Computing Large Matchings Fast

IGNAZ RUTTER
Universität Karlsruhe (TH), KIT
and
ALEXANDER WOLFF
Technische Universiteit Eindhoven

In this article 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.

Categories and Subject Descriptors: G.2.1 [**Discrete Mathematics**]: Combinatorics; G.2.2 [**Discrete Mathematics**]: Graph Theory; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems

General Terms: Algorithms, Theory

Additional Key Words and Phrases: Large matchings, maximum matchings, 3-regular graphs, maxdeg-3 graphs, 3-connected planar graphs, bounded-degree block trees.

1. INTRODUCTION

Recall that a *matching* is a set of independent (that is, 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 [1891],

Work supported by grant WO 758/4-3 of the German Research Foundation (DFG). A preliminary version of this article has appeared as I. Rutter and A. Wolff. Computing Large Matchings Fast. In: *Proc. 19th ACM-SIAM Symposium on Discrete Algorithms (SODA'08)*, pages 183–192, 2008.

Authors' addresses: I. Rutter, Fakultät für Informatik, Universität Karlsruhe, KIT, Germany. E-mail: rutter@ira.uka.de, WWW: http://i11www.iti.uni-karlsruhe.de/people/rutter; A. Wolff, Lehrstuhl für Informatik I, Universität Würzburg, Germany. WWW: http://www1.informatik.uni-wuerzburg.de/en/staff/wolff_alexander

 http://dx.doi.org/10.1145/1868237.1868238

which states that every biconnected 3-regular graph has a *perfect* matching, that is, 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 [1986]. To date the asymptotically fastest (but rather complicated) algorithm for finding maximum matchings in general graphs runs in $O(\sqrt{n}m)$ time [Micali and Vazirani 1980], where n and m are the numbers of vertices and edges of the given graph, respectively. Only recently faster algorithms for dense graphs, for planar graphs, for graphs of bounded genus, and for general H-minor free graphs have been suggested. They are all based on fast matrix multiplication (which, as a tool, is not very practical) and run in $O(n^\omega)$ time for dense graphs [Mucha and Sankowski 2004], $O(n^{\omega/2})$ time for planar graphs [Mucha and Sankowski 2006] and for graphs of bounded genus [Yuster and Zwick 2007], and in $O(n^{3\omega/(\omega+3)}) \subset O(n^{1.326})$ time for H-minor free graphs [Yuster and Zwick 2007], where $\omega \leq 2.376$ is the exponent in the running time of the best-known matrix-multiplication algorithm [Coppersmith and Winograd 1987]. For practical purposes, however, often slower but less complicated algorithms are used: both LEDA [Algorithmic Solutions 2007] and the Boost Graph Library [Siek et al. 2007] provide maximum-matching algorithms with a running time of $O(nm\,\alpha(n,m))$ that are based on repeatedly finding augmenting paths [Gabow 1976; Tarjan 1983].

There has been a sequence of more and more general characterizations of graphs with perfect matchings [Petersen 1891; Hall 1935; Tutte 1947], 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, for example, bipartite k-regular graphs [Schrijver 1999; Cole et al. 2001], 3-regular biconnected graphs [Biedl et al. 2001], and subgraphs of regular grids [Thurston 1990; Hansen and Zheng 1993; Kenyon and Rémila 1996]. The last four algorithms all work in linear time for the corresponding subclasses of planar graphs. In planar bipartite graphs a perfect matching can be computed in $O(n \log^3 n)$ time if it exists [Miller and Naor 1995; Fakcharoenphol and Rao 2006]. There is also a fast algorithm for finding unique maximum matchings [Gabow et al. 2001]. It takes $O(m \log^4 n)$ time in general and $O(n \log n)$ time in planar graphs.

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, that is, in $o(\sqrt{n}m)$ time. This article is a first step into this direction. Our work was inspired by and addresses two open questions posed by Biedl et al. [2004], who gave 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 article 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 $\mathcal{T}_d$, that is, 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 upper bounds on ℓ_d for the corresponding graph class (type-1 bound). For example, Biedl et al. showed that every 3-regular graph has a matching

Table I. Our results. The distinction between type-1 and type-2 bounds follows the work of Biedl et al. [2004]. 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.

Graph Class	Bound on Matching Size		Runtime $O(\cdot)$	
	type-1	type-2	type-1	type-2
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	$n \log^4 n$
3-connected, planar, $n \geq 10$	$(n+4)/3$	$(2n+4-\mathbf{6}\ell_4)/4$	n	$n\,\alpha(n)$
3-regular planar	$(3n-6\ell_2)/6$		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, pl., bounded-deg 2-block tree	maximum		n	
3-connected, pl., bounded-deg 4-block tree	maximum		$n\,\alpha(n)$	

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. improved some of the earlier results of Nishizeki and Baybars [1979] 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.

Biedl et al. asked how quickly one can find matchings that are known to exist. Our first and main result answers this open question by "implementing" 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 I. 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 (that is, 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 [Ramaswami et al. 1998]. Biedl et al. [2004] showed that this bound is tight, but their original construction has no degree-2 vertices, that is, $n_2 = 0$. They gave 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. posed 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 (that is, is a path) contains a perfect matching. Biedl et al. [2001] have shown how to compute perfect matchings

in such a graph in $O(n \log^4 n)$ time. We extend the findings of Biedl et al. by showing how to compute a maximum matching in 3-regular graphs whose 2-block tree has *constant* maximum degree. Our algorithm also takes $O(n \log^4 n)$ time. If, however, the graph is additionally planar, our algorithm runs in optimal $O(n)$ time. 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. [2001] yields a query time of $O(\log^4 n)$. Thorup [2000] claimed to have a data structure with query time $O(\log^3 n \log\log n)$. 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 I.

Admittedly, our fast maximum matching algorithms are restricted to subclasses of 3-regular graphs and 3-connected planar graphs. Our results are, however, of more general interest since Biedl [2001] 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 (that is, 3-connected) planar graphs of maximum degree 9. One can interpret this in two ways. Either one sees Biedl's results as an indication that there are no near-linear-time algorithms for much wider subclasses of 3-regular graphs and 3-connected planar graphs, or one sees our algorithms as further evidence that there are in fact near-linear-time algorithms for maximum matching in planar graphs—or even in general graphs.

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. [2004].

Consider the following simple algorithm, which we call PickLeafEdges. It takes an arbitrary graph G as input and computes a set M of edges of G as follows. Initially M is empty. As long as G has a leaf (that is, 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 [Aronson et al. 1998].

THEOREM 2.1. *Let G be a graph, let $M = \text{PickLeafEdges}(G)$, 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 2.2. *Applying* PICKLEAFEDGES *to a tree yields a maximum matching in linear time.*

THEOREM 2.3. *Let T be a tree with n vertices and maximum degree 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 2.4. *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, for example, by breadth-first search. Then T also has maximum degree at most 3. By Corollary 2.2 we can find a maximum matching in T in linear time, and by Theorem 2.3 it has size at least $(n-1)/3$. □

This is one of the type-1 bounds of Biedl et al. [2004]; see Table I. The same technique can be used for maxdeg-k graphs, leading to a matching of size at least $(n-1)/k$. This, however, 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 2.5. *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 [Strothmann 1997]. Then Corollary 2.4 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. [2004] for 3-connected planar graphs with $n \geq 10$; see Table I. Hence we can reach their bound by finding at most two augmenting paths, which takes $O(n)$ time [Tarjan 1983]. □

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. [2004] (see Table I). 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. Finally we focus on planar 3-regular graphs.

3.1 3-regular graphs

Biedl et al. [2004] have shown that every 3-regular graph G 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 $\mathcal{T}_2$ of G. We show how to find such matchings

in $o(\sqrt{n}m)$ time. This has been known only for a special case: Biedl et al. [2001] have "implemented" Petersen's theorem [1891]. In $O(n \log^4 n)$ time they can find a perfect matching in a 3-regular graph with n vertices and $\ell_2 \leq 2$.

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*, that is, 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. [2004]. Their 2-block tree has a node for each biconnected component of G and a node for each *cut vertex* of G, that is, 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 has a node only 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 refer to vertices of the d-block tree $\mathcal{T}_d$ as *nodes* (as opposed to the *vertices* of the given graph).

In the following lemma we treat the special case $\ell_2 \in \{3, 4\}$. It will serve as the base of the induction in the proof of Theorem 3.2.

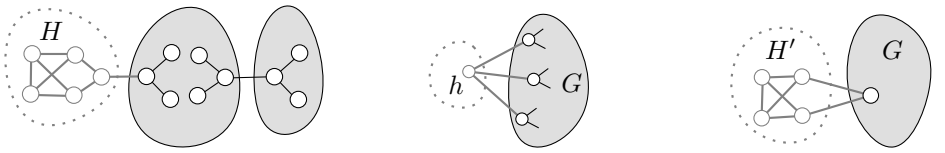
LEMMA 3.1. *Let G be a 3-regular graph whose 2-block tree has ℓ_2 leaves. If $\ell_2 \leq 4$ then G has a matching of size at least $(3n - 2\ell_2)/6$. The matching can be chosen such that every free vertex is incident to a bridge.*

PROOF. If $\ell_2 \in \{1, 2\}$, there exists a perfect matching by Petersen's theorem [1891]. If $\ell_2 = 3$ then $\mathcal{T}_2$ has exactly one node of degree 3. Let C be the corresponding biconnected component of G. We consider two subcases.

In the first subcase, C consists of a single vertex, which we call v. Every edge incident to v is a bridge. We remove these three bridges. Now we have four components, namely, the single vertex v and three *branches*, each containing one vertex of degree 2. We claim that in each of the branches there exists a matching that leaves only the degree-2 vertex free. To show the claim we argue as follows. We attach the helper graph H (see Figure 1a) to each degree-2 vertex. This makes the branches 3-regular. Thus Petersen's theorem [1891] yields a perfect matching in each of them. Finally we delete the helper graphs. Due to the structure of the helper graphs, it is clear that the degree-2 vertices become free, and the claim holds. One of these free vertices can be matched to v. This results in a total of two free vertices. Note that, as desired, every free vertex is incident to a bridge.

Now we treat the second subcase, that is, C contains more than one vertex. We remove two of the three bridges incident to C and call them b and b'. This yields three connected components B_1, B_2, and B_3, where B_1 contains C and thus has two degree-2 vertices, while B_2 and B_3 have only one degree-2 vertex each. Note that the 2-block trees of B_1, B_2, and B_3 are paths. We treat B_2 and B_3 as in the first subcase above. This yields two nearly perfect matchings M_2 and M_3 in B_2 and B_3, respectively, that leave the two degree-2 vertices free.

In B_1 we connect the two degree-2 vertices by a new edge e. This makes B_1 3-regular and yields a perfect matching M_1 in B_1 since the 2-block tree of B_1 is a path.



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

Fig. 1: Restoring 3-regularity.

If e is not in M_1 then $M_1 \cup M_2 \cup M_3$ is a matching in G that leaves two vertices free both of which are incident to a bridge. If e is in M_1 then $(M_1 \setminus \{e\}) \cup \{b, b'\} \cup M_2 \cup M_3$ is a perfect matching in G. This completes our treatment of the second subcase and thus settles the case $\ell_2 = 3$.

If $\ell_2 = 4$, we can argue similarly. □

THEOREM 3.2. *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. The cases $\ell_2 \leq 4$ are covered by Lemma 3.1. 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 upward 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 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, all leaves $\ell' \neq \ell$ remain valid in the sense that cutting off ℓ' at $v_{\ell'}$ reduces the number of leaves of the 2-block tree. 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 ver-

tices; see Figure 1b. This makes the main component 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. [2001] for computing matchings in the branches and for the base case. We only need to make a linear number of cuts because ℓ_2 is linear. 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 of G changes drastically when we link the new vertex h to the three degree-2 vertices of the main component. It remains to show how to maintain the leaf pointers and the dynamically changing 2-block tree of the main component. We call a branch *good* if its removal decreases the number of leaves in the main component.

LEMMA 3.3. *Given a 3-regular graph G, we can in $O(n\alpha(n))$ total time repeatedly determine three good 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.*

PROOF. We first compute the 2-block tree $\mathcal{T}_2$ of G. This takes linear time [Tarjan 1972]. Then we choose an arbitrary node w of $\mathcal{T}_2$ and direct all edges to w. We maintain a list of all leaves. For each leaf node, we have a pointer pointing to a bridge where it can be cut off. At every node of $\mathcal{T}_2$, we keep a list of all leaves that can be cut off at an incoming edge. This initialization takes linear time.

Next we repeatedly find three leaves that we can cut off. We do this as in the proof of Theorem 3.2. At the same time, we identify at most three leaves whose pointers become invalid and hence need an update. With our data structure, each update takes constant time. We denote the nodes where we cut off by a, b, and c as in the proof of Theorem 3.2.

We now have to contract into one big "super node" the nodes a, b, c, and all nodes that lie on the three unique paths between a, b, and c. We search these nodes starting from each of the nodes a, b, and c, following the directed edges. For the sake of efficiency, we search in parallel. Each of the searches marks the vertices it visits. The searches stop when the first node v has been visited by all three searches. The nodes that need to be contracted are v and all nodes that have not been visited before the corresponding search reached v. If we contract k nodes, the three searches have visited at most $3k + 2$ nodes in total. Using a union-find data structure with path compression and union by rank [Tarjan 1983], this step takes $O(n\alpha(n))$ time for the whole algorithm.

After we have computed the 2-block tree of the main component, we have to update the at most three leaves marked as needing an update. We use their pointers to get to our starting points in the graph and walk up from there until we reach a

node of degree at least 3. This establishes the invariant. Since we never walk back, each edge is visited at most once. Hence, this step takes linear time in total.

Note that the above description does not work correctly if the root of the tree has degree less than 3. A small root degree may be due to our initial choice of the root or the result of a contraction. If the root has degree 1, it is a leaf but does not have a bridge where it can be cut off. If the root has degree 2, there may be a leaf whose pointer points to a bridge incident to the root. This bridge, however, does not define a good branch. Both cases can be checked easily. We resolve this issue by using other leaves as long as there are at least six leaves. Once the number of leaves drops to five, we choose a new root with degree at least 3 and direct all edges toward it. This takes linear time and happens only once during the whole algorithm. □

Theorem 3.2 and Lemma 3.3 together yield the following theorem.

THEOREM 3.4. *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

We now 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 3.4 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 or, equivalently, 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 3.4 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. Adding and later removing the helper graphs neither changes ℓ_2 nor the number of free vertices and can be done in linear time. The following theorem summarizes our discussion.

THEOREM 3.5. *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.*

3.3 A sharp bound for maxdeg-3 graphs with many degree-2 nodes

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. [2004] in the affirmative.

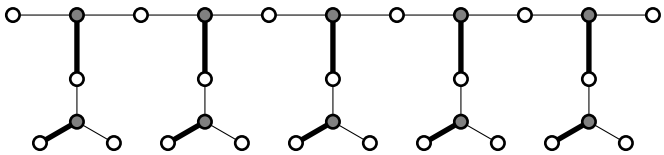


Fig. 2: A maxdeg-3 graph with $n = 31$ vertices, a linear number of degree-2 vertices, and a maximum matching of size $(3n - n_2 - 2\ell_2)/6$ (bold edges).

Let $k > 0$ be a positive integer, and let $n = 6k+1$. We construct an n-vertex graph as follows. We start with a path $v_1, \ldots, v_{2k+1}$. For $i = 1, \ldots, k$, we attach to v_{2i} one of the three leaves of a 4-vertex star; see Figure 2. The resulting graph G has $n_2 = 2k-1 = (n-4)/3$ degree-2 vertices. Note that since G is a tree, it is essentially its own 2-block tree. Hence ℓ_2 equals the number of leaves of G, that is, $(n+5)/3$. Thus the lower bound yields a matching of size at least $(3n-n_2-2\ell_2)/6 = (n-1)/3$. On the other hand, the central vertices of the k stars and the vertices $v_2, v_4, \ldots, v_{2k}$ form a vertex cover of G (the gray shaded vertices in Figure 2). Thus, any matching can have size at most $2k = (n-1)/3$. This shows that the bound $(3n - n_2 - 2\ell_2)/6$ is tight even if there is a linear number of degree-2 vertices.

3.4 Planar 3-regular graphs

We now turn to planar 3-regular graphs. The algorithm of Biedl et al. [2001], which finds perfect matchings in (non-planar) 3-regular graphs whose 2-block trees are paths, runs in $O(n \log^4 n)$ time. We show how to reduce its running time to $O(n)$ in the planar case and how to find large matchings in arbitrary planar 3-regular graphs in $O(n)$ time.

We first sketch the algorithm of Biedl et al. Their algorithm is based on decomposing the input graph G into bridgeless 3-regular components. It works as follows. Remove all bridges. Let G_i with $i = 1, \ldots, s$ be the resulting (bridgeless) connected components with n_i vertices of which at most two have degree 2. Then replace each degree-2 vertex and its incident edges by a single new edge. For each of the resulting 3-regular bridgeless components G_i' call a subroutine that computes in $O(n_i \log^4 n_i)$ time a perfect matching not including the at most two new edges in G_i'. Now the union of the perfect matchings in $G_1', \ldots, G_s'$ plus the bridges is the desired perfect matching in G. In the same work, Biedl et al. also considered the planar bridgeless case, which they can handle in linear time. Our aim is to generalize their algorithm from the planar bridgeless case to the planar version of Petersen's theorem while keeping its linear running time.

The subroutine of Biedl et al. makes use of the following simple observation. Given a 3-regular bridgeless graph G and two edges e_1 and e_2 of G, there is a perfect matching that does not contain e_1 and e_2. This can be seen as follows: subdivide e_1 and e_2 using extra vertices v_1 and v_2, respectively, and add the edge v_1v_2 to the graph. Let $G^\star$ be the resulting graph. Compute a perfect matching of $G^\star$. In case it does not contain v_1v_2, find an alternating cycle that contains v_1v_2. The symmetric difference of the matching and the cycle is the desired matching. Computing an alternating cycle can be done in linear time [Biedl et al. 2001]. Removing v_1v_2 from this matching yields the desired matching without e_1 and e_2 in the original graph G.

Adding the edge v_1v_2 may violate planarity, which forced Biedl et al. to use the $O(n \log^4 n)$-time subroutine for the non-planar case. The following lemma shows that this can be avoided.

LEMMA 3.6. *Let G be a planar 3-regular bridgeless graph and let e_1 and e_2 be any two edges of G. A perfect matching of G that neither contains e_1 nor e_2 can be computed in linear time.*

PROOF. To make use of the algorithm for the planar case we modify the above procedure slightly. We first compute *any* perfect matching M of G using the linear-time algorithm of Biedl et al. [2001] for the planar bridgeless case. Let $G^\star$ be defined as above. Observe that $M \setminus \{e_1, e_2\}$ is a matching in $G^\star$ that leaves only a constant number of vertices free. We now compute a matching $M^\star$ in $G^\star$ that contains the edge v_1v_2 by computing a constant number of augmenting paths with respect to $M \setminus \{e_1, e_2\}$. This takes linear time [Tarjan 1983]. Again $M^\star \setminus \{v_1v_2\}$ is the desired matching in G. □

Now we plug Lemma 3.6 as a subroutine into the algorithm of Biedl et al. that is sketched above. This yields the following.

COROLLARY 3.7. *Let G be a 3-regular planar graphs whose 2-block tree is a path. Then a perfect matching in G can be computed in linear time.*

Naturally we would like to use this improved algorithm as a subroutine of our algorithm for arbitrary 3-regular graphs to improve its running time to linear in case the graph is planar. Our scheme of cutting branches and adding new vertices, however, does not preserve planarity. So we cannot assume that the branches we cut are still planar. We leave open the question whether linear time suffices to compute a matching of size at least $(3n - 2\ell_2)/6$ in any planar 3-regular graph. If we insist on linear time, we can achieve the following weaker bound.

THEOREM 3.8. *Let G be a 3-regular planar graph whose 2-block tree has ℓ_2 leaves. Then a matching of size at least $(3n - 6\ell_2)/6$ can be computed in linear time.*

PROOF. We first compute the 2-block tree of G in linear time. Recall our algorithm for non-planar 3-regular graphs (Theorem 3.2). In each step it cuts off three branches from the main component. Here, in the planar case, we cut branches off one by one. We can easily find a good branch B by walking upward from a leaf in the 2-block tree until we find a node of degree at least 3. The edge between this node and the last node of the branch corresponds to a bridge $b = uv$ in G, where u is a vertex of the branch and v is a vertex of the main component. Now we remove the bridge b. The branch contains a nearly perfect matching that leaves only u free. By Corollary 3.7 we can compute such a matching M_B in time linear in the size of the branch B.

The other endpoint of the bridge, vertex v, has degree 2 in the main component C_{main}. Denote the neighbors of v by v_1 and v_2. We remove v and add the edge v_1v_2 to G. Note that this leaves C_{main} planar, but C_{main} may now be a multigraph with two copies of the edge v_1v_2. The 2-block tree of the main component has $\ell_2 - 1$ leaves. By induction C_{main} contains a matching that leaves at most $2\ell_2 - 2$ vertices free. We can compute such a matching M_{main} recursively; the algorithm

of Biedl et al. [2001] that is used as a subroutine in Corollary 3.7 explicitly allows multiple edges.

Consider the matching $M = (M_{\text{main}} \setminus \{v_1v_2\}) \cup \{b\} \cup M_B$ in G. This matching leaves at most $2\ell_2$ vertices free—the free vertices of M_{main} plus possibly v_1 and v_2 (if $v_1v_2 \in M_{\text{main}}$). Thus M covers at least $n - 2\ell_2$ vertices. In other words, M consists of at least $n/2 - \ell_2$ edges. □

We do not know how to generalize this result to maxdeg-3 graphs since our reduction for making a maxdeg-3 graph 3-regular does not preserve planarity.

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 is a triangle (for example, 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. [2004] 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. [2004] have shown that such a graph always has a perfect or a *nearly perfect* matching, that is, a matching that matches all vertices but one.

If a graph G is 3- but not 4-connected, there exists a separating vertex triplet $T = \{u, v, w\}$. For each component C of $G - T$, we consider the graph $C + T$ and add the *dummy* edges uv, vw, and uw if they did not already exist. 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 a node for every 4-connected component of G; two nodes are connected if the corresponding 4-connected components share a separating triplet (by planarity every separating triplet separates only two components). Note that the definition of Biedl et al. [2004] is more general. For 3-connected planar graphs, however, 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 [Chiba and Nishizeki 1989]. If G is not 4-connected, the basic idea is to find a matching in each 4-connected component separately and combine the resulting matchings 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 need not 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 is still a path, and hence G has a perfect matching. In particular, it follows that it is enough to leave a vertex in G_k free (the one matched to $v^\star$).

Now one could start by computing a Hamiltonian cycle in G_1 with the algorithm of Chiba and Nishizeki [1989]. It seems difficult, however, to extend either of the matchings induced by the cycle to a matching of G. Therefore we go a different way.

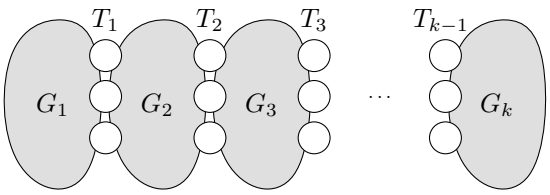


Fig. 3: Graph whose 4-block tree is a path.

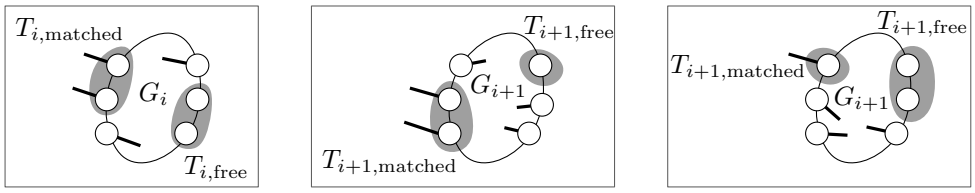


Fig. 4: Examples of matching configurations

Whether a given matching M_1 in G_1 can be extended to a perfect matching of G depends only on which vertices of the separating triplet T_1 are matched and which are free with respect to M_1. The next definition formalizes this observation.

For $1 < i < k$, a *matching configuration* of 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.

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

The first condition makes sure that vertices already matched in G_{i-1} are not declared free in G_i (and may hence not be matched again in G_{i+1}). The second condition makes sure that the configuration at hand can potentially be extended to a perfect matching in G.

Consider the directed acyclic graph whose nodes correspond to the feasible matching configurations of G and whose edges are defined as follows. Two nodes $u = (T_{i,\text{matched}}, T_{i,\text{free}})$ and $v = (T_{j,\text{matched}}, T_{j,\text{free}})$ of this graph are connected by an edge if and only if $j = i + 1$ and $T_i \setminus T_{i,\text{free}} = T_{j,\text{matched}}$. We call this graph the *configuration graph* of G. Note that the configuration graph contains the edge uv if and only if the matching given by u can be extended into the matching given by 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 nodes 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 configuration graph has $O(n)$ nodes since G has $O(n)$ 4-connected 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 configuration graph and vice versa. Such a 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 $G_i - (T_{i,\text{matched}} \cup T_{i,\text{free}})$.

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 starting at the source node of the configuration 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 [Chiba and Nishizeki 1989]. From M we remove all edges that do not belong to G and all edges incident to vertices we may not use (that is, 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 [Tarjan 1983]. 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 $G - v^{\star\star}$, that is, 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 [Kanevsky et al. 1992]. The 4-block tree has at most $(2n-4)/3$ leaves [Biedl et al. 2004]. We pick an arbitrary one and walk upward in the 4-block tree until we reach a component of degree at least 3. The last edge we have traversed corresponds to a separating triplet that we can use to cut off a leaf of the 4-block tree.

We split the graph at the separating triplet. This results in two components, the main component and the branch containing the leaf that we cut off. We add to each of these components the edges between the vertices of the separating triplet if they did not already exist. We again refer to these new edges as dummy edges. 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.*

PROOF. Let $B' = B \cup \{uv, vw, wu\}$. Note that B' is planar. Clearly, B' has a (nearly) perfect matching $M_{B'}$. We claim that we can choose $M_{B'}$ such that its free vertex, if any, belongs to T. If $|B'|$ is even, B' has a perfect matching. Otherwise we connect a dummy vertex to u, v, and w. Then the resulting graph B'' is still 3-connected, it is still planar, and its 4-block tree is still a path. Thus B'' contains a perfect matching. This perfect matching minus the edge that matches the dummy vertex is the claimed nearly perfect matching.

If u is free with respect to $M_{B'}$, let $u^\star = u$; otherwise let $u^\star$ be the vertex that is matched to u. Define $v^\star$ and $w^\star$ analogously.

Consider the matching $M' = (M_{\text{main}} \setminus \{uw, vw, wu\}) \cup (M_{B'} \setminus \{uu^\star, vv^\star, ww^\star\})$. If u, v, and w are matched in M', we can use $M = M'$. Otherwise this means that an edge of the separating triplet T (that is, a possible dummy edge) was used in M_{main}, say uv. In this case, u and v are free with respect to M'.

The vertices that are free with respect to M' but were not already free with respect to M_{main} are a subset of the set $F = \{u, v, u^\star, v^\star, w^\star\}$. We now show that we can either add one edge matching two of these vertices or $|F| \leq 3$.

If $u^\star$ is not in T, we simply take $M := M' \cup \{uu^\star\}$. The same works if $v^\star$ is not in T.

If uv is in $M_{B'}$ then $u^\star = v$ and $v^\star = u$ and hence $|F| = 3$.

The last case to check is that one of the vertices u or v, say u, is free with respect to $M_{B'}$ and the other one, that is, v, is matched to w. But in this case again, $v^\star = w$ and $w^\star = v$ and hence $|F| \leq 3$. □

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 the method described in Section 3.1. In fact here it is even easier since we can cut off the leaves one by one. Hence the complete 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. Due to the initial computation of the 4-block tree, the algorithm needs $O(n\alpha(n))$ time. The following theorem summarizes our discussion.

THEOREM 4.3. *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 4.4. *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 considerably simpler than the general 3-connected planar case (treated in Theorem 4.3). In the special case we 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 [Kant 1997].

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 $(n - (\ell_4 - 1))/2 = (2n + 2 - 2\ell_4)/4$. Computing an augmenting path yields the desired bound of $(2n + 4 - 2\ell_4)/4$ in linear time [Tarjan 1983].

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. Roughly, our idea is as follows. Since C is 4-connected and planar, we can compute *some* Hamiltonian circuit H in C in $O(|C|)$ time [Chiba and Nishizeki 1989]. In the following, we show how to transform H in linear time into a (nearly) perfect matching M_C of C such that (a) M_C respects the matching in the component C^- preceding C (note that C^- may be the main component), (b) M_C leaves at most one vertex free, and (c) if M_C leaves a vertex free and C is not the last component of B, then we can choose M_C such that the free vertex belongs to the component C^+ succeeding C. Given such a matching for each component of B, the linear running time of the whole algorithm can be seen as in the proof of Lemma 4.1.

It remains to show how to compute in linear time the (nearly) perfect matching M_C of C from the Hamiltonian circuit H. This requires some bookkeeping, but the idea is simple. The circuit H yields a (nearly) perfect matching M_H in C. We claim (and show below) that we can identify a constant-size subset R of C such that $C \setminus R$ has a perfect matching M_C that fulfills properties (a)–(c) listed above. This allows us to remove all edges incident to vertices in R from the matching M_H. The resulting matching M'_H is restricted to $C \setminus R$. By construction of R we can transform the matching M'_H into the desired perfect matching M_C by computing a constant number of augmenting paths in $C \setminus R$. Clearly, M_C can be computed in $O(|C|)$ time.

Let $T^- = \{u^-, v^-, w^-\}$ be the triangle separating C from C^-. If C is not the last component in B, let T^+ be the triangle separating C from C^+ 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 with respect to the matching that we have computed for C^-. Next, we specify the above-mentioned constant-size set R such that $C \setminus R$ contains a perfect matching fulfilling properties (a)–(c). The set R will be a subset of $\{u^-, v^-, w^-, u^+\}$. We use a result of Thomas and Yu [1994], which says that every graph $G^\star$ obtained by deleting at most two vertices from a 4-connected planar graph is Hamiltonian (Plummer's conjecture, Theorem 1.1). They also show that one can choose the Hamiltonian circuit in $G^\star$ such that it contains a prescribed edge on one of the facial cycles of $G^\star$ containing one of the deleted vertices (Theorem 3.3). We distinguish two cases depending on the number of free vertices in T^-.

Case I: One vertex of T^-, say w^-, is free.

Then $C \setminus \{u^-, v^-\}$ contains a Hamiltonian circuit. The circuit yields a

(nearly) perfect matching M_C in $C \setminus \{u^-, v^-\}$. If $|C|$ is odd, we can choose M_C such that u^+ is free. In this case, we let $R = \{u^-, v^-, u^+\}$. Otherwise, $|C|$ is even and all vertices are matched. In this case, we let $R = \{u^-, v^-\}$.

Case II: T^- does not contain any free vertices.

If $|C|$ is odd, $C \setminus \{u^-\}$ contains a Hamiltonian circuit with the edge v^-w^-. The circuit hence contains a perfect matching M_{perf} of $C \setminus \{u^-\}$. The matching M_{perf} can be chosen such that it contains the edge v^-w^-. Now $M_C = M_{\text{perf}} \setminus \{v^-w^-\}$ is the desired matching. We let $R = T^-$.

If $|C|$ is even, $C \setminus \{u^-, u^+\}$ contains a Hamiltonian circuit with the edge v^-w^-. As shown above, this circuit contains an appropriate matching. We let $R = T^- \cup \{u^+\}$.

In both cases and their subcases, it is obvious that $C \setminus R$ contains a perfect matching fulfilling properties (a)–(c). This completes our proof of the theorem. □

5. GRAPHS WITH BOUNDED-DEGREE BLOCK-TREES

We already know two algorithms for the special case that the corresponding block tree has maximum degree 2: (a) the algorithm of Biedl et al. [2001] computes perfect matchings in 3-regular graphs whose 2-block trees are paths, and (b) the algorithm from Section 4.1 computes (nearly) perfect matchings in 3-connected planar graphs whose 4-block trees are paths. In this section, we extend these results to graphs with block trees of constant maximum degree.

It is clear that, if we bound the number of leaves of the block tree by a constant, we can find maximum matchings in 3-regular 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 takes linear time [Tarjan 1983].

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 (that is, 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. Recall that we first decomposed the graph into its 4-connected components by iteratively splitting at separating triplets and adding dummy edges as necessary. Then each 4-connected component contains at most two (not necessarily disjoint) separating triplets. The clue was that whether a local matching in such a 4-connected component can be combined with a matching of a neighboring component depends exclusively on the question which vertices of the separating triplets are matched and which are free. We called this a matching configuration.

In the above case (where the 4-block tree is a path), we can afford to check all matching configurations since every component has at most two neighbor compo-

nents and thus only a constant number of matching configurations. Our arguments, however, also work for a constant number of neighbor components. We now give fast algorithms, first for 3-connected planar graphs and then for 3-regular graphs, in both cases assuming that the corresponding block tree has constant maximum degree.

5.1 3-connected planar graphs

When we treated the case of 3-connected planar graphs whose 4-block tree is a path (see Section 4.1), 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. Instead of just considering the feasibility of the matching configurations as in Section 4.1, we now keep track of the number of vertices that we must leave free to reach a given configuration. We use dynamic programming to join matching configurations. The running time of our 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 [Kanevsky et al. 1992], choose an arbitrary node of $\mathcal{T}_4$, and direct all edges toward it. We call a node u of $\mathcal{T}_4$ a *predecessor* of v if there is a directed path from u to v in $\mathcal{T}_4$. In this case, we also say that the 4-connected component of G corresponding to u is a *predecessor* of the component corresponding to v. We process the 4-connected components of G in topological order with respect to this predecessor relation. In this way, whenever we process a component, there is at most one neighbor component that has not been processed before.

We split G into its 4-connected components by keeping, with each component, copies of all vertices that belong to separating triplets incident to that component. In order to actually make the components 4-connected, we add dummy edges that turn each separating triplet into a separating triangle.

Let C be one of the resulting components 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 the question 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 each subset $F \subseteq T$, we compute how many vertices we have to leave free for a maximum matching in $G_C - F$ and store the resulting values in a counter. For this computation, we use the fact that we have already computed the counters of all predecessors of C. There is 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 $\mathcal{T}_4$), this takes asymptotically the same time as checking a single configuration.

Checking the cost of a single configuration takes time proportional to the size of the component C. This is due to the fact that C is 4-connected and planar, which means we can find a Hamiltonian cycle in C in linear time [Chiba and Nishizeki 1989] and

hence a (nearly) perfect matching. After computing such a matching, we remove all dummy edges (for example, edges that have only been 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 enlarge the matching to a maximum matching by computing a constant number of augmenting paths in linear time [Tarjan 1983]. The number of free vertices of the resulting configuration is the number of free vertices in the considered component plus the number of free vertices in the predecessor configurations.

If we store with each counter the corresponding configuration and the corresponding local matching, we can report a maximum matching in linear time once all components have been processed. (Instead of storing all local matchings we could also backtrack once we have processed the whole graph.) The next theorem summarizes this result. The asymptotic running time is dominated by the initial computation of the 4-block tree, which takes $O(n\alpha(n))$ time [Kanevsky et al. 1992].

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.*

5.2 3-regular graphs

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 [Tarjan 1972] and then work upward from the leaves in the same way as before. Here, however, we have only two counters for each (2-connected) 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 (that is, 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. It is not hard to see, however, 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.

To check the configurations of a given component we do the following. We first substitute every degree-2 vertex together with its incident edges by a new edge. This results in a bridgeless 3-regular graph. We then use the algorithm of Biedl et al. [2001] to compute a perfect matching in the modified component. By removing the new edges from the matching, we obtain a matching in the original component that leaves only a constant number of vertices free. This follows from the assumption that the degree of the 2-block tree is bounded. Starting from this matching, we can easily check the cost of every configuration by computing a constant number of augmenting paths, which takes linear time [Tarjan 1983].

The overall running time for the 3-regular case is dominated by the running time of the algorithm of Biedl et al. [2001], which runs in $O(p \log^4 p)$ time for a component of size p and in $O(p)$ time if the input graph is planar. Summing up these running times over all components yields the following theorem.

THEOREM 5.2. *Let G be a 3-regular graph with n vertices 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 at the end of the introduction, the above results concerning special graph classes are of general interest due to Biedl's linear-time reductions [2001] 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 (that is, 3-connected) planar graphs of maximum degree 9.

6. OPEN QUESTIONS

Our most burning question deals with 3-connected planar graphs. In such graphs, 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. [2004] in near-linear time? If not, what about the triangulated case, where we currently achieve $(2n+4-2\ell_4)/4$?

In planar 3-regular graphs, we currently have the choice of computing a matching of size at least $(3n-2\ell_2)/6$ in $O(n\log^4 n)$ time or a matching of size at least $(3n-6\ell_2)/6$ in linear time. The first algorithm does not exploit planarity; the second does. The ingredients of these algorithms are incompatible: the subroutine for treating branches in the linear-time algorithm depends on planarity, whereas the scheme of cutting branches and fixing 3-regularity in the slower but better algorithm does not preserve planarity. Can we still get the best of both worlds, that is, a matching of size at least $(3n-2\ell_2)/6$ in linear time?

Finally, it would be interesting to see whether there are fast algorithms that implement the bounds of Nishizeki and Baybars [1979] for planar graphs. The main question here is how to exploit minimum degrees.

ACKNOWLEDGMENTS

We thank Therese Biedl and the anonymous referees of this article for helpful comments. We also thank David Eppstein for discussions.

REFERENCES

Algorithmic Solutions. visited 07/04/2007. The LEDA user manual version 5.2. `www.algorithmic-solutions.info/leda_manual`.

Aronson, J., Frieze, A., and Pittel, B. G. 1998. Maximum matchings in sparse random graphs: Karp-Sipser revisited. *Rand. Struct. Algorithms 12,* 2, 111–177.

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

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

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

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

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

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

Fakcharoenphol, J. and Rao, S. 2006. Planar graphs, negative weight, shortest paths, and near linear time. *J. Comput. System Sci. 72*, 868–889.

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

Gabow, H. N., Kaplan, H., and Tarjan, R. E. 2001. Unique maximum matching algorithms. *J. Algorithms 40,* 2, 159–183.

Hall, P. 1935. On representatives of subsets. *J. London Math. Soc. 10*, 26–30.

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

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

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

Kant, G. 1997. A more compact visibility representation. *Intern. J. Comput. Geom. Appl. 7,* 3, 197–210.

Kenyon, C. and Rémila, E. 1996. Perfect matchings in the triangular lattice. *Discrete Math. 152,* 1–3, 191–210.

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

Micali, S. and Vazirani, V. V. 1980. 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)*. 17–27.

Miller, G. L. and Naor, J. 1995. Flow in planar graphs with multiple sources and sinks. *SIAM J. Comput. 24,* 5, 1002–1017.

Mucha, M. and Sankowski, P. 2004. Maximum matchings via Gaussian elimination. In *Proc. 45th Annu. IEEE Sympos. Foundat. Comput. Sci. (FOCS'04)*. 248–255.

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

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

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

Ramaswami, S., Ramos, P., and Toussaint, G. 1998. Converting triangulations to quadrangulations. *Comput. Geom. Theory Appl. 9*, 257–276.

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

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

Strothmann, W.-B. 1997. Bounded degree spanning trees. Ph.D. thesis, Heinz-Nixdorf-Institut, Universität Paderborn.

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

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

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

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

Thurston, W. P. 1990. Conway's tiling groups. *Amer. Math. Monthly 97,* 8, 757–773.

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

Yuster, R. and Zwick, U. 2007. Maximum matching in graphs with an exluded minor. In *Proc. 18th Annu. ACM-SIAM Sympos. Discrete Algorithms (SODA'07)*. 108–117.

Received November 2007; revised January 2009; accepted April 2009.