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. We have developed an $\mathcal{O}(n \log^4 n)$-time algorithm for 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.

1 Introduction

In 1891, Petersen [29] published a pioneering paper in matching theory, and he is now considered one of the two principal founders of matching theory [23, 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 [6] and Frink in 1926 [27], published simplified proofs. The interested reader can find extracts from Frink's paper [27] and Petersen's paper [29] in [3]. Frink's proof contained a slight flaw which was later corrected by König [20] in the first text-book on graph theory ever written [23, p. xvi]. Independently, Errera [12] published another proof of Petersen's theorem. Petersen's theorem is now usually known as a simple corollary of the theorem of Tutte [23, 35] 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 a perfect matching in a 3-regular bridgeless graph efficiently. The fastest known maximum matching algorithm for general graphs [25] runs in $\mathcal{O}(m\sqrt{n})$ time, where n and m are the numbers of vertices and edges, respectively. For 3-regular graphs, $m = \mathcal{O}(n)$, so this algorithm runs in $\mathcal{O}(n^{3/2})$ time. For some specialized graph classes, there are algorithms that compute a perfect matching in less time [16, 32], but none of these graph classes include all 3-regular bridgeless graphs. We develop an algorithm for such graphs with time complexity $\mathcal{O}(n \log^4 n)$, using recent results on dynamic maintenance of 2-edge-connectivity information [18].

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. The collection of all planar 3-regular bridgeless graphs is exactly the collection of duals of planar triangulations where the outside face is a triangle. A perfect matching in a 3-regular 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 for numerical simulations and for converting triangulations into quadrangulations, as described in Sections 1.2 and 1.3, respectively.

The rest of this paper is outlined 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 $O(n^2)$-time algorithm based on Frink's proof of Petersen's theorem [27], and show how to improve it to $O(n \log^4 n)$ time. Section 4 presents a linear-time algorithm for planar graphs. We conclude in Section 5.

*School of Computer Science, McGill University, 3480 University Street, Montreal, Quebec H3A 2A7, Canada, e-mail: *therese@cs.mcgill.ca*.

†School of Computer Science, Carleton University, 1125 Colonel by Drive, Ottawa, Ontario K1S 5B6, Canada, e-mail: *jit@scs.carleton.ca*.

‡Department of Computer Science, University of Waterloo, Waterloo, Ontario N2L 3G1, Canada, e-mail: {*eddemaine, alubiw*}*@uwaterloo.ca*. Supported by NSERC.

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

1.1 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 [28, 33] 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 deFloriani et al. [10] who showed that the minimum number of guards can be found using a set covering algorithm. Cole and Sharir [9] subsequently showed that the problem is NP-hard. Goodchild and Lee [15] and Lee [22] 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 to guard the surface of an n-vertex triangulated polyhedral terrain, in the worst case. The complementary result that $\lfloor n/3 \rfloor$ edge-guards are always sufficient was proved by Everett and Rivera-Campo [13]. 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, since 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.

1.2 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* [26]. 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 if 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.

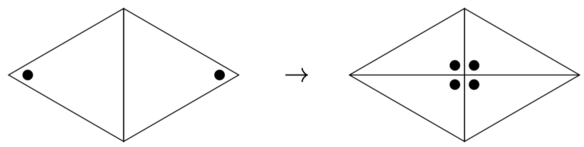

Figure 1: *A uniform newest-vertex bisection applied to a pair of triangles.*

Hence, to start applying newest-vertex bisection, we must assign the newest vertices such that every triangle is compatible. 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, where 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. Indeed, this graph is always 3-regular and bridgeless [26], so by Petersen's theorem it has a perfect matching.

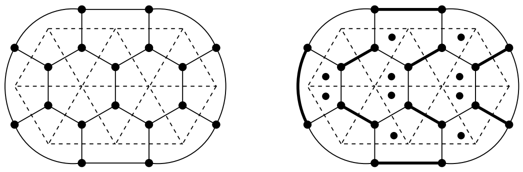

Figure 2: *The relationship between perfect matchings and pairings in triangulations. We show a triangulation with a modified dual graph, and a compatible assignment of newest vertices obtained from a perfect matching.*

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 $\mathcal{O}(n)$ time.

1.3 Quadrangulations. 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* (they are 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 other applications where quadrangles have several advantages over triangles, including scattered bivariate data interpolation [21] 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 [31].

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 $\mathcal{O}(n)$ time.

2 Terminology

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

Maximum matching is a classic problem in graph theory with many practical applications [23]. 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 $v, w \in V$. 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. A *matching* in G is a subset M of edges such that for every vertex v, at most one edge e *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* M 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) where every other edge is matched. Reversal of an alternating cycle, that is, switching matched and unmatched edges, yields another matching. An *augmenting path* is a path between two vertices without incident matched edges where every other edge of the path is matched.

A graph is *k-regular* if every vertex has degree k, that is, k incident edges. 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 $v, w \in V$ that are connected by a path in G but not in $G - C$. G is *k-edge connected* if the smallest edge cut in G has cardinality k. A *bridge* in a graph G is a 1-edge cut. 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 circular clockwise order of edges around each vertex. A planar drawing subdivides the plane into pieces 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 [19, 24]. 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.

3 Non-planar 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 [27] is available in [3]. It contains a slight flaw which was corrected in König's detailed version of the proof [20]. 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 any loop in a 3-regular graph is incident to a bridge; since G is bridgeless, it has no loops. 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, find a simple edge $e = (v, w)$, 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). Because 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.

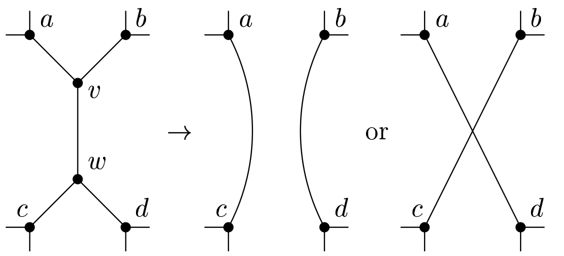

Figure 3: *The straight and crossing reductions in Frink's proof.*

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*.[2] Clearly both reductions lead to 3-regular graphs.

Lemma 3.1. [20, 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. 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 reduced edges is matched. In either case, extend the matching as in Figure 4. This argument relies on the following lemma.

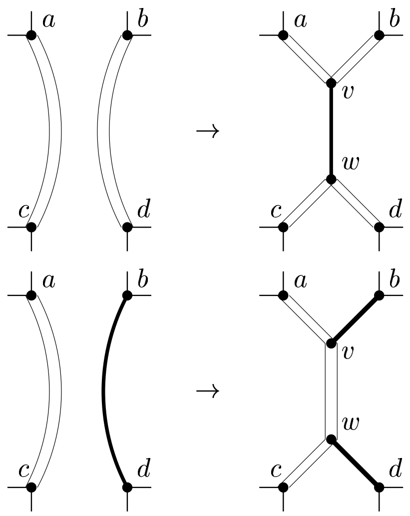

Figure 4: *Extending to a perfect matching in Frink's proof. Matched edges are drawn thick, and unmatched edges are drawn hollow.*

Lemma 3.2. *Given any 3-regular bridgeless graph with a perfect matching and some edge e in the graph, there exists an alternating cycle that includes e. The cycle can be found in linear time.*

Proof. The first part of this lemma is proved in [27] and [20, p. 187]. For the second part, we show how finding an alternating cycle reduces to finding an augmenting path, which (see e.g. [34]) can be done in $\mathcal{O}(m)$ time using the Gabow-Tarjan set union algorithm [14].

We distinguish two cases. If $e = (v, w)$ is matched, then delete e and find an augmenting path in the graph. This augmenting path must connect v and w (all other vertices have an incident matched edge) and can thus be completed to an alternating cycle containing 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 the only incident edge of v is e, this augmenting path must contain e, and can thus be completed with e_1 to an alternating cycle containing e.

[2] In the non-planar 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.

Converting induction into recursion, we have a simple algorithm for Petersen's theorem. Each step of the recursion, i.e., checking for the base case, finding an edge of multiplicity one, picking and applying an appropriate reduction, and optionally finding and reversing an alternating cycle, takes at most linear 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 where 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 *non-matching 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 was arbitrary; now choose one of the edges incident to e^{NM}. Assume 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 set the new non-matching edge to be the reduced edge that is incident to one of the ends of e^{NM}, thus either (a, c) or (a, d). In the resulting graph compute a perfect matching that does not use the non-matching edge. Hence not both reduced edges are matched, so the matching can be extended as before (Figure 4); note that this does not cause $e^{\mathrm{NM}} = (a, v)$ to be matched.

Unfortunately, all edges incident to e^{NM} may be double edges, but e is not allowed to be a double edge. In this case, perform another 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 non-matching 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.

Hence, each step of the induction takes constant time except for determining the correct reduction if e has multiplicity one. This amounts to testing whether one of the reductions results in a bridgeless graph with the same number of connected components. If not, then choose the other reduction by Lemma 3.1. By Menger's theorem, a graph is bridgeless exactly if there exist 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 [20, p. 182]. So testing whether the graph is bridgeless reduces to testing the existence of two edge-disjoint paths between a constant number of pairs of vertices.

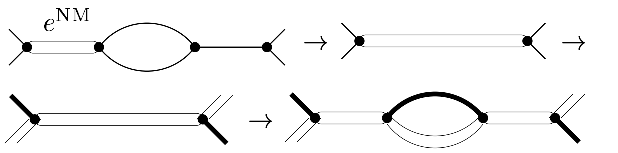

Figure 5: *Reduction and extension for a double edge incident to the non-matching edge* e^{NM}. *The non-matching edge is drawn hollow.*

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 has two edge-disjoint paths between them. 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 [11], and the best randomized result was $\mathcal{O}(\log^5 n)$ expected update time [17]. Recently, Holm, de Lichtenberg, and Thorup [18] 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 best previous algorithm running in $\mathcal{O}(n^{3/2})$ time [25].

THEOREM 3.1. *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.*

4 Planar Case

Since 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, since 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. To remedy this we exploit the remaining freedom in the choice of the reduction edge in order to avoid the crossing reduction, and instead choose between two straight reductions one of which maintains bridgelessness.

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 incident to w in counterclockwise order after v (see Figure 6). 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 third face incident to x_l, and G_r denote the third face incident to x_r.

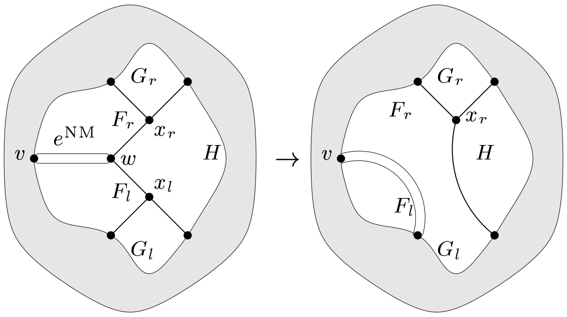

Figure 6: *Definitions of vertex and face names, and the result of reducing edge (w, x_l).*

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 6), and have to show that one of them leads to a bridgeless graph.

LEMMA 4.1. *If $G_l \neq G_r$ then applying the straight reduction to either (w, x_l) or (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. Assume we applied 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 7).

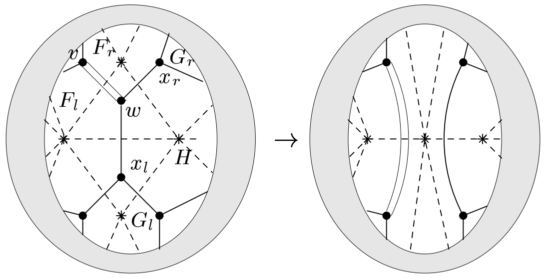

Figure 7: *The straight reduction of (w, x_l), and corresponding changes to the dual graph. Stars and dashed lines denote vertices and edges in the dual graph, respectively.*

Assume the primal graph contains a bridge after the reduction; this corresponds to a loop in the dual graph. The incident vertex of this loop must be the combined vertex $F_r = G_l$, since the primal graph was bridgeless before the reduction, and no edges have been added in the dual graph during the reduction. Hence, if the primal graph has a bridge after the reduction, then F_r and G_l must have been adjacent before the reduction.

Now if $G_l \neq G_r$, then not both (F_r, G_l) and (F_l, G_r) can be edges in the dual graph. Assume to the contrary that they were. 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. 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 6 distinct vertices since $G_r \neq G_l$. This is a contradiction, since no planar graph can contain a $K_{3,3}$.

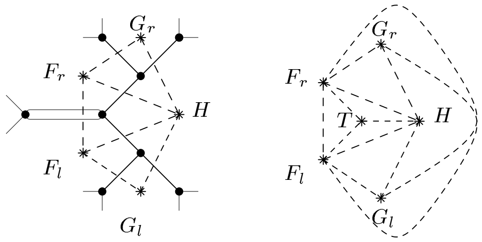

Figure 8: *If $G_l \neq G_r$, then not both $(F_l, G_r) \in E(G^*)$ and $(F_r, G_l) \in E(G^*)$; otherwise we have a planar $K_{3,3}$.*

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, 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 7 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 graph 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 isolated vertices, 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 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. Since 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, since 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 one of these suffice. Call the non-matching 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 7) disconnects the primal graph if $F_r = G_l$, so we must deal with a non-matching edge in each connected component. Thus the special edge will change as the algorithm progresses. The special vertex will retain its identity; its sibling may change, but only after the old sibling is contracted into the special vertex.

We verify in the next section that the following dual operations, together with deletions of edges and isolated vertices, are sufficient to determine which reduction should be done, and to do the reduction.

- Query whether a vertex v in the dual graph is incident to the special vertex.
- 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, we test "$v_s > 0$," 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 place 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 now be (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 update during a contraction increases e_s for some edge e, since 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, therefore a bridge in the primal graph. Hence e_s never decrease, 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 non-matching 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 that contains 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 non-matching edge e^{NM}.

e^{NM} has Multiplicity Three. If the special edge e^{NM} has multiplicity three, then its connected component C is a triple edge. 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 three faces of 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. See also Figure 9.

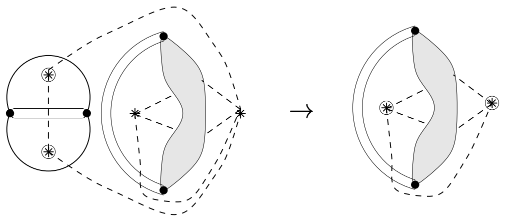

Figure 9: *The case where we transit from one connected component to the next. The special dual vertex and its sibling are circled.*

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 10). 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}.

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 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 one of the sides of the double edge.

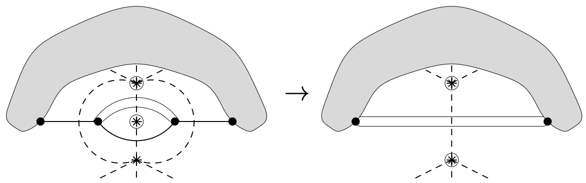

Figure 10: *The case where the non-matching edge is a double edge. The special dual vertex and its sibling are circled.*

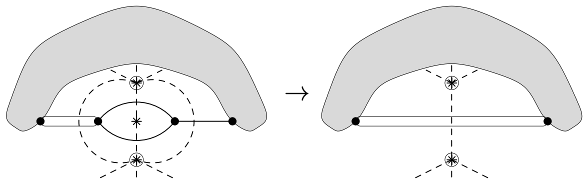

Figure 11: *The case where the non-matching edge is incident to a double edge.*

So we 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 6). 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 12). 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 matched edge yet. Note that such a "triangle reduction" would be possible at any triangle and even in a non-planar graph, but we apply it only in this case.

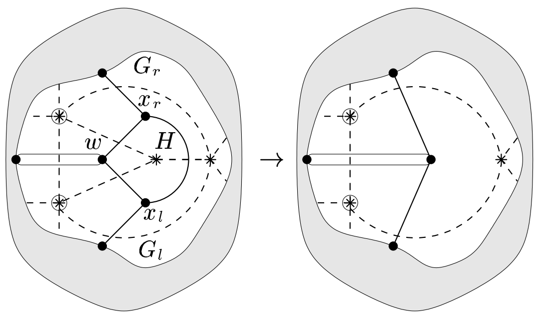

Figure 12: *The case where $G_l = G_r$ and H is a triangle.*

If $G_l = G_r$ but H is not a triangle, then there is a 2-edge cut (see Figure 13). In this case, remove the edges of the 2-cut, contract x_l, w, and x_r to one vertex w', and connect the two vertices of degree 2 with a new edge e. In the dual graph, this corresponds to deleting three incident edges of H, see Figure 13.

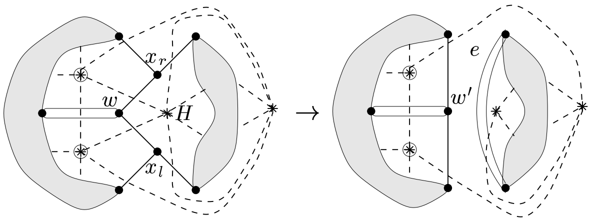

Figure 13: *The case where $G_l = G_r$ and H is not a triangle, so (G_r, H) is a multi-edge in G^*. We split the graph into two connected components.*

The current connected component C has now split in two, the component C' containing e^{NM} and the component C_e containing e, the new non-matching edge. Push e (the non-matching edge in the new component) onto the stack, and continue working on the other component. To prove that the stack invariant still holds, consider three moments of 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, since the face $G_l = G_r$ is incident to both e and C'. It will also hold at the time t_2. For let e^* be the edge below e on the stack. By the stack invariant, there existed at time t_0 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, 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, face F therefore 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 14).

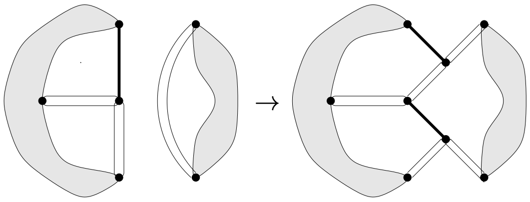

Figure 14: *The case where $G_l = G_r$ and (G_r, H) is a multi-edge in G^*. We show how to extend the recursively computed matching to a perfect matching of G.*

This leaves the case $G_l \neq G_r$, which has been treated already 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 reduced edge (w, x_r), and if F_r and G_l were identical, then the primal graph now becomes disconnected. Declare the other reduced edge to be a non-matching edge, and push it on the stack. The stack invariant holds since the face $F_r = G_l$ is incident to both non-matching edges. An argument similar to the one used before shows that it 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 since 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 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 $\mathcal{O}(n)$, hence we obtain the following theorem.

THEOREM 4.1. *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 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 very efficient algorithm was known to find one. Our algorithms take $\mathcal{O}(n \log^4 n)$ time for non-planar 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. Find a perfect matching in a non-planar 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 [30] 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. Further generalizations are also known [8, 7]. How quickly can these matchings be found?

Acknowledgments

We thank Bill Cunningham, Jack Edmonds and David Kirkpatrick for helpful discussions.

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] M. Bern and D. Eppstein. Mesh generation and optimal triangulation. In *Computing in Euclidean Geometry*, pages 23–90. World Scientific, 1992.

[3] N. L. Biggs, K. E. Lloyd, and R. J. Wilson. *Graph Theory 1735–1936*. Clarendon Press, 1986.

[4] P. Bose, D. Kirkpatrick, and Z. Li. Efficient algorithms for guarding and illuminating the surface of a polyhedral terrain. In *8th Canadian Conference on Computational Geometry*, pages 217–222, 1996.

[5] P. Bose, T. Shermer, G. Toussaint, and B. Zhu. Guarding polyhedral terrains. *Comput. Geom. Theory Appl.*, 6(3):173–195, 1997.

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

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

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

[9] R. Cole and M. Sharir. Visibility problems for polyhedral terrains. *Journal of Symbolic Computation*, 7:11–30, 1989.

[10] L. deFloriani, B. Falcidieno, C. Pienovi, D. Allen, and G. Nagy. A visibility-based model for terrain features. In *2nd International Symposium on Spatial Data Handling*, pages 235–250, 1986.

[11] D. Eppstein, Z. Galil, G. F. Italiano, and A. Nissenzweig. Sparsification – a technique for speeding up dynamic graph algorithms. *Journal of the ACM*, 44(5):669–696, 1997.

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

[13] H. Everett and E. Rivera-Campo. Edge guarding a triangulated polyhedral terrain. In *6th Canadian Conference on Computational Geometry*, pages 293–295, 1994.

[14] H. N. Gabow and R. E. Tarjan. A linear-time algorithm for a special case of disjoint set union. *J. Comput. System Sci.*, 30(2):209–221, 1985.

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

[16] P. Hansen and M. Zheng. A linear algorithm for perfect matching in hexagonal systems. *Discrete Mathematics*, 122:179–196, 1993.

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

[18] J. Holm, K. de Lichtenberg, and M. Thorup. Polylogarithmic deterministic fully-dynamic graph algorithms II: 2-edge and biconnectivity. Technical Report DIKU-TR-97/26, Dept. of Computer Science, Univ. of Copenhagen, November 1997.

[19] J. E. Hopcroft and R. E. Tarjan. Efficient planarity testing. *Journal of the ACM*, 21(4):549–568, 1974.

[20] D. König. *Theorie der endlichen und unendlichen Graphen*. Akademische Verlagsgesellschaft, Leipzig, 1936.

[21] M. J. Lai and L. L. Schumaker. Scattered data interpolation using C^2 piecewise polynomials of degree six. In *Third Workshop on Proximity Graphs*, Starkville, Mississippi, December 1994.

[22] J. Lee. Analyses of visibility sites on topographic surfaces. *Int. J. Geographic Information Systems*, 5(4):413–429, 1991.

[23] L. Lovász and M. Plummer. *Matching Theory*. North-Holland Publishing Company, 1986.

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

[25] S. Micali and V. V. Vazirani. An $O(\sqrt{|V|} \cdot |E|)$ algorithm for finding maximum matching in general graphs. In *21st Annual Symposium on Foundations of Computer Science*, pages 17–27, 1980.

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

[27] O. Frink, Jr. A proof of Petersen's theorem. *Annals of Mathematics*, 27:491–493, 1925-6.

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

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

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

[31] S. Ramaswami, P. Ramos, and G. Toussaint. Converting triangulations to quadrangulations. *Comput. Geom. Theory Appl.*, 9(4):257–276, 1998.

[32] A. Schrijver. Bipartite edge-coloring in $O(\Delta m)$ time. *SIAM J. on Computing*, 1998. To appear.

[33] T. C. Shermer. Recent results in art galleries. *Proc. IEEE*, 80(9):1384–1399, September 1992.

[34] R. E. Tarjan. *Data structures and network algorithms*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, Pa., 1983.

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