it belongs to exactly one of the faces $f_1, f_2, \ldots, f_p$ (see Figure 5). On the other hand if an edge (u, v) lies in the interior of cycle C, then there are two faces inside C, say f_i and f_j, such that (u, v) belongs to f_i (clockwise oriented) and (v, u) belongs to f_j (clockwise oriented). As $w(u, v) = -w(v, u)$, the lemma follows. □

Assigning circulations to the faces: Now, we describe how we fix circulations for 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 $2 \times K^{h(r(T)-1)} \times l(T_i)$. Similarly, consider a separating 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 $2 \times K^{h(r(T)-1)} \times l(T_j)$. This assignment is done for all the faces adjacent to any separating set where subtrees are attached. If a face is adjacent to more than one virtual edge/triangle, then the circulation of that face is assigned to be the sum of different circulation assignments due to each virtual edge/triangle adjacent to it. Note that the circulation assigned for each face is positive.

The intuition behind this scheme is the following: As circulations of all of the faces have the same sign, they cannot cancel each other (Lemma 10). 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. First, we derive an upper bound on the circulation of any cycle completely contained in a subtree T of the working tree $\mathcal{T}$.

Lemma 11. *Let T be a subtree of the working tree $\mathcal{T}$. Let C be a cycle such that all the component nodes in $\mathcal{T}$ where C has a non-empty projection belong to T. Then the upper bound on the* $\mathrm{circ}_{w_0}(C)$ *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 on the circulation of a cycle contained in T' 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 on the circulation of cycle contained in T is $U_T = K^{h(r(T))} \times l(T)$. Recall that from Observation 2, circulation of a cycle is the sum of circulations of its projections.

Let the subtrees attached at $r(T)$ be $\{T_1, T_2, \ldots, T_k\}$. For any cycle C in T, sum of the circulations of its projections on the the component nodes of T_i can be at most U_{T_i} by induction hypothesis. This is true because the projections of C on the component nodes of T_i can be viewed as projections of a cycle contained in T_i. Hence, the total contribution to the circulation from subtrees $T_1, T_2, \ldots, T_k$ can be at most $\sum_{i=1}^{k} U_{T_i}$.

Now, we bound the contribution from the component node $r(T)$. 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 $6 \times K^{h(r(T)-1)} \times l(T_i)$. This is because the circulation of each face adjacent to the separating set connecting with T_i is $2 \times K^{h(r(T)-1)} \times l(T_i)$, and there can be at most 3 such faces. Thus,

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

Now, consider the case when $r(T)$ is a constant sized 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, we get the upper bound $U_T = K^{h(r(T))} \times l(T)$ on the circulation. □

Now, we are ready to show that any cycle which passes through at least two component nodes of $\mathtt{T}(G)$ is given a nonzero circulation by w_0.

Lemma 12. *Let C be a cycle which has non-empty projection on at least two component nodes of $\mathtt{T}(G)$. Then w_0 gives it a nonzero circulation.*

Proof. Recall Observation 2, which says that the circulation of cycle C is the sum of circulations of its projections on different components. Also recall that components with a non-empty projection of C form a subtree S_C in the component tree $\mathtt{T}(G)$ (Observation 3). From Lemma 9, we can find a node $N^* \in S_C$ such that all other nodes in S_C are its descendants in the working tree $\mathcal{T}$. Thus, N^* is the unique minimum depth component on which C has a non-empty projection. Let C^* be the projection of C into N^*. Now, the following two things suffice for proving nonzero circulation of C: (i) circulation of C^* is nonzero, and (ii) it is larger than the sum of circulations of all other projections of C.

Let N^* be the root of a subtree T in the working tree. Let the subtrees attached at $r(T)$ $(= N^*)$ 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. First of all, C^* 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). Thus, the circulation of C^* is nonzero, because the weights given to the real edges are powers of 2. Now, recall that the minimum weight of any edge in $r(T)$ is $2 \times \sum_{i=1}^{k} U_{T_i}$. From Lemma 11, U_{T_i} is the upper bound on the total circulation contribution from component nodes in T_i. Thus, circulation of C^* is larger than the contribution from components in $T_1, T_2, \ldots, T_k$.

Case 2: when $r(T)$ is a planar component. The important observation here is that in a planar graph, circulation of a cycle is the sum of circulations of the faces inside it (Lemma 10).

Since C has a non-empty projection in at least two component nodes, it must pass through at least one of the subtrees attached at $r(T)$, say T_i, and it must go through the separating set τ_i. Hence, the cycle 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, 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. □

3.2.3 Assigning 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 13 describes this procedure for any planar graph.

Lemma 13 ([Kor09])**.** *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. The construction in [Kor09] 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. The construction goes via the dual of the graph.

The dual of a planar graph G (with a given planar embedding) is a graph G^* whose vertices correspond to the faces of G and two vertices in G^* are connected by an edge if the corresponding two faces in G share a common edge. It is easy to see that there is a one-to-one correspondence between the edges of G and G^*. The dual graph can be easily constructed in log-space from the planar embedding, one just needs to find out the edges present in each face.

Let T^* be a spanning tree of G^*. See [NT95, Rei08] for a log-space construction of a spanning tree. Make the tree T^* rooted at the vertex which correspond to the outer face of G. Let $E(T^*)$ denote the set of edges in G whose corresponding edges in G^* belong to the tree T^*. All the edges in $E \setminus E(T^*)$ will be assigned 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 3.4). For any inner face f, let e_f be the edge such that the corresponding dual edge connects f to its parent in the dual tree T^*. We assign weight $w'(T^*_f)$ to the edge e_f, in the direction which we get from the clockwise orientation of face f.

We claim that under this weight assignment, circulation of any inner face f is $w'(f)$. To see this, let us say $f_1, f_2, \ldots, f_k$ are the children of f in the dual tree T^*. These nodes are connected with f using edges $e_{f_1}, e_{f_2}, \ldots, e_{f_k}$ respectively. Now, consider the weights of these edges in the directions we get from the clockwise orientation of 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 $w'(f)$. □

We need to modify the scheme of Lemma 13 a bit, in order to apply it to the planar components of the component tree $\mathtt{T}(G)$. The reason is that this scheme can assign weight to any edge in the given graph, while we have to give weight zero 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). Then, we apply the procedure from Lemma 13 to get the desired circulations of faces. After undoing the collapse, the circulations of the faces will not change.

3.2.4 Constructing w_1

Recall that the weight function w_1 needs to ensure nonzero circulation for any cycle contained within a single component. To construct w_1 for planar components, we assign $+1$ circulation to every face using Lemma 13. Since, the circulation of a cycle is the sum of circulations of the faces inside it (Lemma 10), this would ensure nonzero circulation for every cycle within the planar component. This construction has been used in [Kor09] for bipartite planar graphs. [TV12] also gives a log-space construction which ensures nonzero circulation for all cycles in a planar graph, using Green's theorem. For the constant sized components, w_0 already ensures that each cycle within the component has a non-zero circulation. Therefore, we set $w_1 = 0$ for constant sized components.

Now, we use a combination of w_0 and w_1 such that they do not interfere with each other. That is, we define

$$w = w_0 + w_1 \times (U_{\mathcal{T}} + 1).$$

Since, $U_{\mathcal{T}}$ is an upper bound on $\mathrm{circ}_{w_0}(C)$ for any cycle C, we can see that $\mathrm{circ}_w(C)$ is nonzero if one of $\mathrm{circ}_{w_0}(C)$ and $\mathrm{circ}_{w_1}(C)$ is nonzero. This together with Lemma 12 gives us the following.

Lemma 14. *For any cycle C in G, $\mathrm{circ}_w(C)$ is non-zero.*

3.3 Complexity of the weight assignment

In this section, we argue that the weights given by the scheme of Section 3.2 are polynomially bounded and the weight-construction procedure can be done in log-space.

Lemma 15. *The total weight given by the weight function w is polynomially bounded.*

Proof. Since the weight function w_1 assigns circulation 1 to each face of planar component, the weight given to any edge in the procedure of Lemma 13 is bounded by number of faces, which is $O(n)$.

Now, 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 circulations of all the faces in planar components. By the construction given in the proof of Lemma 13, 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 of any edge given by w_0. Recall that we construct a working tree where the maximum depth of any node can be at most $O(\log n)$. Thus, $h(r(T)) = O(\log n)$. Also, the total number of leaves in $\mathcal{T}$ is at most $O(n)$.

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

Recall that K is a constant, and thus, w_0 is also polynomially bounded. Since w_0, w_1 and $U_{\mathcal{T}}$ are polynomially bounded, same is true for the combined weight function $w_0 + w_1 \times (U_{\mathcal{T}} + 1)$. □

Space complexity of the weight assignment: Section 3.4 gives a log-space construction of the working tree with depth $O(\log n)$. 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 of the working tree: 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 3.4). 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 13 shows that it can be done in log-space.

3.4 Construction of the Working Tree

Now, we describe a log-space construction of a $O(\log n)$-depth working tree. The idea is obtained from the construction of [LMR10, 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 $\mathsf{wt}(S)$. Let $S_1, S_2, \ldots, S_k$ are the connected components obtained by removing the node $c(S)$ from S. Then for each S_i, we recursively build the working tree $\mathsf{wt}(S_i)$ and connect it to the root of $\mathsf{wt}(S)$, as a subtree.

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, with this definition of the center, the depth of the working tree would be $O(\log n)$. However, it is not clear if the recursive procedure of finding centers for each resulting subtree 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 [Lin92]: 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 (in case e is the only edge incident on u, leave using e). This traversal ends at r with every edge being traversed exactly twice. For any node u, one can check the tree traversal sequence to find out the unique neighbor of u which was visited before the first visit to u. This will be the parent of u. Once we have the child-parent relations, we can do a similar tree traversal to find the number of nodes in a subtree rooted at u, or the total weight of the nodes in a subtree for some weight assignment on the nodes.

Now, we come back to defining centers for a tree. 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 components we get after deleting the center will always

be of the form $S_{v,v'}$, for some nodes $v, v' \in S$. Thus, we give a definition of the center only 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 [Jor69]). 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 constructing the working tree.

Lemma 16. *For any tree S, its working tree $\mathsf{wt}(S)$, with the center defined as above, 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 described above, 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 $K_{3,3}$-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 $\langle \mathcal{P}_c \rangle_3$ (see Section 2.1 for the definition). We will further show that the matching problem for a $K_{3,3}$-free or K_5-free graph can be reduced to a graph in class $\langle \mathcal{P}_c \rangle_3^*$ in log-space. We will also argue that a component tree for this graph with the desired properties (mentioned in Section 3) can be constructed in log-space.

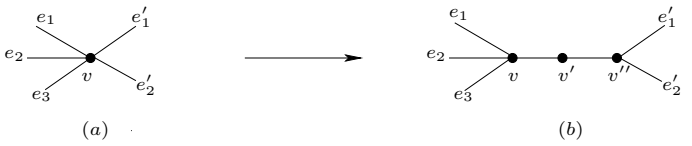

Figure 6: Vertex-split: A vertex v is split into three vertices v, v', v'', which are connected by a path. Some of the edges incident on v are transferred to v''.

4.1 Biconnected Graphs

First we argue that for the matching problem, one can assume that the given graph is biconnected (defined below). 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 [Rei08], we will always assume that the given graph is connected.

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 [Rei08, TW14]. Thus, we will always assume that the given graph is biconnected.

4.2 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 6). 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). It is *matching preserving* in the following sense.

Lemma 17. *Let G' be the graph obtained by applying vertex-split(v, X_1, X_2) operation on G. Then G has a perfect matching if and only if G' has a perfect matching.*

Proof. Let v' and v'' be the two new vertices created in the vertex-split operation. 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.

For the other direction, consider a perfect matching M' in G'. Then M' must contain exactly one of the two edges (v, v') and (v', v''). Removing this 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.

4.3 Component Tree

Wagner [Wag37] and Asano [Asa85] 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

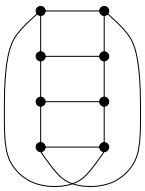

Figure 7: The four-rung Möbius ladder V_8.

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 $\mathcal{C}$.

Theorem 5 ([Asa85])**.** *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 6 ([Wag37, Khu88])**.** *Let $\mathcal{C}$ be the class of all planar graphs together with the four-rung Möbius ladder V_8 (Figure 7). Then $\langle \mathcal{C} \rangle_3$ is the class of K_5-free graphs.*

Note that the class $\langle \mathcal{P}_c \rangle_3$ subsumes both the classes mentioned in the theorems. As mentioned in Section 4.1, we can assume that the given graph is biconnected. Thierauf and Wagner [TW14, Lemma 3.8] gave a log-space construction of a component tree of a biconnected $K_{3,3}$-free graph. The components here are all planar or K_5, which share separating pairs. On similar lines, Straub, Thierauf and Wagner [STW16, Definition 5.2, Lemma 5.3] gave a log-space construction of a component tree of a biconnected K_5-free graph. 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 [AM04, Rei08].

The component tree defined in [TW14, STW16] slightly differs from our definition in Section 2.2. Our definition of the component tree is valid only for the graphs of class $\langle \mathcal{P}_c \rangle_3^*$ (Definition 7). Recall that in this class of graphs, a separating set is used in a clique-sum operation only once. Hence, a separating set is shared by only two components. In our component tree, two component nodes are connected by an edge if they share a separating set. On the other hand, for a graph in $\langle \mathcal{P}_c \rangle_3$, a separating set can be used for many clique-sum operations and thus can be shared by many components. The component tree defined by [TW14, STW16] has an extra node for each separating set. This node is connected to all the components which share this separating set (See Figure 8(a)).

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 [TW14, STW16]. Then we use some matching preserving operations to get a graph $G*$ which belongs to $\langle \mathcal{P}_c \rangle_3^*$. Instead of the original graph G, the operations are done on the component nodes of the component tree given by [TW14, STW16]. This will give us a new component tree which is the way we defined it in Section 2.2. We also argue that this procedure can be done in log-space.

Applying the clique-sum operations on the modified component tree will give us the modified graph $G*$. We will argue that all these modifications in G are just repeated application of the vertex-split operation (Section 4.2) in G. Thus, these are matching preserving. As seen in Lemma 17, 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.

Another minor difference from [TW14, STW16] is that whenever there is a real edge between two nodes of a separating set, it is represented by a special component node in the component tree. It is called a 3-bond component which has one real edge and two parallel virtual edges. The 3-bond component is connected to the corresponding separating set node. On the other hand, we keep the real edge in one of the components containing the separating set. For our purposes, the 3-bond component is not needed and we will simply remove it.

(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 N_τ be the corresponding node

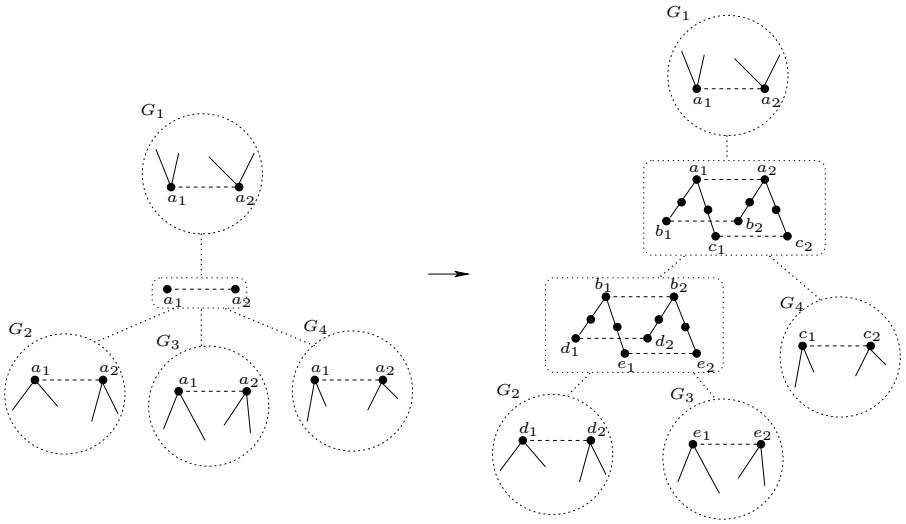

Figure 8: (a) A separating pair $\langle a_1, a_2 \rangle$ is shared by four components G_1, G_2, G_3, G_4. (b) $\{\langle b_1, b_2 \rangle, \langle c_1, c_2 \rangle, \langle d_1, d_2 \rangle, \langle e_1, e_2 \rangle\}$ are copies of $\langle a_1, a_2 \rangle$, which are connected by length-2 paths to form a binary tree. Different copies are shared by different components.

in the component tree, where this 3-bond component is attached (a 3-bond component is always a leaf). Take an arbitrary component node attached to N_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.

(ii) Any separating set is shared by at most two components: Let us say, in the component tree constructed by [TW14, STW16], there is a node for the separating set τ and it is connected with q component nodes $G_1, G_2, \ldots, G_q$. In other words, the separating set τ is shared by these q components. 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 $q-1$ leaves. Replace leaves of T with components $G_2, G_3, \ldots, G_q$. Replace all other nodes of T with copies of the gadget M (see Figure 8). Further, make an edge between component G_1 and the root of T. 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 $G_2, G_3, \ldots, G_q$ 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 node for the separating set, and the components which share a separating set are joined directly by an edge. A binary tree with $q-1$ leaves can be easily constructed in log-space (Take nodes $\{x_1, x_2, \ldots, x_{2q-3}\}$, 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. Basically, we will show that this operation is a series of vertex-splits (Section 4.2). Let us view this operation as a repeated application of the following operation: Partition the set of components $\{G_2, G_3, \ldots G_q\}$ in two parts, say G_1' and G_1''. Now, take a copy of the gadget M and connect it

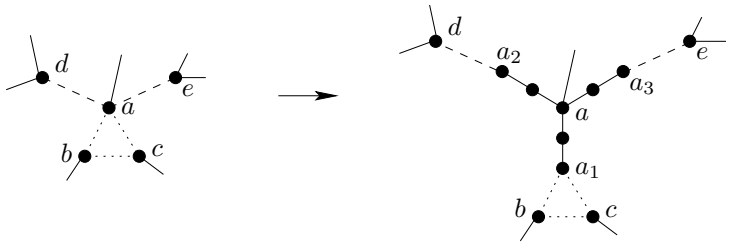

Figure 9: (a) Vertex a is a part of two separating pairs $\langle a, d\rangle$ and $\langle a, e\rangle$ and a separating triplet $\langle a, b, c\rangle$. (b) Vertex-split is applied on vertex a, 3 times, to split it into a star. The new separating sets are $\langle a_1, b, c\rangle$, $\langle a_2, d\rangle$ and $\langle a_3, e\rangle$.

to all three components G_1, G_1' and G_1''. Let us say 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, i.e., a_i, b_i and c_i. Moreover, the first copy is connected to each of the other two copies via a length-2 path. Hence, it is easy to view this as applying the vertex-split operation 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 $q-1$ leaves. This operation would also be matching preserving but the component size will depend on q. 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 modified graph remains in class $\langle \mathcal{P}_c \rangle_3$.

(iii) Any vertex is a part of at most one separating set: Recall that in the definition of class $\langle \mathcal{P}_c \rangle_3^*$ (Definition 7), we do not allow a vertex to be a part of two different clique-sums. On the other hand, this is allowed for a graph in $\langle \mathcal{P}_c \rangle_3$. Thus, in the component tree constructed by [TW14, STW16], there might be a vertex in the component nodes which is a part of many separating sets. We use the vertex-split operation to modify the components such that each vertex is a part of at most one separating set.

Let a be vertex in a component N, where it is a part of separating sets $\tau_1, \tau_2, \ldots, \tau_q$. We apply the vertex-split operation on a, q times, to split a into a star. Formally, create a set of q new nodes $a_1, a_2, \ldots, a_q$. Connect each a_i with a by a path of length 2 (see Figure 9). 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 N which is incident on a, remains that way. 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. The only thing which changes is that some of the edges in G which were incident on a are now incident on a_i. As each a_i is connected to a by a length-2 path, this operation can be viewed as a vertex-split. 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) Separating triplet in a planar component already forms a face: Another assumption we took in the beginning of Section 3 was that in a planar component the virtual triangle for any separating triplet is 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 [STW16] 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. Thus, in their planar components there are no edges inside a virtual triangle, i.e., it forms a face.

5 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. In fact, Kane, Lovett and Rao have recently shown that it is not possible to achieve nonzero circulation for all cycles in a complete bipartite graph with sub-exponential weights. It needs to be investigated what is the most general class of graphs where nonzero circulation can be achieved for all cycles with polynomial weights.

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 [GL99, KMV08]. Thus, Search-PM for small genus bipartite graphs is in NC [KMV08]. 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.

6 Acknowledgements

We thank Arpita Korwar for various helpful discussions.

References

[AHT07] Manindra Agrawal, Thanh Minh Hoang, and Thomas Thierauf. The polynomially bounded perfect matching problem is in NC^2. In *24th Symposium on Theoretical Aspects of Computer Science (STACS)*, Lecture Notes in Computer Science 4393, pages 489–499. Springer-Verlag, 2007.

[AM04] Eric Allender and Meena Mahajan. The complexity of planarity testing. *Information and Computation*, 189(1):117 – 134, 2004.

[AM08] Vikraman Arvind and Partha Mukhopadhyay. Derandomizing the isolation lemma and lower bounds for circuit size. In *Approximation, Randomization and Combinatorial*

Optimization. Algorithms and Techniques, 11th International Workshop, APPROX, and 12th International Workshop, RANDOM, pages 276–289, 2008.

[ARZ99] Eric Allender, Klaus Reinhardt, and Shiyu Zhou. Isolation, matching, and counting uniform and nonuniform upper bounds. *Journal of Computer and System Sciences*, 59(2):164–181, 1999.

[Asa85] Takao Asano. An approach to the subgraph homeomorphism problem. *Theoretical Computer Science*, 38(0):249 – 267, 1985.

[Bre74] Richard P. Brent. The parallel evaluation of general arithmetic expressions. *Journal of the ACM*, 21(2):201–206, April 1974.

[BTV09] Chris Bourke, Raghunath Tewari, and N. V. Vinodchandran. Directed planar reachability is in unambiguous log-space. *ACM Transanctions on Computation Theory*, 1(1):4:1–4:17, February 2009.

[DDN13] 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.

[DKR10] Samir Datta, Raghav Kulkarni, and Sambuddha Roy. Deterministically isolating a perfect matching in bipartite planar graphs. *Theory of Computing Systems*, 47:737–757, 2010.

[DKTV12] Samir Datta, Raghav Kulkarni, Raghunath Tewari, and N. V. Vinodchandran. Space complexity of perfect matching in bounded genus bipartite graphs. *Journal of Computer and System Sciences*, 78(3):765–779, 2012.

[DNTW09] 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.

[Edm65] Jack Edmonds. Path, trees, and flowers. *Canadian Journal of Mathematics*, 17:449467, 1965.

[FGT16] Stephen A. Fenner, Rohit Gurjar, and Thomas Thierauf. Bipartite perfect matching is in quasi-NC. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 754–763, 2016.

[GK87] 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.

[GL99] Anna Galluccio and Martin Loebl. On the theory of Pfaffian orientations. I. perfect matchings and permanents. *The Electronic Journal of Combinatorics*, 6, 1999.

[Hoa10] Thanh Minh Hoang. On the matching problem for special graph classes. In *IEEE Conference on Computational Complexity*, pages 139–150. IEEE Computer Society, 2010.

[HT73] John E. Hopcroft and Robert Endre Tarjan. Dividing a graph into triconnected components. *SIAM Journal on Computing*, 2(3):135–158, 1973.

[Jor69] Camille Jordan. Sur les assemblages de lignes. *Journal für die reine und angewandte Mathematik*, 70:185–190, 1869.

[Khu88] S. Khuller. *Parallel Algorithms for K_5-minor Free Graphs*. Cornell University, Department of Computer Science, 1988.

[KMV08] 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.

[Kor09] Arpita Korwar. Matching in planar graphs. Master's thesis, Indian Institute of Technology Kanpur, 2009.

[KUW86] Richard M. Karp, Eli Upfal, and Avi Wigderson. Constructing a perfect matching is in random NC. *Combinatorica*, 6(1):35–48, 1986.

[Lin92] 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.

[LMR10] Nutan Limaye, Meena Mahajan, and B. V. Raghavendra Rao. Arithmetizing classes around NC^1 and L. *Theory of Computing Systems*, 46(3):499–522, 2010.

[Lov79] László Lovász. On determinants, matchings, and random algorithms. In *FCT*, pages 565–574, 1979.

[MV80] Silvio Micali and Vijay V. Vazirani. An $O(\sqrt{|V|}|E|)$ algorithm for finding maximum matching in general graphs. In *21st Annual Symposium on Foundations of Computer Science, Syracuse, New York, USA, 13-15 October 1980*, pages 17–27, 1980.

[MVV87] Ketan Mulmuley, Umesh V. Vazirani, and Vijay V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7:105–113, 1987.

[NT95] Noam Nisan and Amnon Ta-Shma. Symmetric logspace is closed under complement. *Chicago Journal of Theoretical Computer Science*, 1995, 1995.

[RA00] Klaus Reinhardt and Eric Allender. Making nondeterminism unambiguous. *SIAM Journal on Computing*, 29(4):1118–1131, 2000.

[Rei08] Omer Reingold. Undirected connectivity in log-space. *Journal of the ACM*, 55:17:1–17:24, September 2008.

[RS03] 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.

[STW16] Simon Straub, Thomas Thierauf, and Fabian Wagner. Counting the number of perfect matchings in K_5-free graphs. *Theory of Computing Systems*, 59(3):416–439, 2016.

[TV12] Raghunath Tewari and N. V. Vinodchandran. Green's theorem and isolation in planar graphs. *Information and Computation*, 215:1–7, 2012.

[TW14] Thomas Thierauf and Fabian Wagner. Reachability in $K_{3,3}$-free and K_5-free graphs is in unambiguous logspace. *Chicago Journal of Theoretical Computer Science*, 2014, 2014.

[Vaz89] 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.

[vBV83] Burchard von Braunmühl and Rutger Verbeek. Input-driven languages are recognized in log n space. In *Fundamentals of Computation Theory, Proceedings of the 1983 International FCT-Conference, Borgholm, Sweden, August 21-27, 1983*, pages 40–51, 1983.

[Wag37] Klaus Wagner. Über eine Eigenschaft der ebenen Komplexe. *Mathematische Annalen*, 114, 1937.