Computing Large Matchings Fast*

Ignaz Rutter† Alexander Wolff‡

Abstract

In this paper we present algorithms for computing large matchings in 3-regular graphs, graphs with maximum degree 3, and 3-connected planar graphs. The algorithms give a guarantee on the size of the computed matching and take linear or slightly superlinear time. Thus they are faster than the best-known algorithm for computing maximum matchings in general graphs, which runs in $O(\sqrt{n}m)$ time, where n denotes the number of vertices and m the number of edges of the given graph. For the classes of 3-regular graphs and graphs with maximum degree 3 the bounds we achieve are known to be best possible.

We also investigate graphs with block trees of bounded degree, where the d-block tree is the adjacency graph of the d-connected components of the given graph. In 3-regular graphs and 3-connected planar graphs with bounded-degree 2- and 4-block trees, respectively, we show how to compute *maximum* matchings in slightly superlinear time.

1 Introduction

Recall that a *matching* is a set of independent (i.e., pairwise non-incident) edges in a graph. A *maximum* matching is one of maximum cardinality, and a *maximal* matching cannot be enlarged by adding edges. The problem of finding maximum matchings in graphs has a long history dating back to Petersen's theorem [22], which states that every biconnected 3-regular graph has a *perfect* matching, i.e., a matching that matches every vertex. Finding maximum matchings, or large matchings in general, has many applications, see for example the book on matching theory of Lovász and Plummer [17]. To-date the asymptotically fastest (but rather complicated) algorithm for finding maximum matchings in general graphs runs in $O(\sqrt{n}m)$ time [18], where n and m are the numbers of vertices and edges of the given graph, respectively. Only recently faster algorithms for dense graphs and for planar graphs have been suggested. They are based on fast matrix multiplication (which, as a tool, is not very practical) and run in $O(n^\omega)$ time for dense graphs [19] and $O(n^{\omega/2})$ time for planar graphs [20], where $\omega \leq 2.38$ is the exponent in the running time of the best-known matrix-multiplication algorithm [8]. However, for practical purposes often slower, but less complicated algorithms are used: both LEDA [1] and the Boost Graph Library [26] provide maximum-matching algorithms with a running time of $O(nm\alpha(n,m))$ that are based on repeatedly finding augmenting paths [9, 29].

There has been a sequence of more and more general characterizations of graphs with perfect matchings [22, 11, 33], which are special maximum matchings. This has also led to algorithms that test the existence of or compute perfect matchings in $o(\sqrt{n}m)$ time in, e.g., bipartite k-regular graphs [25, 7], 3-regular biconnected graphs [4], and subgraphs of regular grids [32, 12, 16]. The last four algorithms all work in linear time for the corresponding subclasses of planar graphs. There is also a fast algorithm for finding unique maximum matchings [10]. It takes $O(m \log^4 n)$ time in general and $O(n \log n)$ time in planar graphs.

However, although the theory of matchings is a very well-researched area, there has not been a comprehensive investigation of graph classes where maximum matchings or matchings of guaranteed size can be computed faster than matchings in general graphs, i.e., in $o(\sqrt{n}m)$ time. This paper is a first step into this direction. Our work was inspired by and addresses some open questions of a recent paper of Biedl et al. [5] that gives tight bounds on the sizes of maximal and maximum matchings in certain graph classes. Note that, in order to establish bounds on the size of matchings that depend on n, one has to forbid isolated vertices. In this paper we assume that graphs are connected since matchings can be computed for each connected component separately. The analysis of Biedl et al. uses the d-block tree $\mathcal{T}_d$, i.e., the adjacency graph of the d-(vertex-)connected components of the given graph. The parameter of interest is ℓ_d, the number of leaves of this tree. The bounds of Biedl et al. fall in two categories, those that use ℓ_d (type-2 bound) and those where ℓ_d has been replaced by

*Work supported by grant WO 758/4-3 of the German Research Foundation (DFG).

†Fakultät für Informatik, Universität Karlsruhe, Germany. http://i11www.iti.uni-karlsruhe.de/people/rutter

‡Faculteit Wiskunde en Informatica, Technische Universiteit Eindhoven, the Netherlands. http://www.win.tue.nl/~awolff

upper bounds on ℓ_d for the corresponding graph class (type-1 bound). For example, Biedl et al. show that every 3-regular graph has a matching of size at least $(3n - 2\ell_2)/6$. Using that $\ell_2 \leq (n + 2)/6$ for 3-regular graphs leads to a bound of $(4n - 1)/9$ for the matching size in this graph class. The work of Biedl et al. improves some of the earlier results of Nishizeki and Baybars [21] who investigated lower bounds on the size of maximum matchings in planar graphs depending on the minimum degree (3–5), the connectivity (1–4), and the number of vertices of the graph.

Our first and main result is that we "implement" in $O(n \text{ polylog } n)$ time all of the bounds of Biedl et al. except for the type-2 bound for 3-connected planar graphs, see Table 1. Their bound of $(2n + 4 - \ell_2)/4$ is without the bold 6. Our most urgent open question is how to close this gap. Our general approach is as follows. We use block trees to grasp the coarse structure of the graph. They help us to quickly decompose the graph into pieces with desirable properties (such as higher connectivity). We then compute matchings locally and put these local results together to form a (near-) maximum matching in the whole graph. We treat trees is Section 2, turn to maxdeg-3 graphs (i.e., graphs of maximum degree 3) in Section 3 and deal with 3-connected planar graphs in Section 4.

As an example, one of these algorithms finds matchings of size at least $(3n-n_2-2\ell_2)/6$ in maxdeg-3 graphs, where n_2 denotes the number of degree-2 vertices, see Section 3. Such graphs arise naturally when converting triangulations into quadrangulations [23]. Biedl et al. [5] have shown that this bound is tight, but their original construction has no degree-2 vertices, i.e., $n_2 = 0$. They give another construction with $n_2 = 3n/5$, but that graph has a matching of size $2n/5$, which is larger than $(n - 1)/3$, the corresponding type-1 bound. Therefore Biedl et al. pose the question whether there are graphs with a significant number of degree-2 vertices for which the bound $(3n - n_2 - 2\ell_2)/6$ is actually sharp. We answer this question in the affirmative. Our construction uses roughly $n/3$ degree-2 vertices. This is our second result.

Our third and final result concerns the fast computation of maximum matchings in special 3-regular and special 3-connected planar graphs. Note that Petersen's theorem is actually slightly stronger than stated above. It says that every 3-regular graph whose 2-block tree has maximum degree 2 (i.e., is a path) contains a perfect matching. Biedl et al. [4] have shown how to compute perfect matchings in such a graph in $O(n \log^4 n)$ time and in $O(n)$ time is the graph is additionally planar. We extend the findings of Biedl et al. by showing how to compute a maximum matching in 3-regular graphs with 2-block tree of *constant* maximum degree. Our algorithm takes $O(n \log^4 n)$ time in the general case and $O(n)$ time in the planar case. It is based on dynamic programming and on administrating which and how many vertices are matched in the interfaces between the 2-connected components. Note that for maxdeg-3 graphs 2-vertex connectivity (biconnectivity) and 2-edge connectivity (bridge-connectivity) are equivalent. We apply a similar technique to 3-connected planar graphs with bounded-degree 4-block tree. This yields maximum matchings in such graphs in $O(n\alpha(n))$ time, see Section 5.

For the 3-regular case we actually use the algorithm of Biedl et al. as a subroutine. The bottleneck of that algorithm is the dynamic maintenance of the 2-connected components of a graph. Using a data structure of Holm et al. [13] yields a query time of $O(\log^4 n)$. Thorup [31] has claimed to have a data structure with $O(\log^3 n \log\log n)$ query time. This and any further improvements would immediately improve the $O(\log^4 n)$-factors in the running time of the algorithm of Biedl et al. and of our algorithms, see Table 1.

Although our fast maximum matching algorithms can only handle special cases of 3-connected planar and 3-regular graphs, these results are of general interest since Biedl [3] showed that there exists a linear-time reduction from maximum matching in arbitrary graphs to maximum matching in 3-regular graphs and from maximum matching in planar graphs to maximum matching in triangulated (i.e., 3-connected) planar graphs of maximum degree 9. Biedl's results make it unlikely that there are near-linear-time algorithms for much wider subclasses of 3-regular graphs and 3-connected planar graphs.

For full proofs we refer the reader to the long version [24] of this article.

2 Trees

We first compute maximum matchings in trees and then use this result to find matchings in more complex graph classes: maxdeg-3 graphs and 3-connected planar graphs. Although the techniques in this section are quite simple, they suffice to reach some of the bounds given by Biedl et al. [5].

Consider the following simple algorithm PickLeafEdges that takes an arbitrary graph G as input and outputs a set M of edges in G, which is computed as follows. Initially M is empty. As long as G has a leaf (i.e., a degree-1 vertex), the unique edge e incident to the leaf is put in M and both endpoints of e are removed from G with all their incident edges. The algorithm yields the following well-known theorem [2].

graph class		bound on matching size type-1	bound on matching size type-2	runtime $O(\cdot)$
3-regular		$(4n-1)/9$	$(3n-2\ell_2)/6$	$n \log^4 n$
maxdeg-3		$(n-1)/3$	$(3n-n_2-2\ell_2)/6$	$n \mid n \log^4 n$
3-connected, planar, $n \geq 10$		$(n+4)/3$	$(2n+4-\mathbf{6}\ell_4)/4$	$n \mid n\,\alpha(n)$
triangulated, planar		$(2n+4-2\ell_4)/4$		n
maxdeg-k		$(n-1)/k$		n
3-regular,	bounded-deg 2-block tree	maximum		$n \log^4 n$
3-regular, planar,	bounded-deg 2-block tree	maximum		n
3-connected, planar,	bounded-deg 4-block tree	maximum		$n\,\alpha(n)$

Table 1: Our results. The partition in type-1 and type-2 bounds follows the work of Biedl et al. [5]. Our fast algorithms achieve all of theirs bounds (first three rows) except the type-2 bound for 3-connected planar graphs. Their bound is without the bold 6. The function $\alpha(n) := \alpha(n,n)$ denotes the slowly growing inverse of the Ackermann function.

THEOREM 2.1. *Let G be a graph, let $M =$ PICKLEAFEDGES(G), let $G' = G - \bigcup_{uv \in M}\{u,v\}$, and let M' be a maximum matching in G'. Then $M \cup M'$ is a maximum matching in G.*

Note that if we apply PICKLEAFEDGES to a tree, edges are picked until the remaining graph G' is empty. This shows that the following corollary holds.

COROLLARY 2.1. *Applying* PICKLEAFEDGES *to a tree yields a maximum matching in linear time.*

THEOREM 2.2. *Let T be a tree with n vertices and maxdeg k. Then a maximum matching of T has size at least $(n-1)/k$.*

Proof. When PICKLEAFEDGES matches a leaf u to its parent v and removes both vertices, at most k edges are removed and the matching is enlarged by 1. There are $n-1$ edges, so this can be done at least $(n-1)/k$ times. □

This thereom yields interesting results for maxdeg-3 graphs and 3-connected planar graphs: we first find a spanning tree of bounded degree and then a maximum matching in the spanning tree. Clearly this is a matching in the original graph.

COROLLARY 2.2. *Let G be a maxdeg-3 graph. Then G has a matching of size at least $(n-1)/3$, and such a matching can be found in linear time.*

Proof. First we find a spanning tree T of G in linear time, e.g., by breadth-first search. Then T also has maximum degree at most 3. By Corollary 2.1 we can find a maximum matching in T in linear time, and by Theorem 2.2 it has size at least $(n-1)/3$. □

This is one of the type-1 bounds of Biedl et al. [5], see Table 1. The same technique can be used for maxdeg-k graphs, leading to a matching of size at least $(n-1)/k$. However, this is a rather weak bound. We can achieve better bounds by guaranteeing a good upper bound on the maximum degree of our spanning tree.

COROLLARY 2.3. *Let G be a 3-connected planar graph. Then we can find in G a matching of size at least $(n-1)/3$ in linear time and, if $n \geq 10$, a matching of size $(n+4)/3$ in linear time.*

Proof. A maxdeg-3 spanning tree T of G can be computed in linear time [27]. Then Corollary 2.2 yields in linear time a matching of size at least $(n-1)/3$ in T.

Note that this bound is only by 5/3 smaller than the type-1 bound $(n+4)/3$ of Biedl et al. [5] for 3-connected planar graphs with $n \geq 10$, see Table 1. Hence we can reach their bound by finding at most two augmenting paths, which takes $O(n)$ time [29]. □

3 Graphs with Maximum Degree 3

In this section we consider matchings in maxdeg-3 graphs. We first consider 3-regular graphs and give an algorithm that achieves the tight bounds of Biedl et al. [5] (see Table 1). Then we show how to extend this algorithm to arbitrary maxdeg-3 graphs. We also give a family of maxdeg-3 graphs for which the bound of Biedl et al. is tight. The novelty is that each graph of the family contains a large fraction of degree-2 vertices.

3.1 3-regular graphs. Biedl et al. [5] have shown that every 3-regular graph has a matching of size at least $(4n-1)/9$, or more generally of size $(3n-2\ell_2)/6$, where ℓ_2 denotes the number of leaves of the 2-block tree $\mathcal{T}_2$. We will show how to find such matchings in $o(\sqrt{n}m)$ time. This has been known only for a special case: Biedl

et al. [4] have "implemented" Petersen's theorem. Given a 3-regular graph with $\ell_2 \leq 2$ they can find a perfect matching in that graph in $O(n \log^4 n)$ time.

We present a constructive proof of the bound $(3n - 2\ell_2)/6$ that yields an algorithm with running time $O(n \log^4 n)$ for finding such a matching. The basic idea is to cut off leaves in the 2-block tree such that a small number of *free*, i.e. unmatched, vertices can be guaranteed. Recall that a *bridge* is an edge whose removal disconnects the graph.

We use a slightly simpler definition of the 2-block tree than Biedl et al. [5]. Their 2-block tree has a vertex for each biconnected component of G and a vertex for each *cut vertex* of G, i.e., for each vertex whose removal decomposes G. (The definition of the tree edges is obvious.) Since our graphs have maximum degree 3, each cut vertex must be incident to a bridge. (This observation yields the equivalence of 2-edge and 2-vertex connectivity in maxdeg-3 graphs.) Thus our simplified 2-block tree only has a node for each biconnected component of G and an edge for each bridge in G. Note that the number of leaves in both trees is the same. From now on we will refer to vertices of the d-block tree $\mathcal{T}_d$ as *nodes* (as opposed to the *vertices* of the given graph). We have the following result.

THEOREM 3.1. *Let G be a 3-regular graph whose 2-block tree has ℓ_2 leaves. Then G has a matching of size at least $(3n-2\ell_2)/6$. This matching can be chosen such that every free vertex is incident to a bridge.*

Proof. We use induction on ℓ_2. If $\ell_2 \in \{1, 2\}$, then there exists a perfect matching by Petersen's theorem [22]. For the cases $\ell_2 \in \{3, 4\}$ refer to the long version of this article [24]. Now let $\ell_2 \geq 5$. We cut off three parts of the graph such that we remove three leaves from the 2-block tree $\mathcal{T}_2$ of G at the cost of at most two free vertices. Then the induction hypothesis takes effect.

We first show that there always exist three leaves that are suitable for removal. Choose an arbitrary leaf node ℓ of $\mathcal{T}_2$ and walk upwards until a node v_ℓ of degree at least 3 is reached. The last edge of the traversal corresponds to a bridge b_ℓ after whose removal G decomposes into two components: the *branch* containing the leaf component ℓ and the *main component* now containing one degree-2 vertex. The 2-block tree of the branch is a path, and the 2-block tree of the main component has $\ell_2 - 1$ leaves.

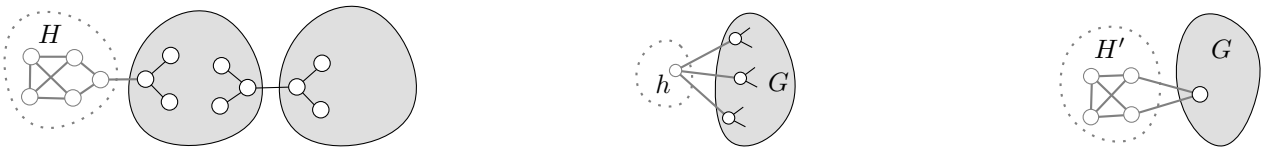
Now assume that every leaf ℓ of $\mathcal{T}_2$ has a pointer to the tree node v_ℓ defined as above. If, after removing b_ℓ, the degree of v_ℓ in the tree is still at least 3, there is nothing to do. Otherwise, there is at most one other leaf ℓ' with $v_\ell = v_{\ell'}$. It cannot be cut off at $v_{\ell'}$ anymore since this would not reduce the number of leaves in the 2-block tree of the main component. Hence by cutting off a leaf ℓ we make at most one other leaf ℓ' invalid. Since $\ell_2 \geq 5$, we can cut off three branches such that the number of leaves in the 2-block tree of the main component decreases by 3 in total.

After removing the three bridges G decomposes into four components: three branches, each with one degree-2 vertex, and the *main component* with three degree-2 vertices. The 2-block tree of the main component has $\ell_2 - 3$ leaves. Now we restore 3-regularity in each component. We extend each branch B by attaching the helper graph H depicted in Figure 1a to the unique degree-2 vertex, which we denote by v_B. Now Petersen's theorem yields a perfect matching in each of the extended branches. Then we remove H from each branch B. This results only in v_B becoming free. Thus so far we have three free vertices, all incident to bridges. Now consider the main component. We add a new vertex h and connect it to each of the three degree-2 vertices, see Figure 1b. Now the main component is again 3-regular. Its 2-block tree still has $\ell_2 - 3$ leaves. By induction the main component has a matching that leaves at most $2(\ell_2 - 3)/3 = 2\ell_2/3 - 2$ vertices free, each incident to a bridge. Since the main component was already connected the new vertex h is not incident to a bridge and hence not free. When we remove h, one of the incident degree-2 vertices becomes free and can be matched to the free vertex in the corresponding branch. Thus in total we have created at most $2\ell_2/3$ free vertices, each incident to a bridge. □

Since the proof is constructive, we simply implement each step of the proof. We use the algorithm of Biedl et al. [4] for computing matchings in the branches and for the base case. We only need to make a linear number of cuts because $\ell_2 \leq (n + 2)/6$ [5]. After each cut we just add a constant number of vertices. Since each vertex is in exactly one component, the computation of all partial matchings takes $O(n \log^4 n)$ time in total.

The 2-block tree $\mathcal{T}_2$ of G can be computed in linear time [28], but $\mathcal{T}_2$ changes drastically when we link the new vertex h to the three degree-2 vertices of the main component. The addition of h creates a new super vertex in $\mathcal{T}_2$ that consists of vertices a, b, and c corresponding to the components with the three degree-2 vertices in G and of all nodes of $\mathcal{T}_2$ that lie on the unique paths between a, b, and c. Thus it remains to show how to efficiently maintain the 2-block tree of the main component and the leaf pointers. We call a branch *good* if its removal decreases the number of leaves in the main component.

LEMMA 3.1. *Given a 3-regular graph G, we can in $O(n\alpha(n))$ total time repeatedly determine three good*



(a) in a branch (b) in the main component (c) at degree-1 vertices

Fig. 1: Restoring 3-regularity.

branches of G, remove the branches, link the degree-2 vertices of the main component to a new vertex, and update the 2-block tree $\mathcal{T}_2$ of the main component. The process ends when $\mathcal{T}_2$ has less than five leaves.

Theorem 3.1 and Lemma 3.1 together yield the following theorem.

THEOREM 3.2. *Let G be a 3-regular graph whose 2-block tree has ℓ_2 leaves. Then we can find in G a matching of size at least $(3n - 2\ell_2)/6$ in $O(n \log^4 n)$ time.*

3.2 Maxdeg-3 graphs. In this section we extend the algorithm of the previous subsection to maxdeg-3 graphs. Let G be such a graph and let n_2 denote the number of degree-2 vertices of G. For now we assume that G has no degree-1 vertices. For every three degree-2 vertices we add a helper vertex and link it to the three vertices. Note that this does not increase the number of leaves of $\mathcal{T}_2$. If n_2 is a multiple of 3 this results in a 3-regular graph. By Theorem 3.2 we can find a matching of size at least $(3n - 2\ell_2)/6$ in $O(n \log^4 n)$ time in this graph. Removing the $n_2/3$ added vertices results in at most $n_2/3$ free vertices and a matching of size at least $(3n - n_2 - 2\ell_2)/6$.

If n_2 is no multiple of 3, we first add helper vertices as before until there are at most two degree-2 vertices left. If there are two degree-2 vertices left, we connect them by an additional edge. If there is only one degree-2 vertex left, we link it to the helper graph H, see Figure 1a. Using Theorem 3.2 we compute a matching in the resulting 3-regular graph. Removing the added vertices results in a matching M of size at least $(3n - n_2 - 2\ell_2)/6 - 1$. If M actually contains exactly $(3n - n_2 - 2\ell_2)/6 - 1$ edges, we can enlarge M by one edge by computing an augmenting path in G in $O(n)$ time. This is due to the fact that we know G has a matching of size $(3n - n_2 - 2\ell_2)/6$. Making G 3-regular as above takes $O(n)$ time, too.

Finally we also admit degree-1 vertices. Each such vertex is a leaf in the 2-block tree. Hence we can make G 3-regular by linking a copy of the helper graph H' depicted in Figure 1c to each degree-1 vertex. This neither changes ℓ_2 nor the number of free vertices and can be done in linear time. We summarize:

THEOREM 3.3. *If G is a maxdeg-3 graph with n_2 degree-2 vertices whose 2-block tree has ℓ_2 leaves, we can find in G a matching of size at least $(3n - n_2 - 2\ell_2)/6$ in $O(n \log^4 n)$ time.*

Now we construct a family of graphs that shows that the bound $(3n - n_2 - 2\ell_2)/6$ holds even in the presence of a large fraction of degree-2 nodes. This answers an open question posed by Biedl et al. [5] in the affirmative.

Consider the graphs in Figure 2. We denote the graph in the right gray box by G_0 and call w its root. The bound $(3n - n_2 - 2\ell_2)/6$ is tight for this graph. We construct G_i inductively: we attach a copy of G_0 to G_{i-1} by connecting their roots via two vertices (one of degree 3 and one of degree 1) and three edges as shown in Figure 2. The resulting graph has $12i + 10$ vertices, $4i + 2$ of which have degree 2. Thus the fraction of degree-2 vertices tends to 1/3. Note that since G_i is a tree, it is essentially its own 2-block trees. Hence ℓ_2 equals the number of leaves of G_i, i.e., $4i + 5$. The algorithm PICKLEAFEDGES of Section 2 yields a maximum matching in G_i with $4i+3$ edges. This shows that the bound $(3n - n_2 - 2\ell_2)/6$ is tight for all G_i.

4 3-Connected Planar Graphs

In this section we give an algorithm for finding a matching of size at least $(2n+4-6\ell_4)/4$ in 3-connected planar graphs. In graphs where every separating triplet

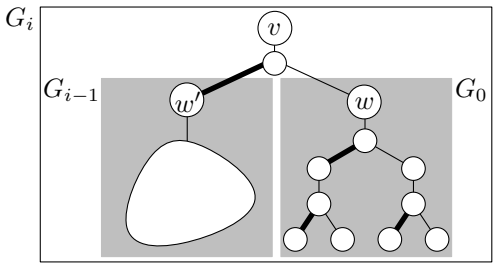


Fig. 2: Maxdeg-3 graphs with many degree-2 vertices and small maximum matching.

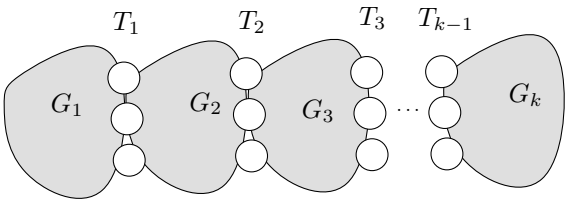


Fig. 3: Graph whose 4-block tree is a path.

is a triangle (e.g., in triangulated graphs) we can even guarantee a size of $(2n + 4 - 2\ell_4)/4$. This is very close to the tight bound $(2n + 4 - \ell_4)/4$ that Biedl et al. [5] gave for 3-connected planar graphs. We use an approach similar to Section 3.1. We cut off leaves of the 4-block tree until it has only two leaves left. To implement this, we first need an algorithm for finding matchings in 3-connected planar graphs whose 4-block tree is a path. Biedl et al. [5] have shown that such a graph always has a perfect or a *nearly perfect* matching, i.e., a matching that matches all vertices but one.

If a graph G is 3- but not 4-connected, there exists a separating vertex triplet $T = \{u, v, w\}$ such that $G - T$ has more than one component. For each of these components C we consider the graph $C + T$ and add the *dummy* edges uv, vw, uw if they did not exist already. We iterate this process until all components are 4-connected. These are the 4-connected components of G. The 4-block tree of G contains one node for every 4-connected component of G. Since we consider only 3-connected planar graphs, every separating triplet separates only two components, otherwise G would contain a subdivision of $K_{3,3}$ (contradicting planarity). So we simply link two nodes of the 4-block tree by an edge if the corresponding 4-connected components share a separating triplet. Note that the definition of Biedl et al. [5] is more general. However, for 3-connected planar graphs both definitions lead to the same value of ℓ_4.

4.1 The 4-block tree is a path. Let G be a 3-connected planar graph whose 4-block tree is a path. If G is 4-connected we can find a Hamiltonian cycle and hence a (nearly) perfect matching in linear time [6]. If G is not 4-connected the basic idea is to find a matching in every block separately and combine them to a matching in G. Let $G_1, \ldots, G_k$ be the 4-connected components of G and for $i = 1, \ldots, k-1$ let T_i be the triplet that separates G_i and G_{i+1}, see Figure 3. Note that consecutive triplets do not need to be disjoint.

If n is odd, choose a face of G_k that is not incident to all vertices of T_{k-1}. Place a new vertex $v^\star$ into this face and connect it to each vertex of the face. Now G has an even number of vertices, G is still 3-connected planar and its 4-block tree a path, and hence has a perfect matching. In particular it follows that it is enough to leave a vertex free in G_k (the one matched to $v^\star$).

One idea would be to find a Hamiltonian cycle in G_1 using the algorithm of Chiba and Nishizeki [6]. However, it seems difficult to extend the corresponding matching to one of G. Therefore we go a different way.

Consider a perfect matching M in a 3-connected planar graph G with $\ell_4 = 2$. Now we restrict M to a 4-connected component C of G. We denote this matching by M'. It is clear that only vertices of C that belong to separating triplets of G are free with respect to M'. Note that the fact that we can combine M' with the rest of M to a perfect matching only depends on which of these vertices are matched and which are free. In particular the combinability is independent of the structure of M' with respect to the vertices of C that do not belong to a separating triplet. The next definition formalizes this idea.

A *matching configuration* of a component G_i is a pair $(T_{i,\text{matched}}, T_{i,\text{free}})$ with $T_{i,\text{matched}} \subseteq T_{i-1}$ and $T_{i,\text{free}} \subseteq T_i$. Such a configuration is called *feasible* if both of the following conditions hold:

- $T_{i,\text{matched}} \cap T_{i,\text{free}} = \emptyset$, and
- there exists a matching M_i in G_i which matches exactly the vertices of $G - (T_{i,\text{matched}} \cup T_{i,\text{free}})$ and uses only edges in G (i.e., no dummy edges).

The first condition makes sure that vertices already matched in G_{i-1} are not used again in G_{i+1}. The second condition makes sure that we can use this configuration to find a perfect matching in G.

The *matching graph* of G is a directed acyclic graph whose vertices correspond to the feasible matching configurations of G. Let $u = (T_{i,\text{matched}}, T_{i,\text{free}})$ and $v = (T_{j,\text{matched}}, T_{j,\text{free}})$ be two vertices of the matching graph. There is an edge uv if $j = i + 1$ and $T_i \setminus T_{i,\text{free}} = T_{j,\text{matched}}$. This is the case if and only if the matching given by u can be extended into v such that the only free vertices are in $T_{j,\text{free}}$. In the situation of Figure 4 there would be an edge between the two vertices representing the first and the last configuration, but not between the first and the second. For ease of description we add a source node with edges to all feasible matching configurations of G_1.

The matching graph has $O(n)$ vertices, since there are only $O(n)$ components and every components has only a constant number of feasible matching configurations. Every perfect matching of G corresponds to a path of length k in the matching graph and vice versa. The path describes a sequence of matching configurations that fit together. For the configuration $(T_{i,\text{matched}}, T_{i,\text{free}})$ of G_i lying on the path there exists a matching M_i that is perfect in $G_i - (T_{i,\text{matched}} \cup T_{i,\text{free}})$.

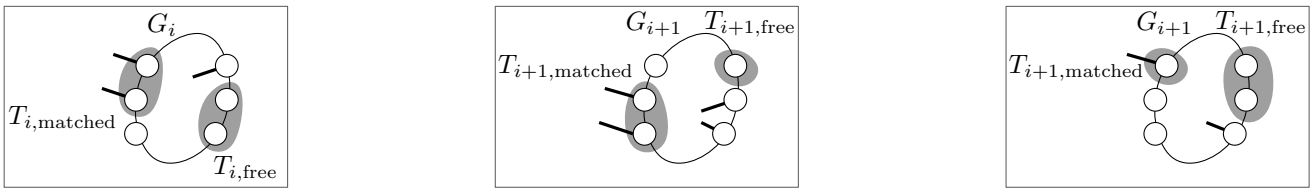


Fig. 4: Examples of matching configurations

Then $M_1 \cup \cdots \cup M_k$ is a perfect matching in G. Such a path can be found in $O(n)$ time by breadth-first search from the source node of the matching graph.

Now we need a fast algorithm for finding the feasible matching configurations of a 4-connected component G_i. Let n_i denote the number of vertices of G_i. Since G_i has only a constant number of matching configurations, it is enough to give an algorithm that can quickly determine feasibility of a given matching configuration. We first compute a (nearly) perfect matching M in G_i by finding a Hamiltonian cycle in $O(n_i)$ time [6]. From M we remove all edges that do not belong to G and all edges incident to vertices we may not use (i.e., vertices in $T_{i,\text{matched}} \cup T_{i,\text{free}}$). This results in $O(1)$ free vertices. Hence if there is a perfect matching in $G_i - (T_{i,\text{matched}} \cup T_{i,\text{free}})$, we can find it in $O(n_i)$ time by computing a constant number of augmenting paths [29]. If the resulting matching is perfect, the configuration is feasible and we store the matching as M_i, otherwise the configuration is not feasible.

Thus we can compute all feasible matching configurations of a component G_i in $O(n_i)$ time. Since every component shares at most six vertices with other components, we can compute the feasible matching configurations for all components in linear time.

Finally, if the graph originally had an odd number of vertices, we have to remove the vertex $v^\star$ added in the beginning, which frees the vertex $v^{\star\star}$ matched to $v^\star$. This yields a perfect matching in $G - v^{\star\star}$, i.e., a nearly perfect matching in G. We summarize our observations as follows.

LEMMA 4.1. *Let G be a 3-connected planar graph whose 4-block tree is a path. Then we can compute a (nearly) perfect matching in G in linear time.*

4.2 Cutting leaves. In this section we apply the algorithm from the previous section to cut off leaves of the 4-block tree $\mathcal{T}_4$ similarly to the way we treated the 2-block tree in Section 3.1. The 4-block tree of a 3-connected planar graph can be computed in $O(n\alpha(n))$ time [14]. The 4-block tree has at most $(2n-4)/3$ leaves [5]. We pick an arbitrary one and walk upwards in the 4-block tree until we reach a component of degree at least 3. This is the place to cut the leaf off. The last used edge corresponds to a separating triplet that we can use to cut off a leaf of the 4-block tree.

We now split the graph at this separating triplet. This results in two components, the *branch* containing the leaf we want to cut off and the *main component*. We add to each of these components the edges between the vertices of the separating triplet if they did not already exist. Now we compute matchings in the main component and in the branch using recursion and the algorithm of the previous section, respectively. The following lemma states that we can combine these matchings without getting too many free vertices.

LEMMA 4.2. *Let G be a 3-connected planar graph. Let B be a branch, let C_{main} be the corresponding main component, and let $T = \{u, v, w\}$ be the triplet that separates B and C_{main}. Let $C'_{\text{main}} = C_{\text{main}} \cup \{uv, vw, wu\}$. Let M_{main} be a matching in C'_{main} and let f be the number of free vertices in C'_{main} with respect to M_{main}. Then there is a matching M of G that leaves at most $f+3$ vertices free.*

This results in an algorithm that produces at most three free vertices for every leaf of the 4-block tree. Hence the matching has size at least $(2n - 6\ell_4)/4$. We can reach the bound $(2n + 4 - 6\ell_4)/4$ by finding at most one augmenting path in linear time. Once the 4-block tree is computed, we can find leaves to cut off in a way similar to Section 3.1. In fact here it is even easier since we can cut off the leaves one by one. Hence the full decomposition of the graph can be done in linear time. We do $O(n)$ splits, and combining the matchings in both components can be done in constant time. By Lemma 4.1 the computation of the matching in the branch takes time linear in the size of the branch. Since two adjacent components share only three vertices, the total number of vertices we process is linear, and the algorithm runs in $O(n\alpha(n))$ time. The following theorem summarizes our discussion.

THEOREM 4.1. *In a 3-connected planar graph whose 4-block tree has ℓ_4 leaves we can compute a matching of size at least $(2n + 4 - 6\ell_4)/4$ in $O(n\alpha(n))$ time.*

In case the graph is a planar triangulation (and hence 3-connected) we can improve this result. This is due to the fact that the edges between the vertices of a separating triplet are already in the graph—every separating triplet is a separating triangle. This means that we can always choose the matching in the branches such that it fits optimally with the matching in the main component.

THEOREM 4.2. *In a triangulated 3-connected planar graph whose 4-block tree has ℓ_4 leaves we can compute a matching of size at least $(2n+4-2\ell_4)/4$ in linear time.*

Proof. The triangulated case is a lot easier than the general 3-connected planar case since we now know that the vertices of a separating triplet form a triangle. Let G be a triangulated 3-connected planar graph. Since G is triangulated, the 4-block tree of G can be computed in linear time [15].

Structurally we do the same as in the general case: we cut off a branch B at a separating triangle T and process the main component recursively. Then we extend the matching M of the main component into the branch. We now show that, in linear time, we can find a matching $M' \supseteq M$ that leaves at most one vertex in the branch free. By induction M' leaves at most $\ell_4 - 1$ vertices free. Hence M' has size at least $(2n+2-2\ell_4)/4$. Computing an augmenting path yields the desired bound in linear time [29].

The branch B is a path of 4-connected components. We go through these components sequentially starting with the one that contains T. Let C be the current component. Let $T' = \{u, v, w\}$ be the triangle separating C from the previous (or the main) component. If C is not the last component in B, let T'' be the triangle separating C from the next component and let u' be a vertex in $T' \setminus T''$. Otherwise let u' be any vertex in $C \setminus T'$. Note that T' contains at most one free vertex. We distinguish two cases. For each case we specify a set $R \subseteq \{u, v, w, u'\}$ such that $C \setminus R$ contains a perfect matching.

Case 1: One vertex, say w, is free.

Then $C-\{u, v\}$ contains a Hamiltonian circuit [30]. The circuit yields a (nearly) perfect matching in $C\setminus\{u, v\}$. If $|C|$ is odd, we can choose the matching such that u' is free. Let $R = \{u, v, u'\}$. If $|C|$ is even, all vertices are matched. Let $R = \{u, v\}$.

Case 2: T' does not contain any free vertices.

If $|C|$ is odd, then $C \setminus \{u\}$ contains a Hamiltonian circuit containing the edge vw [30]. The circuit hence contains a perfect matching P of $C \setminus \{u\}$. It can be chosen such that it contains vw. Now $P \setminus \{vw\}$ is the desired matching. Let $R = T'$.

If $|C|$ is even, then $C \setminus \{u, u'\}$ contains a Hamiltonian circuit with the edge vw [30]. As shown above this circuit contains an appropriate matching. Let $R = T' \cup \{u'\}$.

It remains to analyze the running time. Since C is 4-connected and planar, we can compute *some* Hamiltonian circuit H in C in $O(|C|)$ time [6]. This circuit yields a (nearly) perfect matching M_H in C. From M_H we remove all edges incident to vertices in R. This yields a matching M'_H in $C \setminus R$. By construction of R we get a perfect matching M_C in $C \setminus R$ by computing a constant number of augmenting paths in $C \setminus R$. Clearly, M_C can be computed in $O(|C|)$ time. The linear running time of the whole algorithm can be seen as in the proof of Lemma 4.1. □

5 Graphs with Bounded-Degree Block-Trees

In this section we consider graphs with block trees of bounded degree. Biedl et al. [4] gave an algorithm for computing a perfect matching in a 3-regular graph whose 2-block tree is a path in $O(n \log^4 n)$ time. In Section 4 we showed how to find (nearly) perfect matchings in 3-connected planar graphs whose 4-block trees are paths in linear time. These are special cases where the number of leaves of the corresponding block tree is one or two.

It is clear that if we bound the number of leaves of the block tree by a constant, we can find maximum matchings in maxdeg-3 graphs and 3-connected planar graphs fast. In these cases our algorithms of the previous sections guarantee to find a matching that is smaller than $\lfloor n/2 \rfloor$ by only a constant. Hence we can enlarge the computed matching to a maximum matching by finding a constant number of augmenting paths which is possible in linear time [29].

In this section we relax our previous requirement for the fast computation of maximum matchings: instead of insisting that the block tree has a constant number of leaves, we require only that its degree is bounded. In this way the number of leaves can still be large (i.e., linear in the size of the graph). This also shows which structures make the fast computation of maximum matchings difficult: components with a large number of neighbors.

The technique we use in this section is similar to the one we used in Section 4.1 for the case of 3-connected planar graphs whose 4-block tree is a path. There we could afford to check all local configurations since every component had at most two neighbor components. However, the arguments we used also work for a constant number of neighbor components.

In the case of 3-connected planar graphs whose 4-

block tree is a path, we knew that there exists a (nearly) perfect matching. Hence we could restrict ourselves to considering configurations that can be reached without leaving any vertices free. In the more general setting of arbitrary maximum matchings this is no longer the case. However, instead of just considering the feasibility of the matching configurations as in Section 4.1, we keep track of the number of vertices we must leave free to reach a given configuration. We then use dynamic programming to join local configurations. We now give a sketch of the algorithm for 3-connected planar graphs. The algorithm is exponential in the degree bound.

Let G be a 3-connected planar graph. We first compute its 4-block tree $\mathcal{T}_4$ in $O(n\alpha(n))$ time [14], choose an arbitrary node of $\mathcal{T}_4$, and direct all edges towards it. We call a node a a *predecessor* of b if there is a directed path from a to b in $\mathcal{T}_4$. Then, we process the components in topological order. In this way, whenever we process a component, there is at most one neighbor component that has not been processed before. Let C be such a component and let G_C denote the subgraph of G that consists of all vertices that belong to C or to a predecessor of C. There is at most one separating triplet T that separates C from a component that has not been processed already. We call T the *leaving separating triplet* of C. An important observation is that the extendability of a maximum matching in G_C to a matching of G depends only on which vertices of T are free. Once we have computed all eight possibilities we do not need to consider any vertex of $G_C - T$ again.

For every subset $F \subseteq T$ we check how many vertices we have to leave free for a maximum matching in $G_C - F$. For this check we use the fact that we already have computed the corresponding counters of all predecessors of C. There are only a constant number of neighbors and each of them has a constant number of configurations. To compute the cost of F we now simply check the cost of every possible combination. Since this is only a constant number (even though exponential in the maximum degree of the 4-block tree), this takes asymptotically the same time as checking a single configuration.

Checking the cost of a single configuration is possible in time proportional to the size of the component C by using that C is 4-connected and planar. This means we can then find a Hamiltonian cycle in C in linear time and hence a (nearly) perfect matching. After computing such a matching we remove all dummy edges (e.g., edges only added to make the components 4-connected) and all vertices that are already matched by predecessor components in the given configuration as well as all vertices in F. We remove only a constant number of vertices and edges, resulting in a constant number of free vertices. Thus we can then enlarge the matching to a maximum matching by computing augmenting paths in linear time. The number of free vertices of the given configuration is then just the number of free vertices in the considered component plus the number of free vertices in the predecessor configurations.

By storing along with each counter also the corresponding configuration and matching, we can report a maximum matching in linear time once all components have been processed. The next theorem summarizes this result. Note that the asymptotic running time is dominated by the initial computation of the 4-block tree, which takes $O(n\alpha(n))$ time [14].

THEOREM 5.1. *In a 3-connected planar graph whose 4-block tree has bounded degree we can compute a maximum matching in $O(n\alpha(n))$ time.*

A similar algorithm can be used to compute a maximum matching in a 3-regular graph with bounded-degree 2-block tree. We first compute the 2-block tree in linear time [28] and then work upwards from the leaves in the same way as before. However, here we have only two counters for each component. The vertex incident to the leaving bridge is either matched or free. Note that when combining matchings we can add a bridge if both its incident vertices are free. We compute the cost of each configuration (i.e., the number of free vertices) according to this rule. We must make sure to not match a vertex via two bridges at the same time. However, it is not hard to see that a vertex is either incident to at most one bridge or to three bridges. In the latter case the vertex is actually a component by itself, and this can be checked easily.

The running time for the 3-regular case depends mainly on the running time for computing maximum matchings locally in the 2-connected components. For this we use the algorithm of Biedl et al. [4]. Their algorithm takes $O(n \log^4 n)$ time for the general case and linear time for the planar case.

THEOREM 5.2. *Let G be a 3-regular graph whose 2-block tree has bounded degree. Then we can find a maximum matching in G in $O(n \log^4 n)$ time. If G is planar, the running time reduces to $O(n)$.*

As we pointed out in the introduction, our results are of general interest due to Biedl's linear-time reductions [3]. They make it unlikely that there are near-linear-time algorithms for much wider subclasses of 3-regular graphs and 3-connected planar graphs.

6 Open Questions

Our most burning question deals with 3-connected planar graphs, where we can compute matchings of size

at least $(2n + 4 - \mathbf{6}\ell_4)/4$ in $O(n\alpha(n))$ time. Can we achieve the tight bound $(2n+4-\ell_4)/4$ of Biedl et al. [5] in near-linear time? At least in the triangulated case?

Can we speed up our $O(n \log^4 n)$-time algorithm for 3-regular graphs in the planar case? In the general case we use the algorithm of Biedl et al. [4] as a subroutine, which takes only $O(n)$ time in the planar case. However, our current approach does not preserve planarity.

Are there fast algorithms that implement the bounds of Nishizeki and Baybars [21] for planar graphs? I.e., how can we exploit minimum degrees?

Acknowledgments

We thank Therese Biedl and the anonymous referees for helpful comments.

References

[1] Algorithmic Solutions. The LEDA user manual version 5.2. `www.algorithmic-solutions.info/leda_manual`.

[2] J. Aronson, A. Frieze, and B. G. Pittel. Maximum matchings in sparse random graphs: Karp-Sipser revisited. *Random Structures & Algorithms*, 12(2):111–177, 1998.

[3] T. Biedl. Linear reductions of maximum matching. In *Proc. 12th Annu. ACM-SIAM Sympos. Discrete Algorithms (SODA'01)*, pages 825–826, 2001.

[4] T. Biedl, P. Bose, E. Demaine, and A. Lubiw. Efficient algorithms for Petersen's theorem. *J. Algorithms*, 38:110–134, 2001.

[5] T. Biedl, E. D. Demaine, C. A. Duncan, R. Fleischer, and S. G. Kobourov. Tight bounds on maximal and maximum matchings. *Discrete Math.*, 285(1–3):7–15, 2004.

[6] N. Chiba and T. Nishizeki. The Hamiltonian cycle problem is linear-time solvable for 4-connected planar graphs. *J. Algorithms*, 10:187–211, 1989.

[7] R. Cole, K. Ost, and S. Schirra. Edge-coloring bipartite multigraphs in $O(E \log D)$ time. *Combinatorica*, 21:5–12, 2001.

[8] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. In *Proc. 19th Annu. ACM Conf. Theory Comput. (STOC'87)*, pages 1–6, 1987.

[9] H. N. Gabow. An efficient implementation of Edmonds' algorithm for maximum matching on graphs. *J. ACM*, 23(23):221–234, 1976.

[10] H. N. Gabow, H. Kaplan, and R. E. Tarjan. Unique maximum matching algorithms. *J. Algorithms*, 40(2):159–183, 2001.

[11] P. Hall. On representatives of subsets. *Jour. London Math. Soc.*, 10:26–30, 1935.

[12] P. Hansen and M. L. Zheng. A linear algorithm for perfect matching in hexagonal systems. *Discrete Math.*, 122(1-3):179–196, 1993.

[13] J. Holm, K. de Lichtenberg, and M. Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *J. ACM*, 48(4):723–760, 2001.

[14] A. Kanevsky, R. Tamassia, G. D. Battista, and J. Chen. On-line maintenance of the four-connected components of a graph. In *Proc. 33th Annu. IEEE Sympos. Found. Comput. Sci. (FOCS'92)*, pages 793–801, 1992.

[15] G. Kant. A more compact visibility representation. *Intern. J. Comput. Geom. Appl.*, 7(3):197–210, 1997.

[16] C. Kenyon and E. Rémila. Perfect matchings in the triangular lattice. *Discrete Math.*, 152(1–3):191–210, 1996.

[17] L. Lovász and M. D. Plummer. *Matching Theory*. North Holland, Amsterdam, 1986.

[18] S. Micali and V. V. Vazirani. An $O(\sqrt{|V|} \cdot |E|)$ algorithm for finding maximum matchings in general graphs. In *Proc. 21st Annu. IEEE Sympos. Found. Comput. Sci. (FOCS'80)*, pages 17–27, 1980.

[19] M. Mucha and P. Sankowski. Maximum matchings via Gaussian elimination. In *Proc. 45th Annu. IEEE Sympos. Foundat. Comput. Sci.*, pages 248–255, 2004.

[20] M. Mucha and P. Sankowski. Maximum matchings in planar graphs via Gaussian elimination. *Algorithmica*, 45(1):3–20, 2006.

[21] T. Nishizeki and I. Baybars. Lower bounds on the cardinality of the maximum matchings of planar graphs. *Discrete Math.*, 28(3):255–267, 1979.

[22] J. Petersen. Die Theorie der regulären Graphs. *Acta Mathematica*, 15:193–220, 1891.

[23] S. Ramaswami, P. Ramos, and G. Toussaint. Converting triangulations to quadrangulations. *Computational Geometry Theory and Applications*, 9:257–276, 1998.

[24] I. Rutter and A. Wolff. Computing large matchings fast. Technical Report 2007-19, Fakultät für Informatik, Universität Karlsruhe, 2007. Available at `www.ubka.uni-karlsruhe.de/indexer-vvv/ira/2007/19`.

[25] A. Schrijver. Bipartite edge coloring in $O(\Delta m)$ time. *SIAM J. Comput.*, 28:841–846, 1999.

[26] J. Siek, L.-Q. Lee, and A. Lumsdaine. The Boost Graph Library documentation. `www.boost.org/libs/graph`, visited 07/04/2007.

[27] W.-B. Strothmann. *Bounded Degree Spanning Trees*. PhD thesis, Heinz-Nixdorf-Institut, Universität Paderborn, 1997.

[28] R. E. Tarjan. Depth first search and linear graph algorithms. *SIAM J. Comput.*, 2:146–160, 1972.

[29] R. E. Tarjan. *Data structures and network algorithms*. SIAM, Philadelphia, 1983.

[30] R. Thomas and X. Yu. 4-connected projective-planar graphs are Hamiltonian. *J. Combinat. Theory Ser. B*, 1:114–132, 1994.

[31] M. Thorup. Near-optimal fully-dynamic graph connectivity. In *Proc. 32nd Annu. ACM Sympos. Theory Comput. (STOC'00)*, pages 343–350, 2000.

[32] W. P. Thurston. Conway's tiling groups. *Amer. Math. Monthly*, 97(8):757–773, 1990.

[33] W. T. Tutte. The factorization of linear graphs. *J. Lond. Math. Soc.*, 22:107–111, 1947.