Theory Comput Syst (2016) 59:416–439
DOI 10.1007/s00224-015-9645-1

Counting the Number of Perfect Matchings in K_5-Free Graphs

Simon Straub[1] · **Thomas Thierauf**[2] ·
Fabian Wagner[1]

Published online: 15 July 2015

Abstract Counting the number of perfect matchings in graphs is a computationally hard problem. However, in the case of planar graphs, and even for $K_{3,3}$-free graphs, the number of perfect matchings can be computed efficiently. The technique to achieve this is to compute a *Pfaffian orientation* of a graph. In the case of K_5-free graphs, this technique will not work because some K_5-free graphs do not have a Pfaffian orientation. We circumvent this problem and show that the number of perfect matchings in K_5-free graphs can be computed in polynomial time. We also parallelize the sequential algorithm and show that the problem is in TC^2. We remark that our results generalize to graphs without singly-crossing minor.

Keywords Perfect matching · Counting · Complexity

1 Introduction

Counting the number of perfect matchings in graphs is a computationally hard problem. Valiant [23] showed that it is complete for the class of NP-counting problems, #P. Nonetheless for some classes of restricted graphs the problem can be solved efficiently. Kasteleyn [14] showed that the number of perfect matchings in *planar* graphs can be computed in polynomial time. The technique is to compute a *Pfaffian*

Research supported by DFG grant TH 472/4-1

✉ Thomas Thierauf
thomas.thierauf@uni-ulm.de

1 Ulm University, Ulm, Germany

2 Aalen University, Aalen, Germany

orientation of the given graph. Such an orientation assures that each term in the Pfaffian of the Tutte matrix of the graph has the same sign. Every term in the Pfaffian corresponds to a perfect matching. Hence the Pfaffian yields the number of perfect matchings. Since the Pfaffian can be computed in polynomial time [1, 4, 10], we get an efficient counting algorithm.

Recall that K_5 denotes the complete graph with five vertices and $K_{3,3}$ is the complete bipartite graph where each of the two independent sets has three vertices. Planar graphs can be characterized as the graphs that contain neither the graph $K_{3,3}$ nor the graph K_5 as a minor. Graphs with this property are called $K_{3,3}$-free and K_5-free. Little [17] extended Kasteleyns result to $K_{3,3}$-free graphs. He showed that also $K_{3,3}$-free graphs have a Pfaffian orientation which can be computed efficiently. This yields again an efficient counting algorithm for the number of perfect matchings in $K_{3,3}$-free graphs.

Since then, it has been a challenging open problem to compute the number of perfect matchings in K_5-free graphs efficiently. In this paper, we solve this problem. The Pfaffian orientation technique is not applicable here because some graphs, like the $K_{3,3}$, have no such orientation.

We solve this problem by decomposing a given graph G into its triconnected components. It is known that the non-planar triconnected components of G are either the Möbius ladder M_8, or their decomposition into 4-connected components yields only planar components. Such a decomposition can be computed in logspace [7, 22]. The number of perfect matchings in planar components can be computed efficiently by Kasteleyns algorithm. The major task to be accomplished here is to calculate the overall number of perfect matchings from them. The difficulty is, that the graph is decomposed along separating pairs or triples which have a copy in every component they split off. However, when counting perfect matchings, the nodes of the separating pairs and triples should be matched only in one of the components. We use gadgets to replace components such that every node is matched only in one component.

Valiant [24] introduced the notion of *holographic reductions* between counting problems. In classical reductions there is often a very direct one-to-one correspondence between the solutions of the instances. For example, in the classical reduction from the Boolean satisfiability problem to the clique problem, every satisfying assignment of the formula corresponds to precisely one maximum clique in the constructed graph. In holographic reductions, there is no such obvious correspondence between the solutions, merely their numbers are equal.

Valiant showed several such reductions to the problem of counting perfect matchings in planar graphs. Thereby he showed various counting problems to be in ¶. Valiant maps a problem instance I of a counting problem to a planar graph G, called a *matchgrid*, such that the number of solutions of I equals the number of weighted perfect matchings in G. The matchgrid in turn consists of gadgets, called *matchgates*, which are defined based on individual components of the instance I.

We use similar gadgets as Valiant in our algorithm. However, our algorithm uses dynamic programming and can therefore be seen as a Turing reduction to planar perfect matching. Hence it seems that our algorithm does not fit into Valiant's framework of holographic reductions.

Vazirani [25] showed that the algorithms of Kasteleyn and Little can be parallelized: the number of perfect matchings in $K_{3,3}$-free graphs can be computed within the circuit class NC^2. In Section 5, we show that our algorithm can be parallelized as well: the number of perfect matchings in K_5-free graphs can be computed within the circuit class TC^2.

Constructing a perfect matching is another issue. It is known that perfect matchings can be constructed in polynomial time [8]. Despite the efficient parallel counting algorithm for planar graphs it remains an intriguing open question if a planar perfect matching can also be efficiently *constructed in parallel.*

When we consider graphs that are additionally *bipartite* one can do more: Kulkarni, Mahajan, and Varadarajan [15] showed that if a class of bipartite graphs is closed under edge deletion and the number of perfect matchings can be computed in NC, then a perfect matching can also be constructed in NC. The result applies to planar and $K_{3,3}$-free graphs, and now, by our result, also to K_5-free graphs.

Albeit our discussion is dedicated to K_5-free graphs, our technique works for a larger class of graphs. Elberfeld, Jakoby, and Tantau [9] showed that the number of perfect matchings of graphs with *bounded treewidth* can be computed in logspace. Hence, in our decomposition of a graph into its 4-connected components, instead of being planar these components could as well have bounded treewidth. Then we could still compute the number of perfect matchings. Robertson and Seymour [20] considered the class of graphs that do not have a *singly-crossing* minor. This class contains all $K_{3,3}$-free and K_5-free graphs. They showed that the 4-connected components of such graphs are planar or have bounded treewidth. Hence our results for counting and constructing perfect matchings in TC^2 generalize to graphs without singly-crossing minor. Independently of our work, Curticapean [5] obtained a polynomial-time algorithm for counting perfect matchings in graphs without singly-crossing minor.

After some preliminaries, the decomposition of a K_5-free graph and its properties are explained. Subsequently it is shown how to obtain the number of perfect matchings of the input graph based on this decomposition.

2 Definitions and Notations

We consider undirected graphs $G = (V, E)$ in this paper. In case of a weighted graph, there is a weight function $w : E \to \mathbb{R}$ on the edegs of G. For $U \subseteq V$ let $G - U$ be the *induced subgraph* of G on $V - U$.

Let T be a rooted tree. The subtree of T with root v is denoted by $T(v)$ and $|T(v)|$ is the number of nodes in $T(v)$.

Let $G = (V, E)$ be a graph and $S \subseteq V$ with $|S| = k$. We call S a k-separating set, if $G - S$ is not connected. For $u, v \in V$ we say that S *separates* u *from* v *in* G, if $u \in S$, $v \in S$, or u and v are in different components of $G - S$. For sets of vertices $V_1, V_2 \subseteq V$ we say that S *separates* V_1 *from* V_2 *in* G, if S separates every $v_1 \in V_1$ from every $v_2 \in V_2$.

A k-separating set is called *articulation point* (or *cut vertex*) for $k = 1$, *separating pair* for $k = 2$, and *separating triple* for $k = 3$.

A graph G is *k-connected* if it contains no $(k-1)$-separating set. Hence a 1-connected graph is simply a connected graph. A 2-connected graph is also called *biconnected*, a 3-connected graph is one type of *triconnected* graphs, other types will be introduced later.

Let S be a k-separating set in a k-connected graph G. Let G' be a connected component in $G - S$. A *split graph* or a *split component of* S in G is the induced subgraph of G on vertices $V(G') \cup S$, where we add *virtual edges* between all pairs of vertices in S. Note that the vertices of a separating set S occur in several split graphs of G.

Let H be a fixed graph. We say that a graph G is *H-free* if G has no minor isomorphic to H. In particular, we consider $K_{3,3}$-free and K_5-free graphs in this paper. Graphs which are both, $K_{3,3}$-free *and* K_5-free, are precisely the planar graphs [16].

Let $G = (V, E)$ be an undirected graph, $|V| = n$. A *perfect matching* in G is a set $M \subseteq E$ such that every vertex of G occurs in exactly one edge of M. A consequence is that $|M| = n/2$.

The number of perfect matchings in G is denoted by #pm(G). We extend the notion to weighted graphs. Let w be a weight function. The *weighted number of perfect matchings* in G is defined as

$$\#\mathrm{pm}(G) = \sum_{M} \prod_{(u,v)\in M} w(u, v)\,,$$

where the sum is over all perfect matchings M in G. Weight function $w(u, v) = 1$ for all $(u, v) \in E$ is equivalent to the unweighted case.

For the definition of complexity classes we refer the reader to any standard text book, for example [26]. In short, circuit classes NC^k, AC^k, and TC^k consist of polynomial-size circuits of depth $O(\log^k n)$, where n is the length of the input. NC-circuits have fan-in two and-or-gates, whereas AC-circuits have unbounded fan-in and-or-gates. TC-circuits have unbounded fan-in gates as AC, and additionally threshold-gates. It is known that for all $k \geq 0$

$$\mathrm{NC}^k \subseteq \mathrm{AC}^k \subseteq \mathrm{TC}^k \subseteq \mathrm{NC}^{k+1}\,.$$

The circuit classes are interleaved by logspace complexity classes. The class L stands for problems recognized by logspace bounded Turing machines, the class NL is its nondeterministic counterpart. The function class #L counts the number of accepting computations of a nondeterministic logspace bounded Turing machine. An extension of #L is GapL which is the *difference* of the number of accepting and rejecting computations of a nondeterministic logspace bounded Turing machine. It is known that

$$\mathrm{NC}^1 \subseteq \mathrm{L} \subseteq \mathrm{NL} \subseteq \mathrm{AC}^1$$

and

$$\#\mathrm{L} \subseteq \mathrm{GapL} \subseteq \mathrm{TC}^1 \subseteq \mathrm{NC}^2\,.$$

The interest for GapL stems from the fact that it characterizes the complexity of computing the determinant and the Pfaffian of integer matrices. Also, the number of perfect matchings in planar graphs can be computed in GapL [19]. Our parallel algorithm to count the number of perfect matchings in K_5-free graphs has up to $\log n$ levels where perfect matchings are counted in planar components. This will result in a TC^2-circuit.

3 Decomposition of Graphs

Wagner [27] studied the decomposition of K_5-free graphs into 2-, 3- and 4-connected components. He showed that the components will be planar at some point, except for one type of component which has constant size. The number of perfect matchings in a planar graph can be computed in polynomial time [14], in fact in NC [25]. Our goal is to use Wagners result to reduce the problem of computing the number of perfect matchings in a K_5-free graph to the one for planar graphs.

Let $G = (V, E)$ be a graph. If G is not connected, then the number of perfect matchings in G is the product of the number of perfect matchings in the connected components of G. Hence we may assume in the following that G is connected.

But actually we may assume that G is biconnected. Otherwise there is an articulation point a in G. Let $G_1, \dots, G_l$ be the connected components of $G - a$. For a perfect matching to exist, G must have an even number of vertices. Therefore, exactly one of $G_1, \dots, G_l$ must have an odd number of vertices, say G_1, and the others must have even size. Then the number of perfect matchings in G is the product of the number of perfect matchings in $G_1 \cup a, G_2, \dots, G_l$. We can continue to split these components along the remaining articulation points until we end up with biconnected components only. Hence it suffices to determine the number of perfect matchings in biconnected graphs. We assume in the following that G is biconnected.

3.1 The Triconnected Components

There is an extensive literature on graph decomposition, see for example [2, 3, 12, 13, 18, 21]. We follow the exposition of [7, 22] which works for parallel computation, in fact in logspace.

Definition 3.1 [7] Let $G = (V, E)$ be a biconnected graph. A separating pair $\{a, b\}$ is called *3-connected* if there are three vertex-disjoint paths between a and b in G.

The *triconnected components of* G are the split graphs we obtain from G by splitting G successively along all 3-connected separating pairs, in any order. If a separating pair $\{a, b\}$ is connected by an edge in G, then we also define a 3-*bond* for $\{a, b\}$ as a triconnected component, i.e., a multigraph with two vertices $\{a, b\}$ and three edges between them.

By the definition, 3-connected components, cycle components, and 3-bonds are the triconnected components of a biconnected graph. The 3-bonds represent edges of the graph between separating pairs because they are replaced by virtual edges in the components. The cycle components are not 3-connected, but they are not decomposed further anyway.

Definition 3.2 Let $G = (V, E)$ be a biconnected graph. The *triconnected component tree* $\mathcal{T}$ *of* G is the following graph. There is a node for each triconnected component and for each 3-connected separating pair of G. There is an edge in $\mathcal{T}$ between the node for triconnected component C and the node for a separating pair $\{a, b\}$, if a and b belong to C.

Lemma 3.3 [7, 22] *The triconnected component tree can be computed and traversed in logspace.*

We fix one component node in $\mathcal{T}$ as the root of $\mathcal{T}$. Hence we can talk of a parent and the children of a node. For a node N in $\mathcal{T}$, we define

$$\mathcal{T}(N) = \text{ the subtree of } \mathcal{T} \text{ with root } N.$$

Next we define the subgraph $G(\mathcal{T}(N))$ of G associated with $\mathcal{T}(N)$.

Definition 3.4 By $G(\mathcal{T}(N))$ we denote the subgraph of G induced by the vertices of G that occur in the nodes of $\mathcal{T}(N)$, with one exception: vertices a, b that are a separating pair can occur in several nodes of $\mathcal{T}(N)$. When there is an edges (a, b) in G and a, b occur in N, then we define $G(\mathcal{T}(N))$ to contain the edge (a, b) *only for* the component N which is closest to the root of $\mathcal{T}$. In the other components N' where a, b occurs, we put no edge (a, b) in $G(\mathcal{T}(N'))$.

3.2 The 4-Connected Components

There are some subtleties in the decomposition of a 3-connected graph into 4-connected components to work in logspace. We refer to the exposition in [22]. In a nutshell, it is an inductive process that splits off components along separating triples. However, the separating triples might overlap each other, and even worse, their split components might overlap. In this case separating triples are called *crossing* in [22]. It is shown that one can select one of the crossing triples and throw away the other to obtain a decomposition into 4-connected components.

The components we get are

- separating triples where the vertices are connected by virtual edges,
- 4-connected components that contain the separating triples where they are split off. Again there are virtual edges between the vertices of the separating triples,
- 3-bonds for every pair a, b of vertices that are part of a separating triple and there is an edge $(a, b) \in E$.

Let C be a 3-connected component of G. The *4-connected component tree* $\mathcal{T}_C$ *of* C has a node for every component as described above that occurs in the decomposition process of C. There is an edge between a 4-connected component node D and a separating triple node τ in $\mathcal{T}_C$ if τ belongs to D. If there is a 3-bond for two vertices a, b which are in τ, then we also have an edge between the 3-bond and τ in $\mathcal{T}_C$.

Lemma 3.5 [22] *The 4-connected component tree can be computed and traversed in logspace.*

Fix one component node as the root of $\mathcal{T}_C$. Let D be a component node and τ be a separating triple. Similar as for the triconnected component tree, we define $\mathcal{T}_C(D)$ and $\mathcal{T}_C(\tau)$ as the subtrees of $\mathcal{T}_C$ rooted at D and τ, respectively. Similar to Definition 3.4, we define $G(\mathcal{T}_C(D))$ and $G(\mathcal{T}_C(\tau))$ as the subgraphs of G associated with the subtrees $\mathcal{T}_C(D)$ and $\mathcal{T}_C(\tau)$, respectively.

3.3 Properties of K_5-Free Graphs

The crucial theorem about K_5-free graphs is due to Wagner [27].

Theorem 3.6 [27] *A 3-connected non-planar component of a K_5-free biconnected graph is either the Möbius ladder M_8 or its 4-connected components are all planar.*

The Möbius ladder M_8 is shown in Fig. 1. It is a 3-connected graph on 8 vertices which is non-planar because it contains a $K_{3,3}$.

The Möbius ladder M_8 has 5 perfect matchings. However, we will also have weights on the edges. In this case, we have to count the weighted perfect matchings of M_8. Since M_8 has constant size this is computationally a simple task.

Theorem 3.6 describes the route we follow: we decompose the given biconnected graph G into 3-connected components. When we are lucky, a 3-connected component C is already planar or M_8. In this case we directly compute the number of perfect matchings in C. Otherwise, we decompose C further into 4-connected components. These components are all planar now and we can again compute the number of perfect matchings there. What makes things a bit tricky is, that we have to consider all possibilities of assigning the separating pairs and triples to the components they are part of. This will be the major challenge for the complexity bound on computing the number of perfect matchings.

4 Counting Perfect Matchings in K_5-Free Graphs

In this section we prove the following theorem.

Theorem 4.1 *The number of perfect matchings in a K_5-free graph can be computed in polynomial time.*

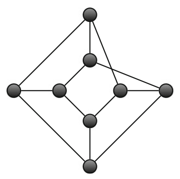
Fig. 1 The Möbius ladder M_8

Let $G = (V, E)$ be a biconnected K_5-free graph. We will decompose G into triconnected components, and, if necessary, into 4-connected components. Thereby we end up with components that are either planar, or the Möbius ladder M_8. The number of perfect matchings of these components can be computed in polynomial time. Note that the Möbius ladder M_8 has constant size. The critical part of our algorithm is to put these numbers together to obtain the number of perfect matchings of G.

Consider the triconnected component tree $\mathcal{T}$ of G. We will compute the number of perfect matchings of the components in a bottom-up fashion according to $\mathcal{T}$ by dynamic programming. If a component C is non-planar and not equal to M_8, then we decompose C and consider its 4-connected component tree $\mathcal{T}_C$. Then we compute the number of perfect matchings of C by dynamic programming according to $\mathcal{T}_C$. Note that the separating pairs and triples occur in several components. However, when we consider perfect matchings in G, we should match the vertices of these pairs or triples only in one of the components, respectively. Hence we have to consider all possibilities to put the vertices of the separating pairs and triples into the split components.

Our algorithm will successively replace components by gadgets. The gadgets will have weighted edges. Hence we will compute the *weighted* number of perfect matchings. In the given graph G, edges have no weights. Equivalently we can say that all edges have weight one. In the decomposition of G into tri- and 4-connected components we introduce virtual edges between the vertices of the separating pairs and triples. The virtual edges that do not have an associated 3-bond are defined to have weight zero. These are the edges which are not present in G. With weight zero they do not contribute to the number of perfect matchings.

We start by considering the algorithm for the triconnected component tree $\mathcal{T}$ of G. Then we look at 4-connected component trees.

4.1 The Triconnected Component Tree

Let $\mathcal{T}$ be the triconnected component tree of G. One component node of $\mathcal{T}$ is labeled as the root of $\mathcal{T}$. We describe an algorithm that computes the number of perfect matchings by dynamic programming. We start with the leaf nodes of $\mathcal{T}$. These are component nodes. Then we inductively work our way up to the root of $\mathcal{T}$.

Let C be a leaf in $\mathcal{T}$ and $\pi = \{a, b\}$ be the parent separating pair of C in $\mathcal{T}$. We compute the number of perfect matchings in C for every possibility of keeping a or b in C or not. If edge (a, b) is present in G, we should put it only into one of the split components of $\{a, b\}$ in order to get the correct number of perfect matchings. We will put edge (a, b) into its parent component, by giving it weight one there. Therefore, we define the weight of edge (a, b) to be zero in C.

- If C has odd size, we compute $p_a(C) = \#\mathrm{pm}(C - a)$ and $p_b(C) = \#\mathrm{pm}(C - b)$.
- If C has even size, we compute $p_\emptyset(C) = \#\mathrm{pm}(C)$ and $p_{ab}(C) = \#\mathrm{pm}(C - \pi)$.

This works directly only when C is a planar component or $C = M_8$. Otherwise C is non-planar and we decompose C into 4-connected components. We show in the next subsection how to compute the number of perfect matchings in this case.

In the inductive step, let $\pi = \{a, b\}$ be a separating pair node in $\mathcal{T}$. Let C_0 be the parent of $\{a, b\}$ in $\mathcal{T}$, and $C_1, C_2, \ldots, C_\ell$ be the children of π. Vertices a, b are contained in all these components. We should match a and b only in one of the components, respectively.

Define n_i to be the number of vertices of the subgraph $G(\mathcal{T}(C_i))$ of G. At most two of $n_1, \ldots, n_\ell$ can be odd, otherwise there is no perfect matching in G.

There are three cases:

- $n_1, \ldots, n_\ell$ are all even. Then a and b have both to be matched within one component C_i, for some $i \in \{1, \ldots, \ell\}$, or within C_0. Hence there are $\ell + 1$ possibilities to assign a, b.
- One of $n_1, \ldots, n_\ell$ is odd, say n_i. Then either a has to be matched within C_i and b within C_0, or vice versa. Hence there are two ways to assign a, b.
- In case where two of $n_1, \ldots, n_\ell$ are odd, we assign one of a, b to each of the two odd components. There are again two ways to assign a, b.

Any assignment of a and b other than the ones described above will result in zero perfect matchings.

We keep track of the assignments of a and b by a vector $\boldsymbol{\beta} = (\beta_1, \beta_2, \ldots, \beta_\ell)$ and β_0, where $\beta_i \subseteq \pi$ are the vertices that should *not* be matched in C_i. Such a vector $\boldsymbol{\beta}$ is called *legal w.r.t.* β_0, if it corresponds to an assignment of a and b as explained above. That is, let $\overline{\beta}_i = \pi - \beta_i$. Then $\boldsymbol{\beta}$ is legal w.r.t. β_0 if the $\overline{\beta}_i$'s are pairwise disjoint and $\bigcup_{i \geq 0} \overline{\beta}_i = \pi$. Moreover, the assignment defined by the vector should respect the odd-even cases explained above. There are at most ℓ legal vectors $\boldsymbol{\beta}$ for a fixed β_0.

Recall that $\mathcal{T}(C_i)$ is the subtree of $\mathcal{T}$ rooted at node C_i and $G(\mathcal{T}(C_i))$ is the graph associated with $\mathcal{T}(C_i)$. Inductively assume that we have already computed

$$p_\beta(C_i) = \#\mathrm{pm}(G(\mathcal{T}(C_i)) - \beta),$$

for every $\beta \subseteq \pi$ and $i = 1, 2, \ldots, \ell$. For a legal vector $\boldsymbol{\beta} = (\beta_1, \beta_2, \ldots, \beta_\ell)$ w.r.t. β_0, define

$$p_{\boldsymbol{\beta}}(\pi) = \prod_{i=1}^{\ell} p_{\beta_i}(C_i)$$

$$p_{\beta_0}(\pi) = \sum_{\boldsymbol{\beta} \text{ legal w.r.t. } \beta_0} p_{\boldsymbol{\beta}}(\pi)$$

Then we have

$$p_{\beta_0}(\pi) = \#\mathrm{pm}(G(\mathcal{T}(\pi)) - \beta_0).$$

There are only two possibilities for β_0. We compute $p_{\beta_0}(\pi)$ for both of these values.

The other case in the inductive step is to consider a component node C in $\mathcal{T}$. Let $\pi_0 = \{a_0, b_0\}$ be the parent separating pair of C in $\mathcal{T}$, and $\pi_1 = \{a_1, b_1\}, \ldots, \pi_\ell = \{a_\ell, b_\ell\}$ be the children of C. As already explained in the leaf-case above, edge (a_i, b_i) gets weight one if it is an edge in G, for $i = 1, 2, \ldots, \ell$. Edge (a_0, b_0) gets weight zero.

Inductively assume that we have already computed

$$p_\beta(\pi_i) = \#\mathrm{pm}(G(\mathcal{T}(\pi_i)) - \beta),$$

for every $\beta \subseteq \pi_i$ and $i = 1, 2, \ldots, \ell$. Our goal is to compute

$$p_\beta(C) = \#\mathrm{pm}(G(\mathcal{T}(C)) - \beta),$$

for every $\beta \subseteq \pi_0$. To do so, we replace the subgraphs $G(\mathcal{T}(\pi_i))$ of $G(\mathcal{T}(C))$ by appropriate weighted gadgets, for $i = 1, 2, \ldots, \ell$. That is, we take component C and add the gadgets at the separating pairs π_i.

- If $G(\mathcal{T}(\pi_i))$ has an odd number of vertices, we add one new vertex v_i to C and
  - an egde (v_i, a_i) of weight $p_{b_i}(\pi_i)$ and
  - an egde (v_i, b_i) of weight $p_{a_i}(\pi_i)$.
- If $G(\mathcal{T}(\pi_i))$ has an even number of vertices, we add two new vertices u_i, v_i to C and
  - an egde (u_i, a_i) of weight $p_\emptyset(\pi_i)$,
  - an egde (v_i, b_i) of weight 1, and
  - an egde (u_i, v_i) of weight $p_{a_i b_i}(\pi_i)$.

Figure 2 shows the gadgets. These gadgets were also used in [24]. Figure 3 shows an example of the construction.

Let C' be the resulting component. The construction is such that the number of weighted perfect matchings of C' is the same as the number of perfect matchings of $G(\mathcal{T}(C))$. If C is planar, then also C' is planar and we can directly compute $\#\mathrm{pm}(G(\mathcal{T}(C)) - \beta)$, for every $\beta \subseteq \pi_0$. The same holds if $C = M_8$ because then also C' has constant size.

The third case is that C is non-planar and not equal to M_8. This case we handle slightly different. Namely we do *not* place the gadgets right now in C. Instead, we first decompose C into 4-connected components. The reason is that C' is not 3-connected because of the gadgets. For every separating pair π_i we choose one 4-connected component where π_i occurs and put the gadget there.

The algorithm runs until C is the root of $\mathcal{T}$. In this case C has no parent separating pair. Then $\#\mathrm{pm}(G(\mathcal{T}(C)))$ is our result, the number of perfect matchings in G.

4.2 The 4-Connected Component Tree

Let $C \neq M_8$ be a non-planar 3-connected component with weighted edges. Let π_0 be the parent separating pair of C in the triconnected component tree of G, and $\beta \subseteq \pi_0$.

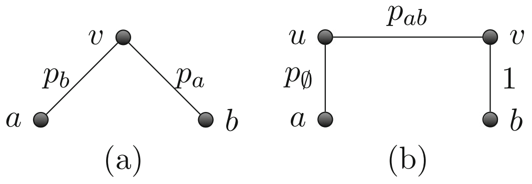

Fig. 2 The gadget for separating pair $\pi = \{a, b\}$ in case $G(\mathcal{T}(\pi))$ has **a** an odd number and **b** an even number of vertices. The gadgets replace $G(\mathcal{T}(\pi))$ in G

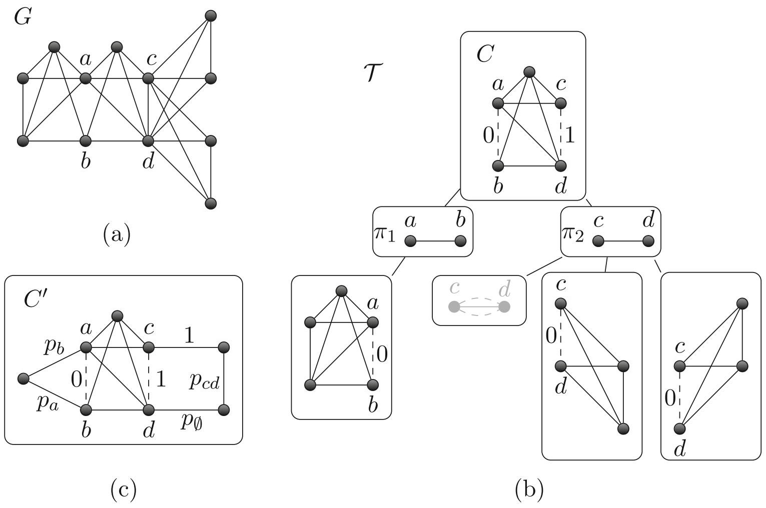

Fig. 3 **a** A biconnected graph G. **b** The triconnected component tree $\mathcal{T}$ of G. Component C is the root of $\mathcal{T}$. In the components, the virtual edges are indicated with dashed lines together with their weights. **c** The component C' constructed from C. The subgraphs that correspond to the subtrees below C in $\mathcal{T}$ are replaced by weighted gadgets in C'

Recall that β contains the vertices that should not be matched within C, i.e., we want to compute the weighted number of perfect matchings in $C - \beta$. In the exposition below, we omit β for better readability. Just keep in mind that β has to be subtracted from every component we consider below.

Let $\mathcal{T}_C$ be the 4-connected component tree of C. Recall that we postponed the placement of the gadgets from the child separating pairs $\pi_1, \pi_2, \ldots, \pi_\ell$ of C in the triconnected component tree $\mathcal{T}$. Our first step now is to choose one component node in $\mathcal{T}_C$ for each π_i where π_i occurs and place the gadget there. For simplicity, we still call the component C in the following.

Let N be a node in $\mathcal{T}_C$. We want to define $G(\mathcal{T}_C(N))$, the graph associated with the subtree $\mathcal{T}_C(N)$ of $\mathcal{T}_C$ similar to the definition of $G(\mathcal{T}(C))$ for the triconnected component tree $\mathcal{T}$. However, there is one difference: the gadgets inherited from $\mathcal{T}$ that are placed in some component node of $\mathcal{T}_C(N)$ have to be added in $G(\mathcal{T}_C(N))$ too. For better readability we do not introduce a new name for $G(\mathcal{T}_C(N))$.

The algorithm to compute the number of perfect matchings in C is similar to the one for the triconnected component tree. One 4-connected component node is labeled as the root of $\mathcal{T}_C$. We start at the leafs of $\mathcal{T}_C$ and inductively proceed to the root of $\mathcal{T}_C$.

Let D be a leaf in $\mathcal{T}_C$ and $\tau = \{a, b, c\}$ be the parent separating triple of D in $\mathcal{T}_C$. The edges between vertices a, b, c which are present in G should be put only into one of the split components of τ in order to get the correct number of perfect matchings. We will put these edges into the parent component of τ, by giving them weight one there. Therefore, we define the weight of the three virtual edges within τ to be zero in D.

We compute the number of perfect matchings in D for every possibility of keeping a, b or c in D or not. That is, we compute $p_\gamma(D) = \#\mathrm{pm}(D - \gamma)$, for all $\gamma \subseteq \tau$. If D has odd size, it suffices to take γ odd, and if D has even size, we can restrict γ to be even. Recall that the 4-connected components are all planar. Hence we can directly compute the number of perfect matchings.

In the inductive step, let $\tau = \{a, b, c\}$ be a separating triple node in $\mathcal{T}_C$. Let D_0 be the parent of τ in $\mathcal{T}_C$, and $D_1, D_2, \ldots, D_\ell$ be the children of τ. We consider the possible case$f$s where to match a, b and c.

Recall that β are the vertices of the parent separating pair π_0 of C that should *not* be matched within C. We start by considering the cases when β and τ overlap.

- If $|\tau \cap \beta| = 2$, say $\tau - \beta = \{a\}$, then we have the same situation as if a were an articulation point: for a perfect matching to exist there is exactly one way to assign a to a component. Hence either $p_\emptyset(\tau) = \#\mathrm{pm}(G(\mathcal{T}_C(\tau)) - \beta)$ or $p_a(\tau) = \#\mathrm{pm}(G(\mathcal{T}_C(\tau)) - a - \beta)$ is possibly non-zero and this number can be computed efficiently.
- If $|\tau \cap \beta| = 1$, say $\tau - \beta = \{a, b\}$, then we have the same situation as if $\{a, b\}$ were a separating pair: we assign these vertices to $D_0, \ldots, D_\ell$ as explained in Section 4.1.

Hence, the interesting case is when $\tau \cap \beta = \emptyset$. We show how to handle this case in the rest of the section. Again, we omit to subtract β from every component for better readability.

Define n_i to be the number of vertices of the subgraph $G(\mathcal{T}_C(D_i) - \tau)$ of G. At most three of $n_1, \ldots, n_\ell$ can be odd, otherwise there is no perfect matching in G.

There are four cases:

- $n_1, \ldots, n_\ell$ are all even. Then either two vertices out of τ are matched within one of $D_1, \ldots, D_\ell$ and the remaining vertex within D_0, or all vertices of τ are matched within D_0. These are $3\ell + 1$ possibilities to assign a, b, c.
- One of $n_1, \ldots, n_\ell$ is odd, say n_i. Then either a, b, c are all matched within D_i, or just one of a, b, c is matched within D_i and the other two in D_j, for some $j \neq i$, or in D_0. Hence there are again $3\ell + 1$ possibilities to assign a, b, c.
- Two of $n_1, \ldots, n_\ell$ are odd, say n_i and n_j. Then one vertex of a, b, c is matched within D_i, one within D_j, and the remaining one in D_0. There are 6 ways to assign a, b, c.
- In case where three of $n_1, \ldots, n_\ell$ are odd, we assign one of a, b, c to each of the corresponding components. There are again 6 ways to assign a, b, c.

Any assignment of a, b and c other than the ones described above will result in zero perfect matchings.

We administrate the assignments of a, b and c again by a vector $\boldsymbol{\gamma} = (\gamma_1, \gamma_2, \ldots, \gamma_\ell)$ and γ_0, where $\gamma_i \subseteq \tau$ are the vertices *not* matched in D_i. Similar as for the $\boldsymbol{\beta}$-vector in the triconnected component tree, we define $\boldsymbol{\gamma}$ to be *legal w.r.t.* γ_0 if it represents an assignment of a, b, c to the D_i's as explained above.

Inductively we have already computed

$$p_\gamma(D_i) = \#\mathrm{pm}(G(\mathcal{T}_C(D_i)) - \gamma),$$

for every $\gamma \subseteq \tau$ and $i = 1, 2, \ldots, \ell$. For a legal vector $\boldsymbol{\gamma} = (\gamma_1, \gamma_2, \ldots, \gamma_\ell)$ w.r.t. γ_0, define

$$p_{\boldsymbol{\gamma}}(\tau) = \prod_{i=1}^{\ell} p_{\gamma_i}(D_i)$$

$$p_{\gamma_0}(\tau) = \sum_{\boldsymbol{\gamma} \text{ legal w.r.t. } \gamma_0} p_{\boldsymbol{\gamma}}(\tau)$$

Then we have

$$p_{\gamma_0}(\tau) = \#\mathrm{pm}(G(\mathcal{T}_C(\tau)) - \gamma_0).$$

There are at most 4 possibilities for γ_0. We compute $p_{\gamma_0}(\tau)$ for all of these values.

The second case in the inductive step is to consider a component node D in $\mathcal{T}_C$. Let $\tau_0 = \{a_0, b_0, c_0\}$ be the parent separating triple of D in $\mathcal{T}_C$, and $\tau_1 = \{a_1, b_1, c_1\}, \ldots, \tau_\ell = \{a_\ell, b_\ell, c_\ell\}$ be the children of D. As already explained in the leaf-case above, the edges within τ_0 get weight zero, and the edges between the vertices in τ_i which are present in G get weight one in D, for $i = 1, 2, \ldots, \ell$. If $\tau_i \cap \tau_0 \neq \emptyset$ for some $i \geq 1$, there might be an edge e within both, τ_0 and τ_i. Then e gets weight zero in D.

Recall again that β are the vertices of the parent separating pair π_0 of C that should *not* be matched within C. First we consider the cases when β overlaps some τ_i.

- When $|\tau_i \cap \beta| = 2$ for some $0 \leq i \leq \ell$, say $\tau_i - \beta = \{a_i\}$, we have two cases:
  - If $G(\mathcal{T}_C(\tau_i)) - \beta$ has even size we add a new vertex x_{a_i} to D and an edge (a_i, x_{a_i}) with weight $p_\emptyset(\tau_i) = \#\mathrm{pm}(G(\mathcal{T}_C(\tau_i)) - \beta)$.
  - If $G(\mathcal{T}_C(\tau_i)) - \beta$ has odd size we add new vertices x_{a_i}, x'_{a_i} to D and edges (a_i, x_{a_i}) with weight one and (x_{a_i}, x'_{a_i}) with weight $p_{a_i}(\tau_i) = \#\mathrm{pm}(G(\mathcal{T}_C(\tau_i)) - a_i - \beta)$.
- When $|\tau_i \cap \beta| = 1$ for some $0 \leq i \leq \ell$, say $\tau_i - \beta = \{a_i, b_i\}$, we add the gadgets of Fig. 2 to a_i and b_i according to the rules discussed in Section 4.1.

Hence, it remains the case when $\tau_i \cap \beta = \emptyset$ for all i. Again, in the following we omit to subtract β from every component for better readability.

Inductively assume that we have already computed

$$p_\gamma(\tau_i) = \#\mathrm{pm}(G(\mathcal{T}_C(\tau_i)) - \gamma),$$

for every $\gamma \subseteq \tau_i$ and $i = 1, 2, \ldots, \ell$. Our goal is to compute

$$p_\gamma(D) = \#\mathrm{pm}(G(\mathcal{T}_C(D)) - \gamma),$$

for every $\gamma \subseteq \tau_0$. We replace the subgraphs $G(\mathcal{T}_C(\tau_i))$ of $G(\mathcal{T}_C(D))$ again by appropriate weighted gadgets, for $i = 1, 2, \ldots, \ell$. That is, we add the gadgets at the separating triples τ_i of D. The gadgets incorporate all possibilities to match some vertices within $G(\mathcal{T}_C(\tau))$ and some vertices in the rest of G. We distinguish the cases whether $G(\mathcal{T}_C(\tau_i))$ has an odd or even number of vertices. The gadgets are shown in Fig. 4. They are similar to those given in [24].

Let D' be the resulting component. The construction guarantees that the number of weighted perfect matchings of D' is equal to the weighted number of perfect

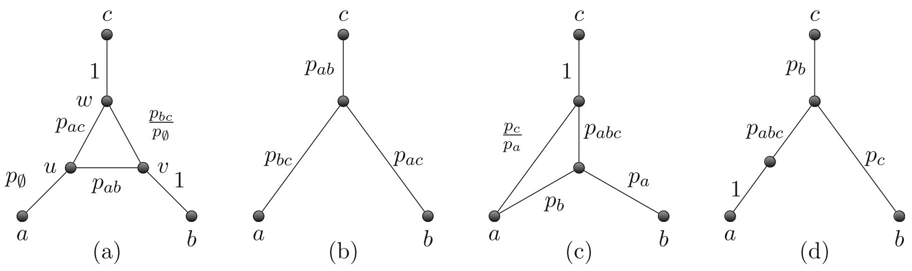

Fig. 4 The gadget for separating triple $\tau = \{a, b, c\}$ in case $G(\mathcal{T}_C(\tau))$ has **a** an even number and **c** an odd number of vertices. The gadgets replace $G(\mathcal{T}_C(\tau))$ in G. In (**a**), an odd number of a, b, c should be matched within the gadget, in (**c**) an even number. In (**a**), when $p_\emptyset = 0$, we use the gadget shown in (**b**) instead. Similarly, when $p_a = 0$ in (**c**) we use the gadget from (**d**). For example in (**a**), if a, b, c should all be matched within the gadget, we use the edges (a, u), (b, v), (c, w). These edges contribute weight $p_\emptyset$ to a perfect matching. If only b should be matched within the gadget, we use edges (b, v) and (u, w). These two edges contribute weight p_{ac} to a perfect matching. Then a and c have to be matched in the rest of the graph. Similarly, when a should be matched within the gadget, we use edges (a, u) and (v, w). These two edges contribute weight $p_\emptyset \cdot \frac{p_{bc}}{p_\emptyset} = p_{bc}$ to a perfect matching

matchings of $G(\mathcal{T}_C(D))$: let $\tau = \{a, b, c\}$ be a child of D and $\gamma \subseteq \tau$. For every perfect matching of G that matches $\overline{\gamma} = \tau - \gamma$ with edges of $G(\mathcal{T}_C(\tau))$, there is precisely one possibility to match $\overline{\gamma}$ within the gadget. By construction, the weight contribution to every perfect matching of D' which matches γ outside of the gadget and $\overline{\gamma}$ within the gadget is $p_\gamma(\tau)$. Therefore we get the same number of perfect matchings.

Observe also that D' is planar. This is because D is planar and the gadgets are planar. Moreover, in any planar embedding of D, a separating triangle τ is one of the faces. Hence we can put the gadget inside this triangle and maintain planarity. It follows that we can compute #pm(D') in polynomial time. Figure 5 shows an example of the construction. This completes the proof of Theorem 4.1.

5 Counting Perfect Matchings in K_5-Free Graphs in Parallel

We parallelize the algorithm from Section 4. We explain a circuit construction and argue that it has polylogarithmic depth. Our plan is to transform the sequential algorithm of Section 4 into a circuit. The decomposition of the given graph into 3- or 4-connected components is in logspace, and hence can be parallelized. However, since the resulting component trees may have depth $O(n)$, we cannot simply evaluate the tree bottom-up as in the sequential case.

To get around this problem we identify paths in the component trees that lead to a large depth.

Definition 5.1 Let T be a tree and u, v be two nodes such that u is a child of v in T. Then u is called a *large child* of v if $|T(u)| > |T(v)|/2$. A *large-child path* in T is a path $v_0, v_1, \ldots, v_k$ of maximal length in T such that v_i is a large child of v_{i-1}, for $i = 1, \ldots, k$.

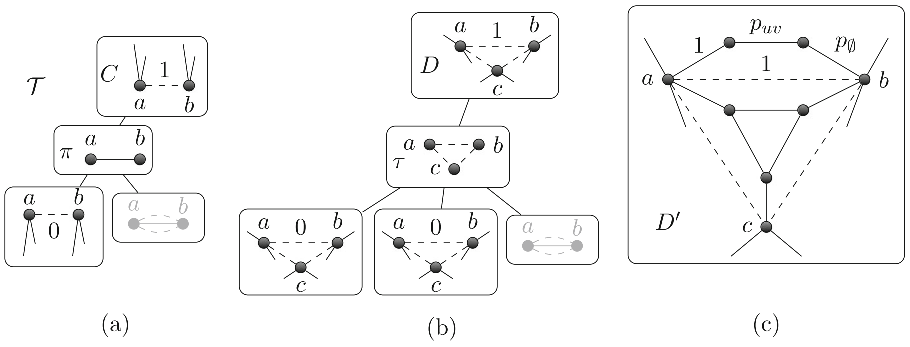

Fig. 5 **a** A triconnected component tree $\mathcal{T}$ with C as a triconnected component node and π as a separating pair node. We want to compute the weighted number of perfect matchings in C. **b** Component $C \neq M_8$ is not planar and hence, is decomposed into a 4-connected component tree $\mathcal{T}_C$. The more interesting case is, when $\pi \subseteq \tau$, then there are several occurrences of π as virtual edges $\{a, b\}$ in 4-connected component nodes. Only one of these virtual edges gets weight 1. **c** In D, the parent component of τ, the gadget corresponding to $\mathcal{T}(\pi)$ is embedded additionally. The gadget (without weights) corresponding to $\mathcal{T}_C(\tau)$ is plugged inside the triangle of vertices a, b, c. Both gadgets preserve planarity

We show how to handle large-child paths in parallel. This will suffice to obtain a small depth circuit.

Theorem 5.2 *Counting perfect matchings in K_5-free graphs is in* TC^2.

Recall that the number of perfect matchings in planar graphs can be computed in TC^1. Hence, for every component node C which is planar or M_8 and has parent π_0, we have a TC^1-circuit which computes $\#\mathrm{pm}(G(\mathcal{T}(C)) - \beta)$ for every $\beta \subseteq \pi_0$. We combine these circuits according to the edge relation in the component trees. I.e., we connect the output of a subcircuit for a node to the input of the subcircuit corresponding to its parent node. To obtain a circuit with polylogarithmic depth we deviate from the circuit construction at *large children* in the tree.

We will compute the number of perfect matchings in the components on a large-child path in parallel. We do this for all possible input values of a subcircuit. Recall that a subcircuit gets edge weights as input which can be large. Therefore we use the Chinese remainder representation to represent large numbers, i.e., all computations are done modulo p for enough small prime numbers p.

5.1 The Computation Graph

Recall that we have tri- *and* 4-connected component trees in the decomposition process of a K_5-free connected graph. We define a new tree, the *computation graph* $\mathcal{K}$ of a K_5-free biconnected graph G which combines the tri- and 4-connected component trees into one tree. Informally we start with the triconnected component tree $\mathcal{T}$ of G and replace every non-planar component node $C \neq M_8$ of the triconnected component tree $\mathcal{T}$ by the 4-connected component tree $\mathcal{T}_C$. More precisely, $\mathcal{K}$ is defined as follows.

Definition 5.3 The *computation graph* $\mathcal{K}$ of a K_5-free biconnected graph G has the same nodes as the triconnected component tree $\mathcal{T}$ of G, but instead of the node for a non-planar component $C \neq M_8$, it has all the nodes of the 4-connected component tree $\mathcal{T}_C$.

The edges between nodes within $\mathcal{T}$ or within a $\mathcal{T}_C$ are the same as in these trees, respectively. For a non-planar component $C \neq M_8$ in $\mathcal{T}$ let π_0 be the parent of C in $\mathcal{T}$ and $\pi_1, \ldots, \pi_\ell$ be its children. Let D be the root of $\mathcal{T}_C$. Then we define an edge between π_0 and D. Furthermore for every child $\pi_i, i \in \{1, \ldots, \ell\}$ there is a unique node in $\mathcal{T}_C$ connected to it. This unique node is the node where the gadget of $G(\mathcal{T}(\pi_i))$ of the algorithm of Section 4 is plugged in.

Note that when we plug in a tree $\mathcal{T}_C$ in $\mathcal{T}$ for some non-planar nodes $C \neq M_8$ in $\mathcal{T}$, then the children of C and its parent are connected to exactly one node in $\mathcal{T}_C$. Therefore $\mathcal{K}$ is again a tree. We can assume that $\mathcal{K}$ is a rooted tree. Recall that the tri- and 4-connected component trees of G can be computed in logspace [22], and hence, this also holds for $\mathcal{K}$. The following lemma summarizes the observations.

Lemma 5.4 *The computation graph $\mathcal{K}$ of a biconnected graph G is a tree. It can be computed in logspace.*

Let C be a component in $\mathcal{T}$ with parent separating pair π. We want to compute $\#\mathrm{pm}(G(\mathcal{T}(C)) - \beta)$, for any $\beta \subseteq \pi$. If $C \neq M_8$ is non-planar and D is a component in $\mathcal{T}_C$ with parent separating triple τ, then we also want to compute $\#\mathrm{pm}(G(\mathcal{T}_C(D)) - \gamma - \beta)$, for any $\gamma \subseteq \tau$. Hence component D does not only depend on its parent separating set node τ, but additionally on a separating pair π. We define a set $\mathcal{V}_\mathcal{K}$ which covers all possibilities of β and γ.

Let R_0 be the root in $\mathcal{K}$ and let S be any separating pair or triple node in $\mathcal{K}$. Let P be a simple path from R_0 to S. Let $\widehat{S}$ be the separating pair node with shortest distance to S on P. I.e. $\widehat{S} = S$ in case S is a separating pair node. We define $\mathcal{V}_\mathcal{K}(S) = \mathcal{P}(S) \cup \mathcal{P}(\widehat{S})$, where $\mathcal{P}$ denotes the power set. Let $\mu = |\mathcal{V}_\mathcal{K}(S)| \leq 2^2 + 2^3$.

For a node N in $\mathcal{K}$, the subtree of $\mathcal{K}$ with root N is denoted by $\mathcal{K}(N)$. The subgraph $G(\mathcal{K}(N))$ of G associated with $\mathcal{K}(N)$ is defined similar as in Definition 3.4 for the triconnected component tree.

Because all component nodes in $\mathcal{K}$ are planar components or M_8, the number of perfect matchings in these components can be computed in TC^1.

Lemma 5.5 *Let S be a separating set node and $S_1, \ldots, S_\ell$ be all the descendant separating set nodes at distance two in $\mathcal{K}$. The following function f is in* TC^1*: on input of*

- $\#\mathrm{pm}(G(\mathcal{K}(S_i)) - \kappa_i)$ *for* $i = 1, \ldots, \ell$ *and* $\kappa_i \in \mathcal{V}_\mathcal{K}(S_i)$,
- *the number of vertices in* $G(\mathcal{K}(S_i))$ *for* $i = 1, \ldots, \ell$,
- $\kappa \subseteq \mathcal{V}_\mathcal{K}(S)$,

the output of f *is* $\#\mathrm{pm}(G(\mathcal{K}(S)) - \kappa)$.

Proof Let N be a component node in the level between S and the S_i's in $\mathcal{K}$. In dependence of the number of vertices of S_i and the number of vertices in $G(\mathcal{K}(S_i))$ we determine the correct gadget in Figs. 2 and 4 and add it to the vertices of S_i. The weights of the gadgets come from the input values $\#\mathrm{pm}(G(\mathcal{K}(S_i)) - \kappa_i)$. Then the vertices of κ are removed in N to obtain a component node N'. Clearly, N' is still planar. Hence, the number of weighted perfect matchings in N' can be computed in TC^1 [19].

This has to be done in all the component nodes between S and the S_i's. Then we have a sum and product of many numbers modulo some small prime number. Recall that modulo division as well as addition and multiplication of n numbers with n bits is in TC^0 [26]. □

The lemma handles the case of component nodes and it remains to stick the results together at the nodes of separating pairs and triples in $\mathcal{K}$. Therefore we reduce $\mathcal{K}$ by removing all the component nodes.

Definition 5.6 Let $\mathcal{K}$ be the computation graph of G. The *reduced computation graph $\widehat{\mathcal{K}}$ of $\mathcal{K}$ is defined by the following process: for every component node N in $\mathcal{K}$ with parent S and children $S_1, \dots, S_\ell$, remove N from $\mathcal{K}$ and instead draw edges between S and $S_1, \dots, S_\ell$.*

Similar as $\mathcal{K}$, the reduced computation graph $\widehat{\mathcal{K}}$ is a tree and can be computed in logspace. We argue that $\widehat{\mathcal{K}}$ has at most n nodes: Observe that a vertex of G which is part of a separating set will occur in several nodes of $\mathcal{K}$. However, when we go down in $\mathcal{K}$ from one component node N via a separating set node to the next component node N', then N' will contain at least one vertex v of G which is not in N. Moreover, all nodes in $\mathcal{K}$ that contain v are below N'. It follows that $\mathcal{K}$ has at most n component nodes. Between two component nodes in $\mathcal{K}$, there is a separating set node. Therefore $\mathcal{K}$ has at most n separating set nodes, and these are all the nodes of $\widehat{\mathcal{K}}$.

The tree $\widehat{\mathcal{K}}$ may have depth up to n. To evaluate $\widehat{\mathcal{K}}$ efficiently in parallel, we have to do some depth reduction. The reason for the large depth are the large children: let T be a tree with root r. If v is a non-large child of r, then we have $|T(v)| \leq |T(r)|/2$. Hence, any simple path in T from r to a leaf along non-large children has length at most $\log n$. However, paths that contain large children could be long. Recall that a large-child path is a path of maximal length such that every node on the path, except the first node, is a large child of its parent. Note that there can be several disjoint subpaths on a path from the root to a leaf that are large-child paths. The next lemma states that there are only few large-child paths on any path in T.

Lemma 5.7 *Let p be a path from the root to a leaf node in a tree T. Then we have*

(i) *the number of large-child paths on p is at most* $\log n$*,*
(ii) *the number of nodes on p that are not large children is at most* $\log n$*.*

Proof Consider two consecutive large-child paths p_1, p_2 on p. Say, the first path p_1 goes from s_1 to t_1, and p_2 goes from s_2 to t_2. Because we defined large-child paths to be of maximal length, t_1 has no large child. Hence we have

$$|T(s_2)| \leq |T(t_1)|/2 < |T(s_1)|/2 .$$

Now the claim follows. □

5.2 Circuit Construction

The computation tree $\mathcal{K}$ can be computed in logspace from the input graph G. Since $\mathrm{L} \subseteq \mathrm{TC}^1$, we may think of $\mathcal{K}$, respectively $\widehat{\mathcal{K}}$, being the output of a TC^1-circuit, in some appropriate coding. The output contains information about

- the vertices of G that are in one component node or separating set node in $\mathcal{K}$,
- the number of vertices ins $G(\mathcal{K}(N))$, for every node N,
- the edges that are between the nodes of $\mathcal{K}$ and $\widehat{\mathcal{K}}$,
- the type of a node, i.e. whether it is the root or a leaf, or a large child,
- all the large-child paths in $\widehat{\mathcal{K}}$.

Our goal is to evaluate $\widehat{\mathcal{K}}$, as we did in the sequential algorithm in Section 5. However, $\widehat{\mathcal{K}}$ depends on the input graph G and our circuit has to work for all graphs with the same number n of vertices. We construct the circuit in levels, where there is a subcircuit for *every* node of $\widehat{\mathcal{K}}$ in each level. Every subcircuit in one level is connected to every subcircuit of the next level. These connections represent the potential edge connections in $\widehat{\mathcal{K}}$. The actual edges in a given $\widehat{\mathcal{K}}$ are then activated by the results of the TC^1-circuit that computes $\widehat{\mathcal{K}}$.

Consider a node S in $\widehat{\mathcal{K}}$ and let $S_1, \ldots, S_\ell$ be its children in $\widehat{\mathcal{K}}$. We want to compute $\#\mathrm{pm}(G(\mathcal{K}(S)) - \kappa_S)$, for $\kappa_S \in \mathcal{V}_{\mathcal{K}}(S)$. If S has no large child then there is a TC^1-circuit as described in Lemma 5.5, where the input values are obtained from lower circuit levels.

Because the depth of $\widehat{\mathcal{K}}$ can be as large as n, we cannot afford such a level of subcircuits at any depth of $\widehat{\mathcal{K}}$. That is, we have to deviate from the sequential bottom-up evaluation of $\widehat{\mathcal{K}}$ and do some kind of depth-reduction. What causes the large depth are the large-child paths. We will parallelize the computation along the large-child paths with the balanced binary tree method, see [11]. By Lemma 5.7, the number of large-child paths is bounded by n.

Consider a large-child path $S = \widetilde{S}_0, \widetilde{S}_1, \ldots, \widetilde{S}_t$ in $\widehat{\mathcal{K}}$. For each $\widetilde{S}_i$ we place many TC^1-circuits in parallel as shown in Fig. 6, namely one circuit for each possible value of

- $\#\mathrm{pm}(G(\mathcal{K}(\widetilde{S}_{i+1})) - \kappa_{\widetilde{S}_{i+1}})$ modulo p,
- $\kappa_{\widetilde{S}_{i+1}} \in \mathcal{V}_{\mathcal{K}}(\widetilde{S}_{i+1})$, and
- prime p.

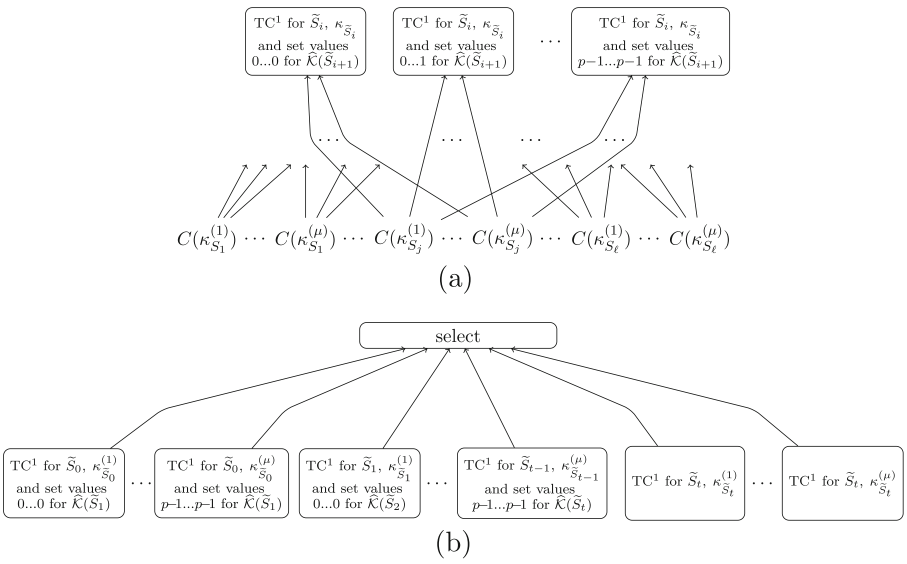
TC^1 for $\widetilde{S}_i$, $\kappa_{\widetilde{S}_i}$ and set values 0...0 for $\widehat{\mathcal{K}}(\widetilde{S}_{i+1})$
TC^1 for $\widetilde{S}_i$, $\kappa_{\widetilde{S}_i}$ and set values 0...1 for $\widehat{\mathcal{K}}(\widetilde{S}_{i+1})$
TC^1 for $\widetilde{S}_i$, $\kappa_{\widetilde{S}_i}$ and set values $p-1...p-1$ for $\widehat{\mathcal{K}}(\widetilde{S}_{i+1})$
$C(\kappa_{S_1}^{(1)}) \cdots C(\kappa_{S_1}^{(\mu)}) \cdots C(\kappa_{S_j}^{(1)}) \cdots C(\kappa_{S_j}^{(\mu)}) \cdots C(\kappa_{S_\ell}^{(1)}) \cdots C(\kappa_{S_\ell}^{(\mu)})$
(a)
select
TC^1 for $\widetilde{S}_0$, $\kappa_{\widetilde{S}_0}^{(1)}$ and set values 0...0 for $\widehat{\mathcal{K}}(\widetilde{S}_1)$
TC^1 for $\widetilde{S}_0$, $\kappa_{\widetilde{S}_0}^{(\mu)}$ and set values p–1...p–1 for $\widehat{\mathcal{K}}(\widetilde{S}_1)$
TC^1 for $\widetilde{S}_1$, $\kappa_{\widetilde{S}_1}^{(1)}$ and set values 0...0 for $\widehat{\mathcal{K}}(\widetilde{S}_2)$
TC^1 for $\widetilde{S}_{t-1}$, $\kappa_{\widetilde{S}_{t-1}}^{(\mu)}$ and set values p–1...p–1 for $\widehat{\mathcal{K}}(\widetilde{S}_t)$
TC^1 for $\widetilde{S}_t$, $\kappa_{\widetilde{S}_t}^{(1)}$
TC^1 for $\widetilde{S}_t$, $\kappa_{\widetilde{S}_t}^{(\mu)}$
(b)

Fig. 6 **a** We place many TC^1-circuits for each $\widetilde{S}_i$ in parallel, namely one for each possible weighting scheme for the gadget of $\widehat{\mathcal{K}}(\widetilde{S}_{i+1})$. The circuits for the non-large children $S_1, \dots, S_\ell$ are indicated by $C(\kappa_{S_j}^{(r)})$, for $\kappa_{S_j}^{(r)} \in \mathcal{V}_{\mathcal{K}}(S_j)$ and $j \in \{1, \dots, \ell\}$. They are connected to all the TC^1-circuits for $\widetilde{S}_i$. **b** For all nodes $\widetilde{S}_0, \widetilde{S}_1, \dots, \widetilde{S}_{t-1}$ along a large-child path, there are μp^μ many circuits in parallel, since for a node $\widetilde{S}_i$, there are μ different sets $\kappa_{\widetilde{S}_i} \in \mathcal{V}_{\mathcal{K}}(\widetilde{S}_i)$ and p^μ many different possibilities for the values of $\widehat{\mathcal{K}}(\widetilde{S}_{i+1})$

Assume for the moment, that for each $\widetilde{S}_i$ of the large-child path, the subtrees at the non-large children of $\widetilde{S}_i$ have already been evaluated. We use a flag to indicate when the assumption is fulfilled. We compose the functions computed by the circuits for each $\widetilde{S}_i$ in a binary tree like fashion. In the bottom layer, the composition of the circuits for $\widetilde{S}_{2i-1}$ and $\widetilde{S}_{2i}$ means: for each circuit C for $\widetilde{S}_{2i}$ we put an AC^0-circuit to select the circuit for $\widetilde{S}_{2i-1}$ which uses the output of C as input. Such a circuit exists, since we have a circuit for $\widetilde{S}_{2i-1}$ for every possible output of C. Clearly, we combine only circuits for the same prime p.

We continue to combine the resulting circuits in higher levels similarly. After $\log t$ levels, we have composed the circuits of the whole large-child path. Then the correct values $\#\mathrm{pm}(G(\mathcal{K}(S)) - \kappa_S)$ for all $\kappa_S \in \mathcal{V}_{\mathcal{K}}(S)$ are computed at the output gates of the constructed circuit. The whole composition circuit is in AC^1. A schematic view is shown in Fig. 7.

We bound the depth of the resulting circuit. By Lemma 5.7 there are at most $\log n$ nodes which are non-large children on every path. Therefore $\log n$ levels suffice to evaluate $\widehat{\mathcal{K}}$. Each level consists of TC^1-circuits to compute the number of perfect matchings in some planar component, followed by AC^1-circuits to evaluate large-child paths. Therefore we obtain circuits in TC^2, for every prime p.

Every prime p has at most $O(\log n)$ many bits. For $b \in \mathbb{Z}_p^*$ its inverse element b^{-1} can be computed in TC^0. Hence fractions like a/b that occur

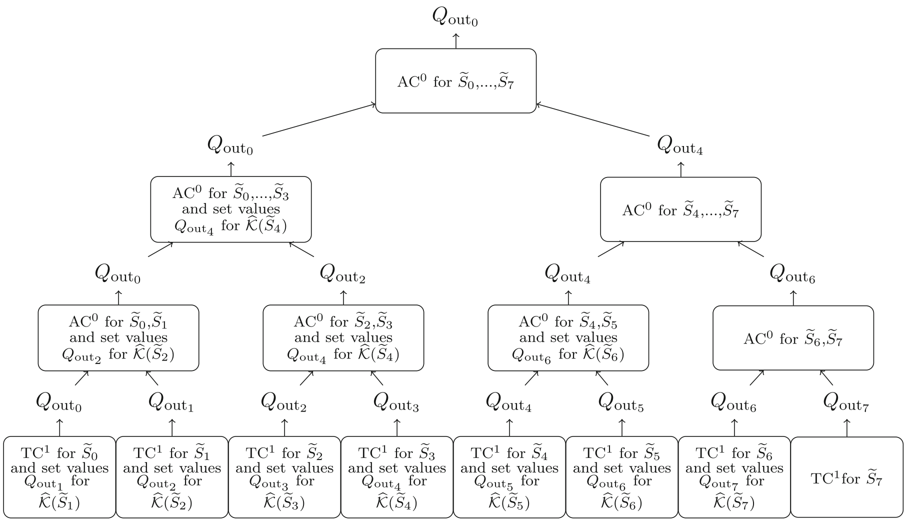
Q_{out_0}
AC^0 for $\widetilde{S}_0,\ldots,\widetilde{S}_7$
Q_{out_0}
AC^0 for $\widetilde{S}_0,\ldots,\widetilde{S}_3$ and set values Q_{out_4} for $\widehat{\mathcal{K}}(\widetilde{S}_4)$
Q_{out_4}
AC^0 for $\widetilde{S}_4,\ldots,\widetilde{S}_7$
Q_{out_0}
Q_{out_2}
Q_{out_4}
Q_{out_6}
AC^0 for $\widetilde{S}_0,\widetilde{S}_1$ and set values Q_{out_2} for $\widehat{\mathcal{K}}(\widetilde{S}_2)$
AC^0 for $\widetilde{S}_2,\widetilde{S}_3$ and set values Q_{out_4} for $\widehat{\mathcal{K}}(\widetilde{S}_4)$
AC^0 for $\widetilde{S}_4,\widetilde{S}_5$ and set values Q_{out_6} for $\widehat{\mathcal{K}}(\widetilde{S}_6)$
AC^0 for $\widetilde{S}_6,\widetilde{S}_7$
Q_{out_0}
Q_{out_1}
Q_{out_2}
Q_{out_3}
Q_{out_4}
Q_{out_5}
Q_{out_6}
Q_{out_7}
TC^1 for $\widetilde{S}_0$ and set values Q_{out_1} for $\widehat{\mathcal{K}}(\widetilde{S}_1)$
TC^1 for $\widetilde{S}_1$ and set values Q_{out_2} for $\widehat{\mathcal{K}}(\widetilde{S}_2)$
TC^1 for $\widetilde{S}_2$ and set values Q_{out_3} for $\widehat{\mathcal{K}}(\widetilde{S}_3)$
TC^1 for $\widetilde{S}_3$ and set values Q_{out_4} for $\widehat{\mathcal{K}}(\widetilde{S}_4)$
TC^1 for $\widetilde{S}_4$ and set values Q_{out_5} for $\widehat{\mathcal{K}}(\widetilde{S}_5)$
TC^1 for $\widetilde{S}_5$ and set values Q_{out_6} for $\widehat{\mathcal{K}}(\widetilde{S}_6)$
TC^1 for $\widetilde{S}_6$ and set values Q_{out_7} for $\widehat{\mathcal{K}}(\widetilde{S}_7)$
TC^{1}for $\widetilde{S}_7$

Fig. 7 A schematic view of the balanced binary tree method is shown for a large-child path $\widetilde{S}_0, \ldots, \widetilde{S}_7$. Except for the rightmost one, every box in the bottom row represents μp^{μ} many circuits, one for every possible result of $\widehat{\mathcal{K}}(\widetilde{S}_i)$ and every possible κ

as weights in the gadgets in the 4-connected components can be replaced by $ab^{-1} \in \mathbb{Z}_p$.

It remains to combine the results of the different primes from the Chinese remainder representation. This finishes the proof of Theorem 5.2.

Kulkarni, Mahajan, and Varadarajan [15] showed that for a any class of bipartite graphs which is

- closed under edge deletion and
- the number of perfect matchings can be computed in NC,

a perfect matching can be *constructed* in NC. More precisely, the construction is a NC^2-computation relative to counting perfect matchings in one level of the NC^2-circuit. Since K_5-free graph are closed under edge-deletion, we get the following corollary of Theorem 5.2.

Corollary 5.8 *Constructing a perfect matching in a bipartite K_5-free graph is in* TC^2.

5.3 Excluding a Singly-Crossing Graph

Robertson and Seymour [20] define the notion of a singly-crossing graph.

Definition 5.9 The *crossing number* of a graph G is the minimum number c such that G has an embedding in the plane with c edge crossings. A graph H is *singly-crossing* if H is isomorphic to a minor of a graph G with crossing number at most one.

For example, K_5 and $K_{3,3}$ have crossing number one and are therefore singly-crossing. But a singly-crossing graph can have a crossing number larger than one. Figure 8 shows an example.

Robertson and Seymour showed a decomposition for H-free graphs, for any singly-crossing graph H, which is similar to that for K_5-free and $K_{3,3}$-free graphs. The only difference is that the 4-connected components in the last step of the decomposition may be either planar or of *bounded treewidth*.

Theorem 5.10 [20] *For every singly-crossing graph H there is a constant w_H such that every H-free 4-connected graph is planar or has treewidth at most w_H.*

Demaine et al. [6] showed that for any singly-crossing graph H, the decomposition of an H-free graph G into planar components and components of bounded treewidth can be computed in polynomial time. A technical part in the paper of Demaine et al. is to show that the components they compute in their decomposition are all minors of G, and therefore Theorem 5.10 is applicable to these components. However, their decomposition is slightly different from the one presented in Section 3. A technical assumption made by Demaine et al. is that we should not split a component along a separating triple τ if it generates just two split components and one of them has only one vertex besides τ. We can easily adapt this in our decomposition algorithm in Section 3: instead of putting our gadget for the single vertex that would be split off by τ, we simply leave the vertex inside triangle τ. This maintains planarity and the number of perfect matchings. With this modification, the proof in [6, Lemma 5] shows that the components we compute are all minors of G. Furthermore, these components are either planar or have bounded treewidth [6, Theorem 2].

Moreover, Elberfeld, Jakoby, and Tantau [9] showed that the number of perfect matchings of graphs with bounded treewidth can be computed in logspace. This remains true for weighted graphs with polynomial bounded weights, because an edge of weight w can be replaced by a gadget of w unweighted many parallel edges as shown in Fig. 9.

Putting things together, we conclude that our parallel counting algorithm extends to the case of graphs with an excluded singly-crossing minor.

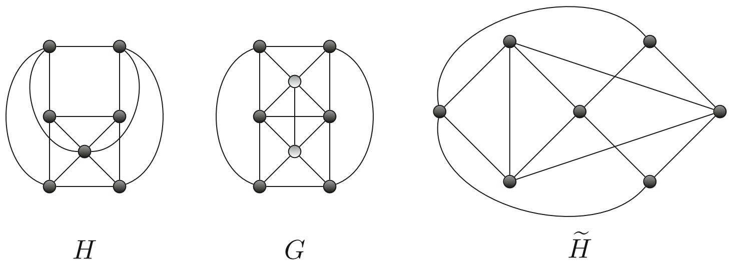

Fig. 8 Graph H has crossing number 2 and is singly-crossing, since it is isomorphic to the minor of G obtained by contracting the edge between the gray vertices [6]. Graph $\widetilde{H}$ is not singly-crossing

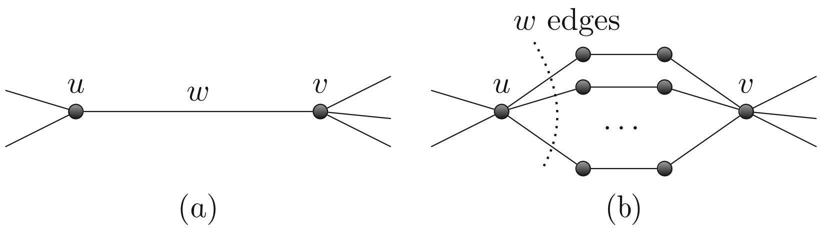

Fig. 9 A gadget to reduce the weighted perfect matching problem with polynomially bounded weights to the unweighted perfect matching problem. The edge $e = (u, v)$ with weight w shown in (**a**) is replaced by the gadget shown in (**b**). For a perfect matching that does *not* use edge e, there is a unique corresponding perfect matching with the gadget that uses the w middle edges of the gadget. For a perfect matching that does use edge e there are w corresponding perfect matchings with the gadget

Corollary 5.11 *For every singly-crossing graph H, the number of perfect matchings in H-free graphs can be computed in* TC^2.

We remark that Curticapean [5] obtained the polynomial time version of Corollary 5.11 independently of our work.

6 Discussion

We presented efficient sequential and parallel algorithms to compute the number of perfect matchings in K_5-free graphs. Our result remains true for H-free graphs, where H is a singly-crossing graph. This extends the work of Kasteleyn for planar graphs and of Little and Vazirani for $K_{3,3}$-free graphs.

After a talk about this result, Michael Saks asked for which minors the counting problem for perfect matching remains #P-hard. Observe that the singly-crossing graph H in Fig. 10 has a K_5 and a $K_{3,3}$ as a minor. Hence it is possible to count even on graph classes which are neither K_5- nor $K_{3,3}$-free (but H-free). It is an interesting open question for which minors counting remains #P-hard.

With respect to the construction problem for perfect matching, recall that we still need the restriction to bipartite graphs to construct a perfect matching in parallel. We pushed this now to bipartite H-free graphs, for any singly-crossing graph H. But it remains a puzzling open question if a perfect matching can be constructed in parallel in non-bipartite graphs, even in the planar case.

Valiant [24] gave a number of holographic reductions to planar perfect matchings. By our result, perfect matchings can be efficiently counted in a much broader class

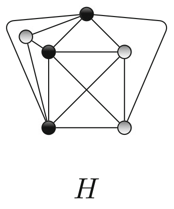

Fig. 10 The singly-crossing graph H has K_5 and $K_{3,3}$ as a minor

of graphs. Hence one can ask whether there are holographic algorithms for a wider range of problems.

Acknowledgments We want to thank Radu Curticapean for pointing us to the literature about graphs that have no singly-crossing minor which lead to Corollary 5.11. We are greatful to Rohit Gurjar for indicating that our result on counting perfect matchings also yields the construction of a perfect matching (Corollary 5.8).

References

1. Barahona, F.: Balancing signed toroidal graphs in polynomial time. Technical report, University of Chile (1983)
2. Di Battista, G., Tamassia, R.: Incremental planarity testing. In: IEEE Symposium on Foundations of Computer Science (FOCS), pp. 436–441 (1989)
3. Di Battista, G., Tamassia, R.: On-line maintenance of triconnected components with SPQR-trees. Algorithmica **15**(4), 302–318 (1996)
4. Cayley, A.: Sur les déterminants gauches. J. Pure Appl. Math. **38**, 93–96 (1847)
5. Curticapean, R.: Counting perfect matchings in graphs that exclude a single-crossing minor. arXiv:1406.4056 (2014)
6. Demaine, E.D., Hajiaghayi, M., Nishimura, N., Ragde, P., Thilikos, D.M.: Approximation algorithms for classes of graphs excluding single-crossing graphs as minors. J. Comput. Syst. Sci. **69**(2), 166–195 (2004)
7. Datta, S., Nimbhorkar, P., Thierauf, T., Wagner, F.: Isomorphism for $K_{3,3}$-free and K_5-free graphs is in log-space. In: Proceedings of the 29th Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS), pp. 145–156 (2009)
8. Edmonds, J.: Paths, trees, and flowers. Can. J. Math. **17**, 449–467 (1965)
9. Elberfeld, M., Jakoby, A., Tantau, T.: Logspace versions of the theorems of Bodlaender and Courcelle. In: 51st Annual IEEE Symposium on Foundations of Computer Science (FOCS), pp. 143–152 (2010)
10. Galbiati, G., Maffioli, F.: On the computation of pfaffians. Discrete Appl. Math. **51**(3), 269–275 (1994)
11. Gibbons, A., Rytter, W.: Efficient Parallel Algorithms. Cambridge University Press, Cambridge (1988)
12. Harary, F.: Graph Theory. Addison-Wesley, Reading (1969)
13. Hopcroft, J.E., Tarjan, R.E.: A $V \log V$ algorithm for isomorphism of triconnected planar graphs. J. Comput. Syst. Sci. **7**(3), 323–331 (1973)
14. Kasteleyn, P.W.: Graph theory and crystal physics. In: Harary, F. (ed.) Graph Theory and Theoretical Physics, pp. 43–110. Academic, New York (1967)
15. Kulkarni, R., Mahajan, M., Varadarajan, K.: Some perfect matchings and perfect half-integral matchings in NC. Chic. J. Theor. Comput. Sci. **2008**(4), 1–26 (2008)
16. Kuratowski, K.: Sur le probléme des courbes gauches en topologie. Fundam. Math. **15**, 271–283 (1930)
17. Little, C.H.C.: An extension of Kasteleyn's method of enumerating the 1-factors of planar graphs. In: Holton, D.A. (ed.) Combinatorial Mathematics, volume 403 of Lecture Notes in Mathematics, pp. 63–72. Springer, Berlin Heidelberg (1974)
18. Miller, G.L., Ramachandran, V.: A new graph triconnectivity algorithm and its parallelization. Combinatorica **12**, 53–76 (1992)
19. Mahajan, M., Subramanya, P.R., Vinay, V.: The combinatorial approach yields an NC algorithm for computing Pfaffians. Discrete Appl. Math. **143**(1–3), 1–16 (2004)
20. Robertson, N., Seymour, P.: Excluding a graph with one crossing. In: Graph Structure Theory, pp. 669–675. American Mathematical Society (1993)
21. Tutte, W.T.: Connectivity in Graphs. University of Toronto Press, Toronto (1966)
22. Thierauf, T., Wagner, F.: Reachability in $K_{3,3}$-free graphs and K_5-free graphs is in unambiguous log-space. Chic. J. Theor. Comput. Sci., To appear (2014)
23. Valiant, L.: The complexity of computing the permanent. Theor. Comput. Sci. **8**, 189–201 (1979)

24. Valiant, L.: Holographic algorithms. SIAM J. Comput. **37**(5), 1565–1594 (2008)
25. Vazirani, V.: NC algorithms for computing the number of perfect matchings in $K_{3,3}$-free graphs and related problems. Inf. Comput. **80**(2), 152–164 (1989)
26. Vollmer, H.: Introduction to Circuit Complexity. Springer, Berlin (1999)
27. Wagner, K.: Über eine Eigenschaft der ebenen Komplexe. Math. Ann. **114**(1), 570–590 (1937)