Efficient Algorithms for Petersen's Matching Theorem*

Therese C. Biedl† Prosenjit Bose‡
Erik D. Demaine† Anna Lubiw†

Abstract

Petersen's theorem is a classic result in matching theory from 1891, stating that every 3-regular bridgeless graph has a perfect matching. Our work explores efficient algorithms for finding perfect matchings in such graphs. Previously, the only relevant matching algorithms were for general graphs, and the fastest algorithm ran in $\mathcal{O}(n^{3/2})$ time for 3-regular graphs. We have developed an $\mathcal{O}(n \log^4 n)$-time algorithm for perfect matching in a 3-regular bridgeless graph. When the graph is also planar, we have as the main result of our paper an optimal $\mathcal{O}(n)$-time algorithm. We present three applications of this result: terrain guarding, adaptive mesh refinement, and quadrangulation.

Keywords: Petersen's theorem, perfect matching, 3-regular graphs

1 Introduction

In 1891, Petersen [36] published a pioneering paper in matching theory, and he is now considered one of the two principal founders of matching theory [28, p. xi]. In the paper, he proved what is now known as *Petersen's theorem*: "Ein primitiver *graph* vom dritten Grade muss wenigstens drei Blätter haben."[1] In modern terminology (see Section 2), the theorem implies that every 3-regular bridgeless graph has a perfect matching.

Petersen's original proof is very complicated. The American school of topologists recognized the importance of the theorem [3], and two members of that school, namely Brahana in 1917 [7] and Frink in 1926 [16], published simplified proofs. The interested reader can

*An extended abstract of this paper appeared at the 10th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'99).

†Department of Computer Science, University of Waterloo, Waterloo, Ontario N2L 3G1, Canada, email: {`biedl`, `eddemaine`, `alubiw`}`@uwaterloo.ca`. This work was done while the first author was at McGill University.

‡School of Computer Science, Carleton University, 1125 Colonel by Drive, Ottawa, Ontario K1S 5B6, Canada, email: `jit@scs.carleton.ca`.

[1]In English, "A primitive graph of third degree must have at least three leaves." A graph is *primitive* if its edges cannot be partitioned into smaller regular subgraphs. A graph has kth *degree* if it is k-regular. So a primitive 3-regular graph of third degree is a graph without perfect matching. A *leaf* is a 2-edge-connected component with exactly one incident bridge.

find extracts from Frink's paper [16] and Petersen's paper [36] in [3]. Frink's proof contained a slight flaw which was later corrected by König [25] in the first textbook on graph theory ever written [28, p. xvi]. Independently, Errera [13] published another proof of Petersen's theorem. Petersen's theorem is now usually known as a simple corollary of the theorem of Tutte [28, 44] characterizing the existence of perfect matchings in general graphs.

The goal of this paper is to find an efficient algorithm for Petersen's theorem, that is, to construct efficiently a perfect matching in a 3-regular bridgeless graph.

The first polynomial-time maximum matching algorithm for general graphs was discovered by Edmonds [11]. His method, using augmenting paths and "blossoms," is the basis for several faster algorithms; see [28]. The ultimate work in this direction to date is the algorithm of Micali and Vazirani [30, 37, 46], with a running time of $\mathcal{O}(m\sqrt{n})$, where n and m are the numbers of vertices and edges, respectively. Devising a faster matching algorithm for general graphs using augmenting paths and blossoms seems difficult, and no alternatives have been discovered.

We are interested in 3-regular graphs. Here $m = \frac{3}{2}n$, so the algorithm mentioned above runs in $\mathcal{O}(n^{3/2})$ time. We develop an algorithm to find a perfect matching in a 3-regular bridgeless graph with time complexity $\mathcal{O}(n \log^4 n)$, using recent results on dynamic maintenance of 2-edge-connectivity information [22]. In our main applications of Petersen's theorem, the graph is also planar. In this case we obtain an optimal $\mathcal{O}(n)$-time algorithm, which is self-contained in that it does not rely on results on dynamic maintenance of 2-edge connectivity information. Our algorithms are not based on the ideas of Edmonds' algorithm—we go back in time, past Edmonds' algorithm, past Tutte's theorem, back to the early proofs of Petersen's theorem. It is tantalizing to imagine a faster general matching algorithm based on alternatives to augmenting paths, but we have nothing to suggest along these lines.

The class of planar 3-regular bridgeless graphs, to which our linear-time algorithm applies, is quite rich. This class is exactly the class of duals of planar triangulations in which the outside face is a triangle. A perfect matching in a 3-regular planar graph gives us a pairing of triangles in the dual graph such that every triangle has a unique partner. Section 1.1 shows how such a pairing gives a good heuristic for placing guards to watch a triangulated terrain. Pairings are also important for adaptive refinement of triangular meshes in numerical simulations and for converting triangulations into quadrangulations, as described in Sections 1.2 and 1.3, respectively.

There have been a few matching algorithms for other specialized classes of graphs with running times faster than that of the best general matching algorithm. Schrijver [40] gives an $\mathcal{O}(km)$-time algorithm for finding a perfect matching in a k-regular bipartite graph, which is guaranteed to exist by a theorem of König from 1916. Schrijver's method also deviates from the standard approach of augmenting paths; instead, it repeatedly finds an arbitrary cycle, doubling every second edge in the cycle, and removing the remaining edges in the cycle. Gabow, Kaplan, and Tarjan [17] give an $\mathcal{O}(m \log^4 n)$-time algorithm to test whether a general graph has a unique perfect matching, using a theorem of Kotzig from 1959 that characterizes such graphs in terms of bridges. Like our algorithm for nonplanar graphs, their algorithm is based on the recent data structure for maintaining 2-edge-connectivity [22]. In addition, using Edmonds' blossom-shrinking approach, they give an $\mathcal{O}(m)$-time algorithm to test whether a given perfect matching is unique. Thurston [43] describes $\mathcal{O}(n)$-time algorithms, based on group-theoretic techniques developed by Conway, for testing for perfect matchings in finite subgraphs of the planar square grid or hexagonal grid in which every face (except the outside face) is an equilateral triangle or square. Hansen and Zheng [20] give another $\mathcal{O}(n)$-time algorithm for the hexagonal case. Kenyon and Rémila [24] give an $\mathcal{O}(n)$-time algorithm for the analogous problem on the planar triangular lattice.

The rest of this paper is organized as follows. This section continues with applications of algorithms for Petersen's theorem. Section 2 defines our terminology. In Section 3, we describe a simple $\mathcal{O}(n^2)$-time algorithm based on Frink's proof of Petersen's theorem [16], and show how to improve it to $\mathcal{O}(n \log^4 n)$ time. Section 4 presents an $\mathcal{O}(n)$-time algorithm for planar graphs. Some corollaries are given in Section 5, before we conclude in Section 6.

1.1 Application to Terrain Guarding

A classic problem in computational geometry is the problem of illuminating or guarding an object, using as few guards as possible. Our results on perfect matchings provide an improvement in the time complexity needed to find good guard placements in terrains, as we explain in this subsection.

Much of the research in the area of illumination has been carried out in two dimensions (see [34, 41, 45] for overviews). A step towards the corresponding problems in three dimensions is the study of polyhedral *terrains*, i.e., polyhedral surfaces that intersect every vertical line in at most a single point. The problem of guarding a polyhedral terrain was investigated by De Floriani et al. [15] who showed that the minimum number of guards needed to see the entire terrain can be found using a set covering algorithm. Cole and Sharir [10] subsequently showed that the problem is NP-hard. Goodchild and Lee [19] and Lee [27] presented some heuristics for placing guards at a subset of the vertices of a terrain.

Most of the work to date on guarding triangulated polyhedral terrains has focused on the underlying combinatorial problem of guarding a triangulated plane graph. A plane graph is *guarded* by a set of guards (placed on vertices or edges) if at least one guard is incident to every face of the graph. The relation between the geometric and underlying combinatorial problems is based on the observation that the visible region associated with a guard contains the union of all faces incident to that guard, and that this is all it contains when the underlying polyhedral terrain is convex. Therefore, upper bounds on the number of guards needed to guard a plane graph provide upper bounds on the number of guards needed to guard polyhedral terrains.

Bose, Shermer, Toussaint and Zhu [5] showed that $\lfloor n/2 \rfloor$ vertex guards are always sufficient and sometimes necessary to guard an n-vertex triangulated polyhedral terrain. With respect to edge guards (guards free to patrol an entire edge of the terrain), they established that at least $\lfloor (4n-4)/13 \rfloor$ edge guards are necessary in the worst case to guard the surface of an n-vertex triangulated polyhedral terrain. The complementary result that $\lfloor n/3 \rfloor$ edge guards are always sufficient was proved by Everett and Rivera-Campo [14]. Both sufficiency results apply to arbitrary triangulated polyhedral terrains and are based on the Four-Color Theorem, for which practical algorithms are not known to exist.

Recently, Bose, Kirkpatrick and Li [4] presented $\mathcal{O}(n^{3/2})$-time algorithms to guard an n-vertex triangulation with $\lfloor n/2 \rfloor$ vertex guards or $\lfloor n/3 \rfloor$ edge guards. The key behind these algorithms is to avoid the use of the Four-Color Theorem by relying instead on matchings.

Specifically, because the dual graph of the triangulation is 3-regular and bridgeless, it has a perfect matching. Removing the edges in the primal graph that correspond to the matched edges in the dual graph makes the primal graph bipartite, and thus yields a vertex 2-coloring of the triangulation. The authors show that using this K_3-free 2-coloring, one can obtain guard placements with $\lfloor n/2 \rfloor$ vertex guards or $\lfloor n/3 \rfloor$ edge guards.

The time complexity of these algorithms is dominated by the time to find the perfect matching in a planar 3-regular bridgeless graph. Therefore, with our results these algorithms can be implemented in optimal $\mathcal{O}(n)$ time.

The first use of matchings in terrain guarding was by Grünbaum and O'Rourke in 1983 [35], who proved that $\lfloor (2F-4)/3 \rfloor$ vertex guards suffice (and are sometimes necessary) to guard a *convex* polyhedron with F *faces.* They used a theorem by Nishizeki [33] that bounds the size of maximum matchings in a graph with vertex degrees ≥ 3. It would be interesting to know whether a matching achieving these bounds can be found in such a graph in linear time.

1.2 Application to Adaptive Mesh Refinement

Many numerical simulations involve the solution of continuous (e.g., partial differential) equations in some domain, which for our purposes is just a polygon. One typically discretizes this problem to a finite mesh (i.e., subdivision) of the domain. In general, the more detailed the mesh, the more accurate the simulation. However, the areas of the domain that require the most accuracy are often difficult to predict before simulation, and may vary over time. One can solve this problem by adaptively *refining* the mesh where the most error occurs.

One method for adaptive refinement of triangular meshes is *newest-vertex bisection* [31, 32]. In this method, each triangle has one vertex marked as the *newest vertex*, denoted in our figures by a small circle near the vertex. The *neighbor* of a triangle is the triangle incident to the opposite edge of the newest vertex. A triangle is *compatible* if either it has no neighbor or its newest vertex opposes its neighbor's newest vertex, that is, the neighbor relation is symmetric. To refine two compatible triangles, we bisect each triangle from its newest vertex to the midpoint of the opposite edge (see Figure 1), and assign the vertices at the midpoint to be the newest vertices of the new triangles.

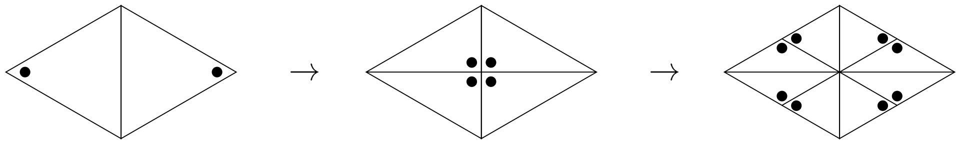

Figure 1: *Two iterations of uniform newest-vertex bisection applied to a pair of triangles with opposing newest vertices.*

For newest-vertex bisection to be applied effectively, we must first assign the newest vertices such that every triangle is compatible (a *compatible assignment*). Refinement from such a state has several important properties. For example, if triangles are refined uniformly as above, then compatibility is maintained. A simple generalization of the above refinement scheme allows refining the mesh by varying amounts in different regions. Because we start from a compatible assignment, we can bound the number of extra triangles introduced in the transition between a coarse region and a refined region. Throughout this process, the quality of triangles is preserved: newest-vertex bisection only introduces eight equivalence classes of congruent triangles.

It remains to show how to find a compatible assignment of newest vertices. Conceptually, we need a perfect pairing of triangles; then we can assign newest vertices to make each pair of triangles neighbors. However, any number of boundary triangles can "pair with the boundary." Consider a modified dual graph, in which a "boundary vertex" is created for every boundary edge in the primal (see Figure 2). Then any perfect matching in this graph results in a compatible assignment of newest vertices. Furthermore, this graph is always 3-regular and bridgeless [31], so by Petersen's theorem it has a perfect matching.

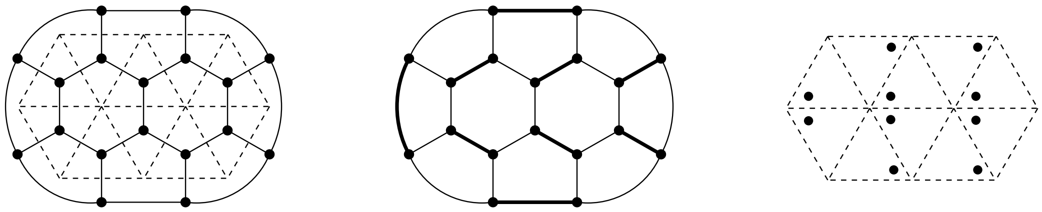

Figure 2: *The relationship between perfect matchings and pairings in triangulations. We show (from left to right) a triangulation with a modified dual graph, a perfect matching in this dual graph, and the resulting compatible assignment of newest vertices in the triangulation.*

The time complexity of assigning the newest vertices is thus dominated by the time to find the perfect matching in a planar 3-regular bridgeless graph. With our results this can be implemented in optimal linear time.

1.3 Application to Quadrangulation

In a variety of numerical simulations, such as those involving the flow of incompressible fluid, triangulations can be inappropriate meshes. This is because triangles are rigid and can *lock* (become unable to move), effectively halting the simulation. In such cases, it is preferable to use a quadrangulation, that is, a subdivision of the domain into quadrangles (quadrilaterals). There are many other applications in which quadrangles have advantages over triangles, including scattered bivariate data interpolation [26] and elasticity analysis [1].

In contrast with triangulations, which have been studied for several decades [2], relatively little is known about quadrangulations. For this reason, several people have considered the problem of converting triangulations to quadrangulations, most recently in a computational-geometry setting [39].

We obtain an immediate result in this area using the modified dual graph described in the previous section. By deleting the duals of matched edges, except those on the boundary or the outside face, we obtain a *weak quadrangulation*, that is, a mesh that has quadrangles except for some triangles along the boundary. Weak quadrangulations are usually acceptable for numerical simulations. By our results, this weak quadrangulation can be found in optimal linear time.

2 Terminology

This section defines the graph-theory terminology used in this paper.

Maximum matching is a classic problem in graph theory with many practical applications [28]. Briefly, let $G = (V, E)$ denote a graph with vertex set V and edge multiset E, where each edge $e \in E$ is a set (v, w) of two vertices $v, w \in V$. We use n and m to denote $|V|$ and $|E|$, respectively. Unless we specify that G is simple, we allow it to have multi-edges and loops. Edges with multiplicity one, two, and three will be called *simple*, *double*, and *triple edges*, respectively. Two edges are called *incident* if they share an endpoint.

A *matching* in G is a subset M of edges such that for every vertex v, at most one edge e in M *covers* v, that is, satisfies $v \in e \in M$. An edge e is called *matched* if it is contained in the matching, and *unmatched* otherwise. A *maximum matching* is a matching with largest possible cardinality. A *perfect matching* (or 1-*factor*) is a matching such that every vertex is covered. Given a matching of a graph, an *alternating cycle* is a cycle (a closed path that does not repeat any vertices) every second edge of which is matched. Reversal of an alternating cycle (that is, switching matched and unmatched edges), yields another matching of the same cardinality. An *augmenting path* is a path (without repeated vertices) between two uncovered vertices on which every second edge is matched.

A graph is k-*regular* if every vertex has degree k, that is, k incident edges. Such a graph satisfies $m = \frac{k}{2}n$. An *edge cut* in a graph G is a subset C of edges such that $G - C$ has more connected components than G, that is, there are vertices v and w that are connected by a path in G but not in $G - C$. G is k-*edge connected* if all edge cuts in G have cardinality at least k. A *bridge* in a graph G (also called a *cut edge* or *isthmus*) is an edge cut of cardinality one. We call a graph *bridgeless* if it has no bridges, that is, it is 2-edge connected.

A graph is called *planar* if it can be drawn in the plane without edge crossings. A specific *planar embedding* is given by the clockwise circular order of edges around each vertex. A planar drawing subdivides the plane into regions called *faces*; these faces are determined by the planar embedding alone. Whenever we speak of a planar graph, we assume that some (arbitrary) planar embedding has been fixed beforehand; this can be computed in linear time [6, 23, 29]. The *dual graph* G^* of a planar graph G is obtained by creating a vertex in G^* for every face in G, and adding an edge (F_1, F_2) in G^* for every edge e in G that is incident to the two faces F_1 and F_2; (F_1, F_2) is called the *dual edge* of e. The planar embedding of the dual graph is determined by the embedding of the primal graph.

3 Nonplanar Case

3.1 Frink's Proof of Petersen's Theorem

In this section, we overview Frink's proof of Petersen's theorem, which is the basis for our algorithms. Frink's original proof [16] is available in [3]. It contains a slight flaw which was corrected in König's detailed version of the proof [25]. As we will see, the proof is constructive, leading to a simple $\mathcal{O}(n^2)$-time algorithm.

The proof is by induction on n. Let G be a bridgeless 3-regular graph. Note that because the reductions to come do not preserve simplicity, we cannot assume that G is simple.[2] However, G cannot have loops, because any loop in a 3-regular graph is incident to a bridge. In the base case, the graph only has triple edges. Pick one of the three edges in each connected component to obtain a perfect matching.

Assuming we are not in the base case, the graph has an edge that is not a triple edge. By 3-regularity, either this edge or one incident to it must be a simple edge $e = (v, w)$. Call e the *reduction edge*. Let a, b, and w be the three neighbors of v, and let c, d, and v be the three neighbors of w (see Figure 3). By our assumption that e is simple, $a, b \neq w$ and $c, d \neq v$, but we may have some of a, b, c, and d being equal.

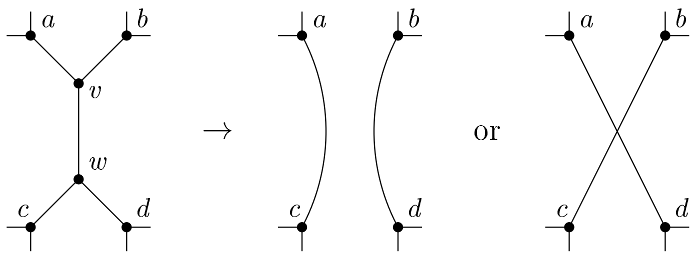

Figure 3: *The straight and crossing reductions in Frink's proof.*

Reduce the graph by removing the vertices v and w, and interconnecting a, b, c, and d with two new edges called *reduced edges*. There are two possible reductions (see Figure 3): either connect a to c and b to d, called the *straight reduction*; or connect a to d and b to c, called the *crossing reduction*.[3] Clearly both reductions lead to 3-regular graphs.

Lemma 3.1 [25, p. 182] *One of the reductions results in a bridgeless graph with the same number of connected components as the original graph.*

As a result of this lemma, one reduced graph satisfies the conditions of Petersen's theorem. (In fact, we did not need the number of connected components to be preserved, and we will exploit this freedom later.) By induction, find a perfect matching in this reduced graph. To complete the proof, this matching must be extended to a perfect matching in the original graph. This operation is purely local if at most one of the two reduced edges is in the matching of the reduced graph (see Figure 4 for the case of the straight reduction).

If both reduced edges are matched, then reverse an alternating cycle containing one of the reduced edges. If the other reduced edge was in the alternating cycle as well, then now neither of the reduced edges is matched; otherwise, exactly one of the reduced edges is matched. In either case, extend the matching as in Figure 4. This argument relies on the following lemma.

Lemma 3.2 [16] [25, p. 187] *Given any perfectly matched 3-regular bridgeless graph and some edge e in the graph, there exists an alternating cycle that includes e.*

[2] Indeed, this was Frink's error, to assume that the graph was simple.

[3] In the nonplanar case, where the drawing of the graph is arbitrary, these names effectively mean "one reduction" and "the other." We use this terminology to be consistent with the planar case, where these names gain significance, assuming that the planar embedding is as in Figure 3.

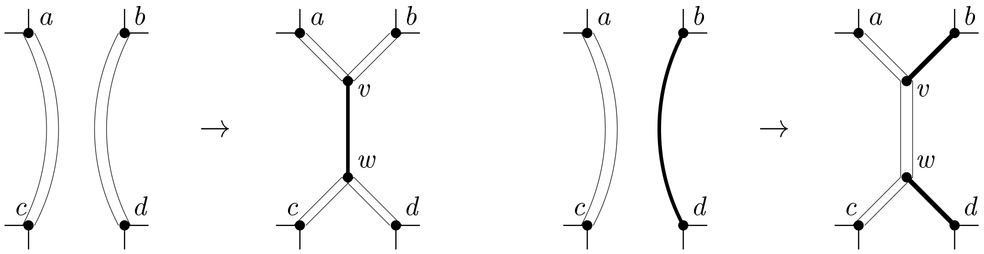

Figure 4: *Extending to a perfect matching in Frink's proof. Matched edges are drawn thick, and unmatched edges are drawn hollow.*

Converting induction into recursion, we have a simple algorithm for Petersen's theorem. Each step of the recursion consists of checking for the base case, finding an edge of multiplicity one, picking and applying an appropriate reduction, and possibly finding and reversing an alternating cycle. All of these operations are easy to perform in $\mathcal{O}(n)$ time, except for finding an alternating cycle which is somewhat more difficult.

Lemma 3.3 *Given any perfectly matched 3-regular bridgeless graph and some edge e in the graph, an alternating cycle including e can be found in $\mathcal{O}(n)$ time.*

Proof: We show how finding an alternating cycle reduces to finding an augmenting path, which can be done in $\mathcal{O}(m)$ time for general graphs with m edges (see e.g. [42]) using the Gabow-Tarjan set-union algorithm [18].

We distinguish two cases. If $e = (v, w)$ is matched, then delete e and find an augmenting path in the resulting graph. By Lemma 3.2, there is an alternating cycle through e in the original graph, and thus there exists an augmenting path from v to w in the modified graph. Furthermore, such an augmenting path must be found, because all other vertices have an incident matched edge. We can complete this augmenting path to an alternating cycle through e.

If e is not matched, then let e_1 and e_2 be the other two edges incident to v. One of them, say e_1, must be matched because we have a perfect matching. No alternating cycle can contain both unmatched edges e and e_2, so the alternating cycle containing e is also an alternating cycle in $G - e_2$. We can find this alternating cycle by finding an augmenting path in $G - \{e_1, e_2\}$. This path must connect v and the other endpoint of e_1 (all other vertices have an incident matched edge). Because e is the only edge incident to v, this augmenting path must contain e, and thus e_1 completes it to an alternating cycle containing e. □

As a consequence, each step in the recursion takes $\mathcal{O}(n)$ time, and hence the matching algorithm takes $\mathcal{O}(n^2)$ time.

3.2 Improving the Time Complexity

The two main bottlenecks in reducing the time complexity are (a) determining which of the two reductions results in a bridgeless graph with the same number of connected components, and (b) finding an alternating cycle. In this section, we show how to avoid the necessity of finding alternating cycles, hence removing bottleneck (b), and how to solve bottleneck (a) in $\mathcal{O}(\log^4 n)$ time, reducing the overall complexity to $\mathcal{O}(n \log^4 n)$ time.

We can remove the use of alternating cycles in Frink's proof by entirely avoiding the case in which both reduced edges are matched. To do this, we strengthen Petersen's theorem to the following: every 3-regular bridgeless graph has a perfect matching not using a particular edge e^{NM}, called the *nonmatching edge*. This result follows immediately from Lemma 3.2, but makes the induction easier and the resulting algorithm faster.

Note that in Frink's proof the choice of the reduction edge e was arbitrary; now choose one of the edges incident to e^{NM}. Assume for now that there exists a simple edge e that is incident to e^{NM}; with respect to the labeling of Figure 3, $e^{\mathrm{NM}} = (a, v)$ (say) and $e = (v, w)$. Reduce as before, and define the new nonmatching edge to be the reduced edge that is incident to one of the ends of e^{NM}, thus either (a, c) or (a, d) depending on the reduction. In the resulting graph, compute a perfect matching that does not use the new nonmatching edge. As a consequence of this restriction, at most one of the reduced edges is matched, so the matching can be extended as in Figure 4, without the need for alternating cycles. Note that neither extension causes $e^{\mathrm{NM}} = (a, v)$ to be matched, as required.

Unfortunately, all edges incident to e^{NM} may be double edges, but the reduction edge e is not allowed to be a double edge. In this case, perform a different reduction: pick some double edge incident to e^{NM}, and reduce this double edge and the two other incident edges (one of which is e^{NM}) down to a single nonmatching edge (see Figure 5). Recursively compute a perfect matching in the resulting 3-regular bridgeless graph, and extend it to a perfect matching of G by adding one side of the double edge to the matching.

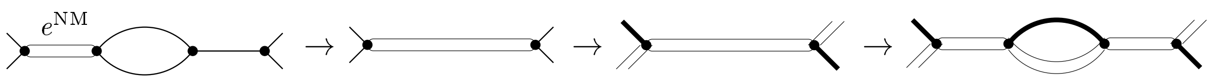

Figure 5: *Reduction and extension for a double edge incident to the nonmatching edge e^{NM}. The nonmatching edge is drawn hollow.*

In this algorithm, each step in the recursion takes constant time except for determining the correct reduction (if e is simple). This amounts to testing whether one of the reductions results in a bridgeless graph with the same number of connected components. If not, the other reduction must have this desired property by Lemma 3.1. By Menger's theorem, a graph is bridgeless precisely if there are two edge-disjoint paths between any pair of vertices. A reduction can only potentially destroy this property between pairs from a, b, c, d that are not connected by a reduced edge [25, p. 182]. As a consequence, testing whether the resulting graph is bridgeless reduces to testing the existence of two edge-disjoint paths between a constant number of pairs of vertices.

Thus we want to maintain a dynamic graph subject to insertion and deletion of edges, and support queries that ask whether a pair of vertices are connected by two edge-disjoint paths. This *2-edge-connectivity problem* has a fairly long history. It was a long-standing open problem whether deterministic polylogarithmic update time was possible. Previously, the best worst-case result was $\mathcal{O}(\sqrt{n})$ update time [12], and the best randomized result was $\mathcal{O}(\log^5 n)$ expected update time [21]. Recently, Holm, de Lichtenberg, and Thorup [22] developed a data structure with $\mathcal{O}(\log^4 n)$ worst-case update and query time.

Therefore, we can find perfect matchings in 3-regular bridgeless graphs in $\mathcal{O}(n \log^4 n)$ time, which is asymptotically smaller than the previous best algorithm running in $\mathcal{O}(n^{3/2})$ time [30].

Theorem 1 *Let G be a 3-regular bridgeless graph, and let e^{NM} be an edge of G. Then there exists a perfect matching of G that does not contain e^{NM}, and it can be found in $\mathcal{O}(n \log^4 n)$ time.*

We note that the same 2-edge-connectivity data structure has been used to test whether a general graph has a *unique* perfect matching in $\mathcal{O}(n \log^4 n)$ time [17].

4 Planar Case

Because our interest in perfect matchings arose from applications with planar graphs, we would like to improve the time complexity even further in this case. This section describes an algorithm for finding perfect matchings in planar 3-regular bridgeless graphs in $\mathcal{O}(n)$ time, which is optimal.

For planar graphs we avoid using a dynamic 2-edge-connectivity data structure and instead read the required information from the dual graph, using the property that a bridge in the primal graph is a loop in the dual graph. So fix a planar embedding (if not given already), compute the dual graph, update the dual graph throughout the changes to the primal graph, and test for loops as needed. The main impediment to this plan is that the crossing reduction used in the previous section does not preserve the planar embedding, or even planarity (see Figure 6).

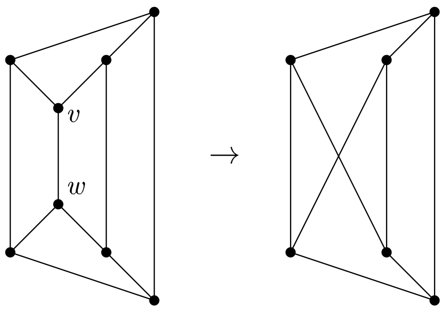

Figure 6: *Applying the crossing reduction to a planar 3-regular bridgeless graph can introduce a $K_{3,3}$.*

To remedy this we exploit the remaining freedom in the choice of the reduction edge. In particular, we choose among the potential reduction edges adjacent to the nonmatching edge e^{NM}, searching for a straight reduction that maintains bridgelessness. In this way we avoid the crossing reduction entirely.

We begin by describing the main reduction. The following section formulates and implements the necessary primal and dual graph operations, and the last section gives the details of the algorithm.

4.1 The Main Reduction

Assume that e^{NM} and all its incident edges are simple; all other cases will be treated in Section 4.3. Let $e^{\mathrm{NM}} = (v, w)$, and let x_l and x_r be the two other vertices adjacent to w in counterclockwise order after v (see Figure 7). By assumption these four vertices are distinct. Define F_l to be the face incident to (v, w) and (w, x_l), F_r to be the face incident to (v, w) and (w, x_r), and H to be the face incident to (w, x_l) and (w, x_r). Let G_l denote the face other than F_l and H that is incident to x_l, and G_r denote the face other than F_r and H that is incident to x_r.

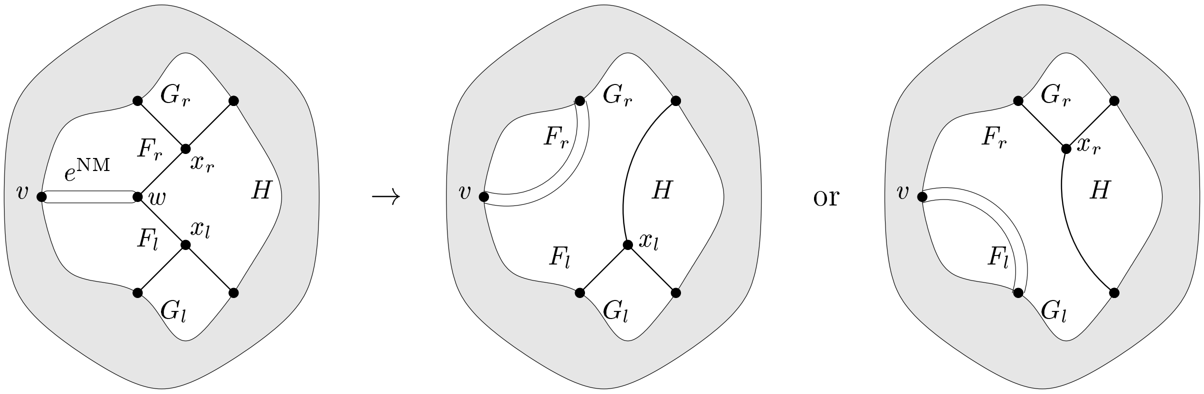

Figure 7: *Definitions of vertex and face names, and the resulting faces if we reduce edges* (w, x_r) *and* (w, x_l)*, respectively.*

Note that any two faces with a common edge are distinct because the graph is bridgeless, but we may have $F_r = G_l$, $F_l = G_r$, or $G_l = G_r$. Here we discuss the case $G_l \neq G_r$; the other case is again left to Section 4.3. We want to apply the straight reduction to either (w, x_l) or (w, x_r) (see Figure 7), and have to show that one of them leads to a bridgeless graph.

Lemma 4.1 *If* $G_l \neq G_r$ *then either applying the straight reduction to* (w, x_l) *or applying it to* (w, x_r) *results in a bridgeless graph. The correct reduction can be found by testing whether* F_r *is adjacent to* G_l or *by testing whether* F_l *is adjacent to* G_r*.*

Proof: Suppose we are about to apply the reduction to the edge (w, x_l). In the dual graph, this corresponds to identifying the faces F_r and G_l, after deleting the edges (F_l, F_r), (F_r, H), and (H, F_l) (see Figure 8).

Assume the primal graph will contain a bridge after the reduction; this corresponds to a loop appearing in the dual graph. The incident vertex of this loop must be the combined vertex formed by contracting F_r and G_l, because the primal graph is bridgeless before the reduction, and no edges will be added to the dual graph during the reduction. Hence, if the primal graph will have a bridge after the reduction, then F_r and G_l must be adjacent before the reduction.

Now if $G_l \neq G_r$, we claim that at most one of (F_r, G_l) and (F_l, G_r) is an edge in the dual graph before the reduction. Assume to the contrary that they were both edges in the dual graph. Because the primal graph is bridgeless, the dual graph has no loops. In particular, the edges (F_r, G_l) and (F_l, G_r) in the dual graph must have distinct endpoints, i.e., $F_r \neq G_l$

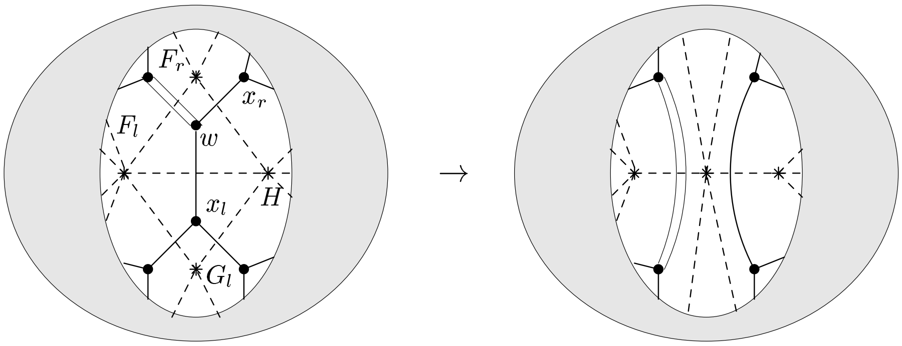

Figure 8: *The straight reduction of (w, x_l), and corresponding changes to the dual graph. Stars and dashed lines denote vertices and edges in the dual graph, respectively.*

and $F_l \neq G_r$. In the dual graph a triangle is formed by H, F_r, F_l. Add a new dual vertex T to the middle of this triangle, and join T to H, F_r, and F_l (see Figure 9). Clearly this maintains planarity. However, there is a $K_{3,3}$ formed by the dual vertices T, G_r, G_l and F_r, F_l, H, and these are six distinct vertices because $G_r \neq G_l$. This is a contradiction, because no planar graph can contain a $K_{3,3}$.

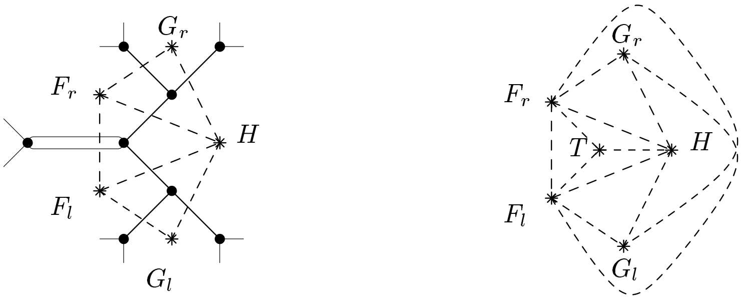

Figure 9: *If $G_l \neq G_r$, then not both $(F_l, G_r) \in E(G^*)$ and $(F_r, G_l) \in E(G^*)$; otherwise we have a planar $K_{3,3}$.*

Thus, if F_r and G_l are adjacent, then F_l and G_r cannot be adjacent, and reducing edge (w, x_r) yields a bridgeless graph; otherwise, reducing edge (w, x_l) yields a bridgeless graph. Alternatively, the correct reduction can be found by testing whether F_l is adjacent to G_r. □

Note that this lemma allows the graph to become disconnected (in the case $F_r = G_l$), unlike Frink's lemma (Lemma 3.1).

4.2 Data Structure

We now formulate and implement the dynamic graph operations needed to perform reductions. Refer to Figure 8 for the main reduction.

Consider first the update operations. In the primal graph we want to delete isolated vertices, and to delete and insert edges. These edge operations correspond in the dual graph to contracting two vertices and to splitting a vertex, respectively. We avoid some of these operations by not keeping the primal and dual graphs in lock-step, but rather performing a small sequence of operations in the primal graph and a small sequence of operations in the dual graph, obtaining once more a pair of dual planar graphs. It is the responsibility of the main algorithm, not the update operations in this section, to maintain the correspondence between the primal and dual graphs. For update operations in the dual graph, we will make do with the ability to delete edges and contract pairs of vertices. We have already seen that these suffice in the main reduction.

As for queries, we want to list the vertices and edges incident to a vertex in the primal graph, and we want to be able to find the dual of an edge in either graph; note that this allows us to find the incident faces of edges and vertices as well. Finally, to decide on the correct reduction, we want to answer *adjacency queries* of the form, “Are F_1 and F_2 adjacent vertices in the dual graph?”

To solve this dynamic-graph problem we use incidence lists to store edges in cyclic order around the vertices, both in the primal and the dual graph. Every edge refers to both its entries in the incidence lists, and to the dual edge in the other graph. Because the maximum degree of the primal graph is three, this allows us to perform the operations in the primal graph and to find the dual of an edge in constant time.

However, the operations in the dual graph in general take more than constant time, because the maximum degree is not bounded. To remedy this problem, we will narrow our sights and limit the dual operations.

For our main reduction, observe that we only need to contract a vertex to F_l or F_r, and we only need to query whether a vertex is adjacent to F_l or F_r; in fact, queries to just one of these suffice, as shown in Lemma 4.1. Call the nonmatching edge e^{NM} the *special edge*. One of the two faces incident to it will be maintained as a *special face*; in the dual graph we call it the *special vertex*. The special vertex will be one of F_l or F_r, not always the same one; we can find out which one by storing a flag. The other of F_l, F_r will be called the *sibling* of the special vertex.

The main reduction (see Figure 8) disconnects the primal graph if $F_r = G_l$, so we must deal with a nonmatching edge in each connected component. Thus the special edge will change as the algorithm progresses. However, the special vertex will retain its identity; only its sibling may change, and this can happen only after the old sibling is contracted into the special vertex.

We verify in the next section that the following dual operations are sufficient to determine which reduction should be done, and to perform the reduction.

- Query whether a vertex v in the dual graph is incident to the special vertex.
- Delete an edge e in the dual graph.
- Contract a vertex v of the dual graph with the special vertex or its sibling. The contracted vertices share a face in the dual graph, and this face is known at the time of the contraction.

To implement the queries, we store with every dual vertex v an integer v_s that specifies the number of edges between v and the special vertex. To answer the adjacency query for v, test whether $v_s > 0$, which takes constant time.

To delete an edge $e = (v, w)$ in the dual graph, decrement v_s if w is special, decrement w_s if v is special, and delete e in the incidence lists of v and w. This takes constant time.

To contract v into x, where x is the special vertex or its sibling, merge the incidence lists of v and x at the location indicated by the face containing both v and x; this takes constant time. For each neighbor w of v, update the endpoints of edge (v, w) to (x, w), and in addition, if x is special, increment w_s. This takes more than constant time for a single contraction, and we account for the work by charging it to the edges that are updated. For an edge e in the dual graph, let e_s denote the number of endpoints of e (0, 1, or 2) that are the special vertex or its sibling. Note that every edge updated during a contraction increases e_s for some edge e, because we only contract to the special vertex or its sibling. The only way e_s could decrease would be if one endpoint of e were deleted, or if the two endpoints of e were contracted. But we only delete isolated vertices, and we never contract adjacent vertices, because this would create a loop in the dual graph and thus a bridge in the primal graph. Hence e_s never decreases, which implies that endpoints of e are updated during contractions at most twice during the lifetime of e. Therefore the total time spent during contractions is $\mathcal{O}(n)$.

Finally, in order to deal with many connected components in the primal graph, we keep a stack consisting of one nonmatching edge from each connected component. We pop the stack to obtain the current special edge, implicitly determining the current connected component. In order to preserve the identity of the special vertex, we maintain the invariant that if e is the top edge on the stack (i.e., e belongs to the next connected component to be handled) then in the planar embedding there exists a face incident to e and at least one edge of the current component. This means that when the current component is deleted, the special face becomes incident to e, the new special edge. See the next section for details.

4.3 Algorithm

With our dual graph operations in hand, we now explain the details of the reduction to find a perfect matching. We distinguish cases by the multiplicity of the nonmatching edge e^{NM}. The cases of multiplicity more than one are easy to deal with; most of this section is concerned with the case of multiplicity one. Recall that the subcase in which $G_l \neq G_r$ was already overviewed in Section 4.1.

In each case, there are three main steps. First, we reduce the graph to a smaller graph by applying the operations supported by the data structure described in the previous section. Second, we update the stack while maintaining the stack invariant. At this point, we recursively find a perfect matching in the reduced graph. Finally, we must extend this matching to a perfect matching in the original graph.

4.3.1 e^{NM} has Multiplicity Three

If the special edge e^{NM} has multiplicity three, then its connected component C is a triple edge (see Figure 10). We add one of the other two edges to the matching, and are done with C. If the stack is empty, we are done. Otherwise, delete the three edges of C and their dual edges. Contract the dual vertices of the three faces in C into one dual vertex; the special vertex has then absorbed its sibling. Pop the stack to obtain the new special edge. By the stack invariant, the special face is incident to this new special edge. The other incident face of this edge is the new sibling.

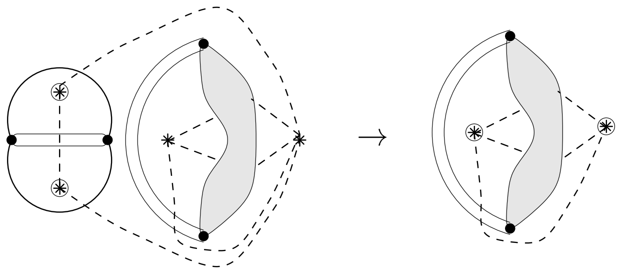

Figure 10: *The case where we transit from one connected component to the next. The special dual vertex and its sibling are circled.*

4.3.2 e^{NM} has Multiplicity Two

If the multiplicity of e^{NM} is two, reduce the edges incident to e^{NM} down to a single non-matching edge (see Figure 11). The dual graph can be updated by deleting three edges and contracting two vertices. Recursively compute a perfect matching in the resulting graph, and extend it by adding the other side of the double edge e^{NM}.

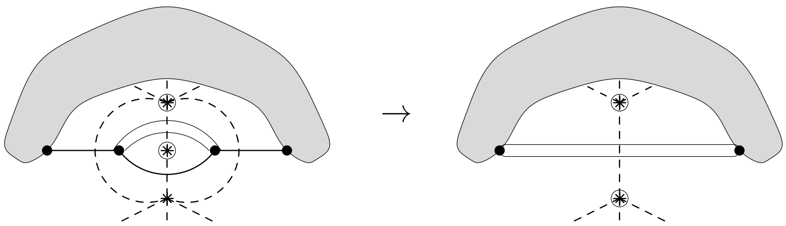

Figure 11: *The case where the nonmatching edge is a double edge.*

4.3.3 e^{NM} has Multiplicity One

Assume henceforth that e^{NM} has multiplicity one. If there is a double edge incident to e^{NM}, reduce the edges incident to the double edge down to a single nonmatching edge (see Figure 12). The dual graph can be updated by deleting three edges and contracting two vertices. Recursively compute a perfect matching in the resulting graph, and extend it by adding one of the sides of the double edge.

So assume from now on that e^{NM} and all incident edges are single edges. Let $e^{\mathrm{NM}} = (v, w)$, and define $x_l, x_r, F_l, F_r, G_l, G_r$, and H as in Section 4.1 (see Figure 7). Let us first consider the case where $G_l = G_r$. We have two subcases, depending on whether x_l and x_r are adjacent.

If $G_l = G_r$ and x_l and x_r are adjacent, i.e., if H is a triangle, then reduce H to a single vertex (see Figure 13). In the dual graph, this corresponds to deleting the three incident edges of H, and then deleting H. Recursively compute a perfect matching of the resulting graph, and extend it by adding the edge of the triangle $\{w, x_l, x_r\}$ that is not incident to a

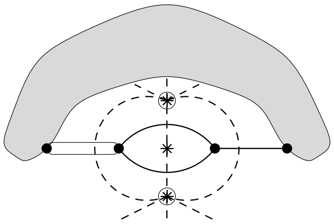

Figure 12: *The case where the nonmatching edge is incident to a double edge.*

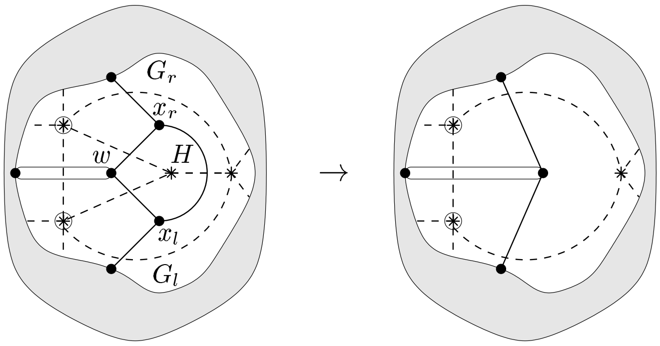

Figure 13: *The case where $G_l = G_r$ and H is a triangle.*

matched edge yet. Note that such a "triangle reduction" would be possible at any triangle and even in a nonplanar graph, but we apply it only in this case.

If $G_l = G_r$ but H is not a triangle, then there is an edge cut of cardinality two, say $\{c_1, c_2\}$ (see Figure 14). In this case, delete c_1 and c_2; contract x_l, w, and x_r to one vertex w'; and connect the two vertices of degree two with a new edge e. In the dual graph, this corresponds to deleting three edges incident to H.

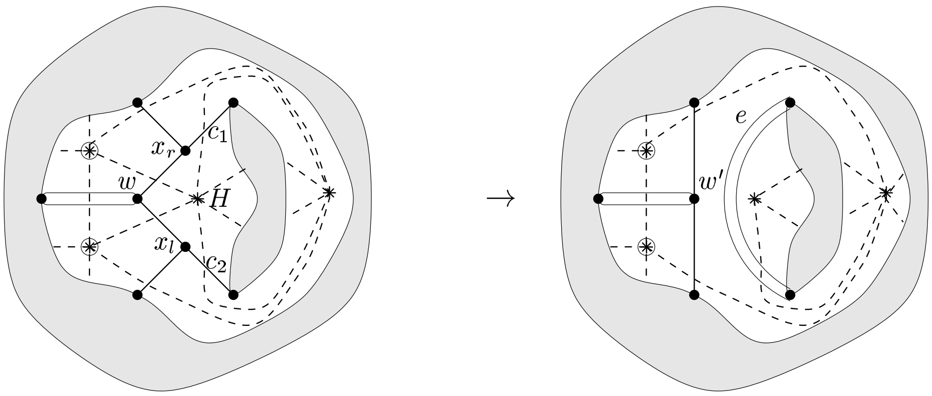

Figure 14: *The case where $G_l = G_r$ and H is not a triangle, so (G_r, H) is a multi-edge in the dual graph. We split the graph into two connected components using the edge cut $\{c_1, c_2\}$.*

The current connected component C has now split in two, the component C' containing e^{NM} and the component C_e containing the new edge e, which we want to be a nonmatching edge. Push e (the nonmatching edge in the new component C_e) onto the stack, and continue working on the other component C'. To prove that the stack invariant still holds, consider three moments in time: t_0 is the time before the reduction, t_1 is the time just after the reduction when we push e onto the stack, and t_2 is the time when we pop e from the stack. The stack invariant held at time t_0, and it still holds at time t_1, because the face $G_l = G_r$ is incident to both e and C'. We claim that it will also hold at time t_2. Let e^* be the edge below e on the stack. By the stack invariant, at time t_0 there was a face F common to e^* and C, which at time t_1 is part of C' or C_e or both. If F is part of C_e at time t_1, then it is also part of C_e at time t_2, and therefore the stack invariant holds at t_2. If F is part of C' at time t_1, then at time t_2 the component C' has been deleted, so face F has been expanded into the face common to e and C', and has thus become incident to component C_e. So again the stack invariant holds at time t_2.

Recursively compute a perfect matching in the resulting graph, and extend it by adding an edge incident to w that is not incident to a matched edge yet (see Figure 15).

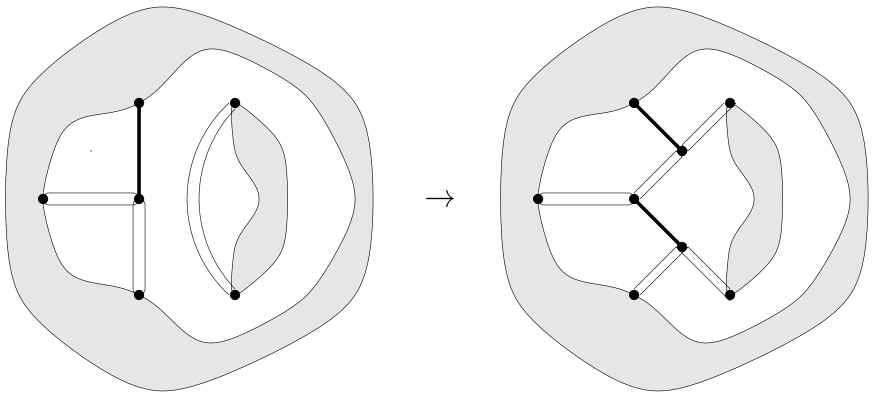

Figure 15: *The case where $G_l = G_r$ and (G_r, H) is a multi-edge in the dual graph. We show how to extend the recursively computed perfect matching to a perfect matching of G.*

This leaves the case $G_l \neq G_r$, which has already been treated in Section 4.1. Determine whether the special dual vertex is F_l or F_r, test adjacency of the special vertex to G_r or G_l, respectively, and apply the straight reduction to the appropriate edge, either (w, x_r) or (w, x_l), as described in the proof of Lemma 4.1.

If, say, we reduce edge (w, x_r), and if F_r and G_l are identical, then the primal graph becomes disconnected. Declare the other reduced edge to be a nonmatching edge, and push it onto the stack. The stack invariant holds because the face $F_r = G_l$ is incident to both nonmatching edges. An argument similar to the one used before shows that the stack invariant still holds when this newly created component is popped off the stack.

Note that we can use the data structure of the dual graph to test for distinctness, and for adjacency because F_r or F_l is special. Hence choosing the correct reduction edge, and adding entries to the stack, if necessary, takes constant time. Recursively compute a perfect matching in the resulting graph, and extend it as in Figure 4.

This finishes the discussion of all cases. As we saw in the discussion of the data structure, the total time for all of the reductions is $\mathcal{O}(n)$. The final detail is how the recursion unwinds to recover a matching in the original graph. There are two possibilities. The first option is that we could undo each reduction as we go up the recursion stack, but only maintain the primal graph and the matching in that graph. The dual graph was only needed to detect bridges, and hence we do not need to maintain it in this recovery phase. The second option is that whenever we delete edges in the primal graph, we only unlink them from the rest of the graph, and leave them allocated. This allows us to store the perfect matching in the graph at any step of the recursion, without explicitly maintaining the entire graph at each level. Either way, we obtain the following theorem.

Theorem 2 *Let G be a planar 3-regular bridgeless graph, and let e^{NM} be an edge of G. Then there exists a perfect matching of G that does not contain e^{NM}, and it can be found in $\mathcal{O}(n)$ time.*

5 Extensions

This section describes some corollaries of the fact that we can efficiently compute a perfect matching of a 3-regular bridgeless graph that does not contain a specified edge e^{NM}.

5.1 Avoiding Two Edges

Assume that we are given not one, but two edges e_1^{NM} and e_2^{NM} that should not be in the perfect matching. We claim that such a perfect matching exists and can be found in $\mathcal{O}(n \log^4 n)$ time.

Specifically, subdivide the edges e_1^{NM} and e_2^{NM}, and add a new edge e^{M} connecting the two subdivision vertices (see Figure 16). Compute a perfect matching M in the resulting 3-regular bridgeless graph in $\mathcal{O}(n \log^4 n)$ time. Now force e^{M} to be matched: if it is not, reverse an alternating cycle including it. By Lemma 3.2, this takes $\mathcal{O}(n)$ time. The edges that are part of e_1^{NM} and e_2^{NM} therefore are now unmatched. Undo the modification of the graph to obtain a perfect matching that does not contain e_1^{NM} and e_2^{NM}.

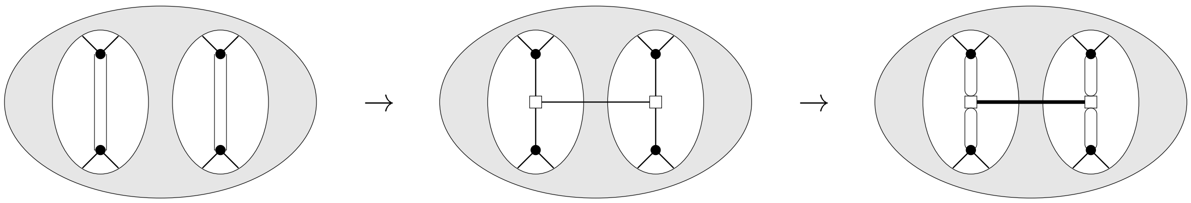

Figure 16: *We can find a matching that does not contain two specified edges by modifying the graph and revising an alternating cycle, if necessary, to force the new edge into the matching.*

We do not know whether it is possible to find this matching in linear time for planar graphs, because introducing the new edge may destroy planarity.

5.2 Petersen's Original Theorem

The *blocktree* of a graph is the tree of 2-edge-connected components, whose edges correspond to bridges in the graph. Petersen's original theorem in fact says that every 3-regular graph whose blocktree has at most two leaves (i.e., whose blocktree is a path) has a perfect matching. We can compute this matching in $\mathcal{O}(n \log^4 n)$ time as follows.

Let G be the graph. Remove all bridges of G, and obtain the connected components $G_1, \ldots, G_s$ of the resulting graph. For $i = 1, \ldots, s$, G_i has up to two vertices of degree two (if there were more, then there would be a vertex of degree three in the blocktree). Remove these vertices of degree two; connect the loose ends of their incident edges; and mark these newly added edges as nonmatching edges.

This results in a graph G'_i with up to two nonmatching edges. Compute a perfect matching of this graph using the method in the previous section. Combining the perfect matchings of G'_i, $i = 1, \ldots, s$, and adding the bridges to the matching, we obtain a perfect matching of G.

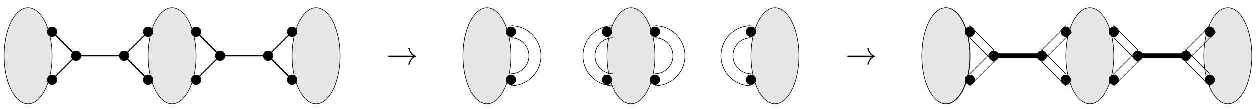

Figure 17: *If the graph has bridges, but the blocktree is a path, then we can compute a perfect matching by computing a matching in each 2-edge-connected component.*

6 Conclusion and Open Problems

It has been known for over a century that a perfect matching always exists in a 3-regular bridgeless graph, but until now, no efficient algorithm was known to find one. Our algorithms take $\mathcal{O}(n \log^4 n)$ time for nonplanar graphs, and optimal $\mathcal{O}(n)$ time for planar graphs. As a consequence, we reduce to linear time the complexity of algorithms in three application areas: terrain guarding, adaptive mesh refinement, and quadrangulation.

Several algorithmic questions about matchings in special graphs remain.

1. Can we find a perfect matching in a nonplanar 3-regular bridgeless graph in $\mathcal{O}(n)$ time?

2. How quickly can we find a maximum matching in a 3-regular graph that has bridges? Does planarity help?

3. Plesník [38] generalized Petersen's theorem to arbitrary regularity as follows: any r-regular $(r-1)$-edge-connected graph with an even number of vertices has a perfect matching not using $r-1$ given edges. In particular, for any $e \in E$, there is a perfect matching M with $e \in M$ (we say that the graph is *matching covered*). Further generalizations are also known [8, 9]. How quickly can these matchings be found?

Acknowledgments

We thank Bill Cunningham, Jack Edmonds, David Kirkpatrick, and Daniel Younger for helpful discussions. In particular, in an early meeting Jack pointed us in the right direction by suggesting we look at older proofs of Petersen's theorem. We also thank Joseph O'Rourke and two anonymous referees for helpful comments on the paper.

References

[1] D. J. Allman. A quadrilateral finite element including vertex rotation for plane elasticity analysis. *International Journal for Numerical Methods in Engineering*, 26:717–730, 1988.

[2] Marshall Bern and David Eppstein. Mesh generation and optimal triangulation. In *Computing in Euclidean Geometry*, pages 23–90. World Scientific, 2 edition, 1992.

[3] Norman L. Biggs, Keith E. Lloyd, and Robin J. Wilson. *Graph Theory 1735–1936.* Clarendon Press (Oxford), 1986.

[4] Prosenjit Bose, David Kirkpatrick, and Zaiqing Li. Efficient algorithms for guarding or illuminating the surface of a polyhedral terrain. In F. Fiala, E. Kranakis, and J.-R. Sack, editors, *Proceedings of the 8th Canadian Conference on Computational Geometry*, volume 5 of *International Informatics Series*, pages 217–222. Carleton University Press, 1996.

[5] Prosenjit Bose, Thomas Shermer, Godfried Toussaint, and Binhai Zhu. Guarding polyhedral terrains. *Computational Geometry: Theory and Applications*, 7(3):173–195, February 1997.

[6] John Boyer and Wendy Myrvold. Stop minding your P's and Q's: A simplified $O(n)$ planar embedding algorithm. In *Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 140–146, Baltimore, Maryland, January 1999.

[7] H. R. Brahana. A proof of Petersen's theorem. *Annals of Mathematics*, series 2, 19:59–63, 1917–1918.

[8] Gary Chartrand, Donald L. Goldsmith, and Seymour Schuster. A sufficient condition for graphs with 1-factors. *Colloquium Mathematicum*, 41(2):339–344, 1979.

[9] Gary Chartrand and L. Nebeský. A note on 1-factors in graphs. *Periodica Mathematica Hungarica*, 10(1):41–46, 1979.

[10] Richard Cole and Micha Sharir. Visibility problems for polyhedral terrains. *Journal of Symbolic Computation*, 7(1):11–30, January 1989.

[11] Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.

[12] David Eppstein, Zvi Galil, Giuseppe F. Italiano, and Amnon Nissenzweig. Sparsification–a technique for speeding up dynamic graph algorithms. *Journal of the ACM*, 44(5):669–696, September 1997.

[13] Alfred Errera. Une demonstration du théorème de Petersen. *Mathesis*, 36:56–61, 1922.

[14] Hazel Everett and Eduardo Rivera-Campo. Edge guarding polyhedral terrains. *Computational Geometry: Theory and Applications*, 7(3):201–203, February 1997.

[15] Leila De Floriani, Bianca Falcidieno, Caterina Pienovi, David Allen, and George Nagy. A visibility-based model for terrain features. In *Proceedings of the 2nd International Symposium on Spatial Data Handling*, pages 235–250, Seattle, Washington, July 1986.

[16] Orrin Frink, Jr. A proof of Petersen's theorem. *Annals of Mathematics*, series 2, 27:491–493, 1925-1926.

[17] Harold N. Gabow, Haim Kaplan, and Robert E. Tarjan. Unique maximum matching algorithms. In *Proceedings of the 31st Annual ACM Symposium on Theory of Computing*, pages 70–78, Atlanta, Georgia, May 1999.

[18] Harold N. Gabow and Robert E. Tarjan. A linear-time algorithm for a special case of disjoint set union. *Journal of Computer and System Sciences*, 30(2):209–221, April 1985.

[19] Michael F. Goodchild and Jay Lee. Coverage problems and visibility regions on topographic surfaces. *Annals of Operations Research*, 18:175–186, 1989.

[20] Pierre Hansen and Mao Lin Zheng. A linear algorithm for perfect matching in hexagonal systems. *Discrete Mathematics*, 122(1–3):179–196, 1993.

[21] Monika Rauch Henzinger and Valerie King. Fully dynamic 2-edge connectivity algorithm in polylogarithmic time per operation. Technical Note 1997-004a, Digital SRC, June 1997. A preliminary version appeared in *Proceedings of the 27th Annual ACM Symposium on Theory of Computing*, Las Vegas, Nevada, May 1995, pp. 519–527.

[22] Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. Poly-logarithmic deterministic fully-dynamic graph algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. In *Proceedings of the 13th Annual ACM Symposium on Theory of Computing*, pages 79–89, 1998.

[23] John Hopcroft and Robert Tarjan. Efficient planarity testing. *Journal of the ACM*, 21(4):549–568, October 1974.

[24] Claire Kenyon and Eric Rémila. Perfect matchings in the triangular lattice. *Discrete Mathematics*, 152(1–3):191–210, 1996.

[25] Dénes König. *Theorie der endlichen und unendlichen Graphen.* Akademische Verlagsgesellschaft (Leipzig), 1936. See also the English translation, *Theory of Finite and Infinite Graphs*, Birkhauser (Boston), 1990.

[26] Ming-Jun Lai and Larry L. Schumaker. Scattered data interpolation using C^2 supersplines of degree six. *SIAM Journal on Numerical Analysis*, 34(3):905–921, 1997.

[27] Jay Lee. Analyses of visibility sites on topographic surfaces. *International Journal of Geographic Information Systems*, 5(4):413–429, October–December 1991.

[28] László Lovász and M. D. Plummer. *Matching Theory.* Akadémiai Kiadó (Budapest) and North-Holland Publishing Company (Amsterdam), 1986.

[29] K. Mehlhorn and P. Mutzel. On the embedding phase of the Hopcroft and Tarjan planarity testing algorithm. *Algorithmica*, 16(2):233–242, August 1996.

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

[31] William F. Mitchell. *Unified Multilevel Adaptive Finite Element Methods for Elliptic Problems*. PhD thesis, Technical Report UIUCDCS-R-88-1436, Department of Computer Science, University of Illinois, Urbana, IL, 1988.

[32] William F. Mitchell. Adaptive refinement for arbitrary finite-element spaces with hierarchical bases. *Journal of Computational and Applied Mathematics*, 36(1):65–78, August 1991.

[33] Takao Nishizeki and Ilker Baybars. Lower bounds on the cardinality of the maximum matchings of planar graphs. *Discrete Mathematics*, 28(3):255–267, 1979.

[34] Joseph O'Rourke. *Art Gallery Theorems and Algorithms*. Oxford University Press, 1987.

[35] Joseph O'Rourke. Satellite sentries. In *Art Gallery Theorems and Algorithms*, section 10.2.4, pages 257–258. Oxford University Press, Oxford, 1987.

[36] Julius Peter Christian Petersen. Die Theorie der regulären Graphs (The theory of regular graphs). *Acta Mathematica*, 15:193–220, 1891.

[37] Paul A. Peterson and Michael C. Loui. The general maximum matching algorithm of Micali and Vazirani. *Algorithmica*, 3:511–533, 1988.

[38] Ján Plesník. Connectivity of regular graphs and the existence of 1-factors. *Matematický Časopis*, 22(4):310–318, 1972.

[39] Suneeta Ramaswami, Pedro Ramos, and Godfried Toussaint. Converting triangulations to quadrangulations. *Computational Geometry: Theory and Applications*, 9(4):257–276, March 1998.

[40] Alexander Schrijver. Bipartite edge-coloring in $O(\Delta m)$ time. *SIAM Journal on Computing*, 28(3):841–846, 1999.

[41] Thomas C. Shermer. Recent results in art galleries. *Proceedings of the IEEE*, 80(9):1384–1399, September 1992.

[42] Robert Endre Tarjan. *Data Structures and Network Algorithms*. Society for Industrial and Applied Mathematics (Philadelphia), 1983.

[43] William P. Thurston. Conway's tiling groups. *American Mathematical Monthly*, 97(8):757–773, October 1990.

[44] W. T. Tutte. The factorization of linear graphs. *Journal of the London Mathematical Society*, 22:107–111, 1947.

[45] Jorge Urrutia. Art gallery and illumination problems. In J. R. Sack and J. Urrutia, editors, *Handbook on Computational Geometry*, chapter 22, pages 973–1027. Elsevier Science B.V., 2000.

[46] Vijay V. Vazirani. A theory of alternating paths and blossoms for proving correctness of the $o(\sqrt{V}e)$ general graph maximum matching algorithm. *Combinatorica*, 14(1):71–109, 1994.