Finding Large Cycles in Hamiltonian Graphs

Tomás Feder* Rajeev Motwani†

Abstract

We show how to find in Hamiltonian graphs a cycle of length $n^{\Omega(1/\log\log n)} = \exp(\Omega(\log n/\log\log n))$. This is a consequence of a more general result in which we show that if G has maximum degree d and has a cycle with k vertices (or a 3-cyclable minor H with k vertices), then we can find in $O(n^3)$ time a cycle in G of length $k^{\Omega(1/\log d)}$. From this we infer that if G has a cycle of length k, then one can find in $O(n^3)$ time a cycle of length $k^{\Omega(1/(\log(n/k)+\log\log n))}$, which implies the result for Hamiltonian graphs. Our results improve, for some values of k and d, a recent result of Gabow [11] showing that if G has a cycle of length k, then one can find in polynomial time a cycle in G of length $\exp(\Omega(\sqrt{\log k/\log\log k}))$. We finally show that if G has fixed Euler genus g and has a cycle with k vertices (or a 3-cyclable minor H with k vertices), then we can find in polynomial time a cycle in G of length $f(g)k^{\Omega(1)}$, running in time $O(n^2)$ for planar graphs.

1 Introduction

In spite of the significant progress over the last decade in the area of approximation algorithms and hardness results, there has been very little progress in establishing positive or negative results for the problem of finding long paths and cycles[1] in undirected graphs. Until recently, there was no known algorithm which guarantees approximation ratio better than $n/\text{polylog}(n)$, and no hardness of approximation results that explain this situation. This is true even for the problem of finding long paths and cycles in *bounded-degree Hamiltonian* graphs, and indeed it has been conjectured that this very special case is already very hard to approximate. In the very recent past, there has been activity on this problem culminating in a breakthrough result of Gabow [11] which for the first time led to an algorithm that could find paths and cycles of super-polylogarithmic length in arbitrary graphs with long paths. In part based on Gabow's techniques, we improve the bounds to show how to find in Hamiltonian graphs a cycle of length $n^{\Omega(1/\log\log n)}$. This is a consequence of a more general result that is described below.

Previous Work. Karger, Motwani, and Ramkumar [15] showed that obtaining a constant factor approximation to the longest undirected path is NP-hard. Furthermore for any $\epsilon > 0$, approximating to within a factor of $2^{O(\log^{1-\epsilon} n)}$ is quasi-NP-hard. Later, Bazgan, Santha, and Tuza [1] established similar negative results for finding long paths in *cubic Hamiltonian* graphs. Stronger hardness results for *directed* graphs were obtained by Björklund, Husfeldt, and Khanna [3].

Let k be the number of vertices in a longest cycle containing a given vertex v in an undirected graph. Gabow [11] showed how to find a cycle through v of length $\exp(\Omega(\sqrt{\log k/\log\log k}))$ in

*268 Waverley Street, Palo Alto, CA 94301. Email: tomas@theory.stanford.edu

†Department of Computer Science, Stanford University, Stanford, CA 94305. Supported in part by NSF Grants IIS-0118173, EIA-0137761, and ITR-0331640, and grants from Microsoft, SNRC, and Veritas. Email: rajeev@cs.stanford.edu

[1]Generally, all results apply equally well to finding paths or cycles, with or without a specified vertex or edge, so we will ignore the differences between these variants.

polynomial time. This implies the same bound for the longest cycle, longest vw-path and longest path. The previous best bound for longest path is length $\Omega((\log k)^2/\log\log k)$ due to Björklund and Husfeldt [2].

Jackson [13] showed that 3-connected n-vertex cubic graphs have cycles through any two given edges of length at least n^c+1 for $c = \log_2(1+\sqrt{5})-1$. Feder, Motwani, and Subi [8] considered the problem of finding long cycles in 3-connected cubic graphs whose edges have weights $w_i \geq 0$, and find cycles of weight at least $(\sum w_i^a)^{\frac{1}{a}}$ for $a = \log_2 3$. A graph is 3-cyclable if for every set of three vertices u, v, w, there is a cycle going through all three vertices; in particular 3-connected graphs are 3-cyclable. It is also shown by Feder, Motwani, and Subi [8] how to find a cycle of length at least $k^{(\log_3 2)/2}$ in a graph with vertices of degree at most 3 that has a 3-cyclable minor with k edges (or in particular, has a cycle of length k), in polynomial time. Their algorithms run in $O(n^2)$ time, using a linear time algorithm for dividing a graph into triconnected components [12].

Jackson and Wormald [14] proved that 3-connected n-vertex graphs with maximum degree at most d have a cycle of length at least $\frac{1}{2}n^{\log_b 2}+1$, where $b = 6d^2$. This bound was improved by Chen, Xu, and Yu [6] to $n^{\log_b 2}+2$, where $b = 2(d-1)^2+1$, who also gave a polynomial time algorithm for finding such a cycle; the algorithm runs in time $O(n^3)$. The bound was later improved to $b = 4d+1$ by Chen, Gao, Yu, and Zang [5].

Chen and Yu [7] showed that every planar 3-connected n-vertex graph has a cycle of length at least $cn^{\log_3 2}$ for some constant $c > 0$. Their proof gives an algorithm running in time $O(n^2)$ for finding such a cycle. Böhme, Mohar and Thomassen [4] generalized this result by showing that every 3-connected n-vertex graph of Euler genus g has a cycle of length at least $f(g)n^{\log_3 2}$ for some $f(g) > 0$. It can be inferred from their proof that such a cycle can be found in polynomial time for fixed g. Sanders and Zhao [19] showed that every graph of Euler genus g has a spanning 2-connected subgraph of degree at most $6+2g$.

Results. In this paper we show that if G has maximum degree d and has a cycle of length k, or more generally a 3-cyclable minor H with k vertices, then one can find in polynomial time a cycle in G of length $k^{\Omega(1/\log d)}$. The algorithm runs in time $O(n^3)$. From this we infer that if G has a cycle of length k, then one can find in $O(n^3)$ time a cycle of length $k^{\Omega(1/(\log(n/k)+\log\log n))}$. In particular, a cycle of length $n^{\Omega(1/\log\log n)}$ is found in Hamiltonian graphs. This improves the bound of Gabow [11] for finding a long cycle in graphs containing a cycle of length k in the case of graphs of degree bounded by $d \leq \exp(o(\sqrt{\log k \log\log k}))$, and in the case where the longest cycle has length $k \geq n/\exp(o(\sqrt{\log n \log\log n}))$.

We also show that if G has Euler genus g, and has a cycle of length k, or more generally a 3-cyclable minor H with k vertices, then one can find in polynomial time a cycle in G of length $f(g)k^{\Omega(1)}$ for fixed g, where $f(g) > 0$. On planar graphs, the algorithm runs in time $O(n^2)$.

Our algorithm, like that of Gabow, requires finding a cycle through three specified vertices. A more general result of Robertson and Seymour [18] gives a polynomial-time algorithm for the fixed vertex subgraph homeomorphism problem, but with unreasonably large constants even for our case of triangles. Fortunately, there is a linear time algorithm for our case of triangles due to LaPaugh and Rivest [16], not involving any large hidden constants. Their paper gives a linear time algorithm for finding a cycle through any three given edges, if such a cycle exists. By examining all cases of their proof, one can show the following.

Lemma 1 *A 3-connected graph G has a cycle through three edges e, f, and g if and only if it is not the case that either $\{e, f, g\}$ is an edge-cut in G, or e, f, and g share a common vertex v.*

The rest of this paper is organized as follows. In Section 2, we present first a simple algorithm achieving the bound $\exp(\Omega(\sqrt{\log k/\log d}))$, and then the algorithm achieving the bound $k^{\Omega(1/\log d)}$. This second algorithm depends on two other results obtained, namely finding a cycle with at least $k^{\Omega(1/\log d)}$ special edges in a 3-connected graph of maximum degree d having k special edges; and finding in a 3-connected graph of maximum degree d whose edges have weight w_i a cycle of total weight at least $\left(\sum w_i^b\right)^{1/b}$ with $b = O(\log d)$. Then, in Section 3, we give the algorithm achieving the $k^{\Omega(1/(\log(n/k)+\log\log n))}$ bound, and in particular the $n^{\Omega(1/\log\log n)}$ bound for Hamiltonian graphs. Section 4 shows the results on graphs of fixed Euler genus.

2 Long Cycles in Graphs with 3-Cyclable Minors

Before we establish the main result of this paper, we present a simpler result with a weaker bound. This result exemplifies the main idea that will be used in the rest of the paper.

Theorem 1 *Let G be a graph with maximum degree d that has a 3-cyclable minor H with k vertices (or in particular, a cycle with k vertices). Then one can find in polynomial time a cycle in G of length at least $\exp(c\sqrt{\log k/\log d})$ for some constant $c > 0$. The algorithm runs in time $O(n^3)$.*

Proof. Let G be a graph in which we wish to find a long cycle. We assume G is 2-connected, since every 3-cyclable minor lies in a 2-connected block. If G has two vertices u, v such that $G - \{u, v\}$ is not connected and has connected components R_i, then we may decompose G into graphs G_i with vertices $V(R_i) \cup \{u, v\}$ and the edges of R_i, the edges joining the vertices of R_i to u and v, plus a copy of the edge (u, v). We may then further decompose the graphs G_i similarly, thus obtaining a tree decomposition of G into graphs G_i such that each G_i is either (1) 3-connected, (2) a cycle, or (3) a multigraph consisting of two vertices u, v joined by multiple parallel edges. In this tree decomposition whose vertices are graphs, the root graph G_0 has children G_i corresponding to edges uv in G_0, and similarly for the children G_i, until the leaf graphs G_j are reached. See Hopcroft and Tarjan [12] for an algorithm to obtain such a decomposition.

Suppose G has maximum degree d and a 3-cyclable minor H with k vertices. We may choose the root graph G_0 to be any of the graphs G_i, so we assume that H involves an edge (u, v) in the root graph G_0. Consider each descendant G_1 of G_0 containing k_1 vertices in H. For each edge (u_i, v_i) in G_1, other than (u_0, v_0) corresponding to the parent of G_1, if (u_i, v_i) is an edge that was added to substitute a subgraph R_i of G attached at u and v containing k_i of the vertices in H, we assume inductively that we have found a cycle in R_i going through (u_i, v_i) of length $\exp(c\sqrt{\log k_i/\log d})$ for some constant $c > 0$.

If G_1 has r vertices and is 3-connected, then we may find a cycle C_1 in G_1 of length at least $\exp(c' \log r/\log d)$ for some constant c' by the result of Chen, Gao, Yu, and Zang [5] for 3-connected graphs. If C_1 does not go through the edge (u_0, v_0), then we may join u_0 and v_0 by disjoint paths to the cycle C_1 since G is 2-connected, and thus obtain a cycle C_1' in G going through (u_0, v_0) of length at least $|C_1|/2 \geq \exp(c \log r/\log d)$ for some constant c. This bound applies also if G_1 is a graph with two vertices joined by at most d parallel edges, or a cycle. Thus if $r \geq \exp(\sqrt{\log k_1 \log d})$ we obtain a cycle going through (u_0, v_0) of length at least $\exp(c\sqrt{\log k_1/\log d})$.

Suppose instead $r < \exp(\sqrt{\log k_1 \log d})$. If the two edges of G_1 corresponding to subgraphs R_i of G with the most vertices, say k_i, in H are (u_2, v_2) and (u_3, v_3), then we may inductively find a cycle C_1'' in G going through (u_0, v_0), (u_2, v_2), and (u_3, v_3) of length at least $\exp(c\sqrt{\log k_2/\log d}) + \exp(c\sqrt{\log k_3/\log d})$ by 3-cyclability and the algorithm of LaPaugh and Rivest [16]. When (u_3, v_3) does not exist, the computation is simpler.

Say $k_2 \geq k_3$. If $k_2 \geq k_1(1 - 1/\exp(c\sqrt{\log k_1/\log d}))$, then the cycle C_1'' has length at least $\exp(c\sqrt{\log k_2/\log d}) + 1 \geq \exp(c\sqrt{\log k_1/\log d})$ because

$$\exp(c\sqrt{\log k_2/\log d}) \geq \exp(c\sqrt{(\log k_1 - 1/\exp(c\sqrt{\log k_1/\log d}))/\log d})$$

$$\geq \exp(c\sqrt{\log k_1/\log d}(1 - 1/(2\log k_1 \log d \exp(c\sqrt{\log k_1/\log d}))))$$

$$\geq \exp(c\sqrt{\log k_1/\log d}) - 1.$$

Finally, if $k_2 < k_1(1 - 1/\exp(c\sqrt{\log k_1/\log d}))$, then $k_2 \geq k_3 \geq (k_1/\exp(c\sqrt{\log k_1/\log d}))/r \geq k_1/\exp((1+c)\sqrt{\log k_1 \log d})$, because at most two edges of G_1 incident to a vertex v correspond to a child G_i containing a vertex of H, by 3-cyclability of H, so that at most r edges of G_1 correspond to a child G_i containing a vertex of H. The cycle C_1'' in G then has length at least $2\exp(c\sqrt{\log k_3/\log d}) \geq 2\exp(c(\sqrt{\log k_1/\log d} - 1)) \geq \exp(c\sqrt{\log k_1/\log d})$ for $c > 0$ small enough since

$$\sqrt{\log k_3/\log d} \geq \sqrt{\log(k_1 - (1+c)\sqrt{\log k_1/\log d})/\log d}$$

$$\geq \sqrt{\log k_1/\log d} - 1.$$

This completes the induction and the proof of the bound on the cycle length.

The tree decomposition of G can be found in time $O(n^2)$. The algorithm for finding a long cycle by Chen, Gao, Yu, and Zang [5] in a 3-connected G_1 takes $O(r^3)$ time. It follows from the analysis of LaPaugh and Rivest [16] that a cycle through $e_0 = (u_0, v_0)$, $e_2 = (u_2, v_2)$, $e_3 = (u_3, v_3)$ in the 3-connected graph G_1 exists if and only if (i) there is no vertex v to which all three edges e_0, e_2, e_3 are incident and (ii) $G_1 - \{e_0, e_2, e_3\}$ is connected, so the two edges e_2, e_3 can be found from a tree decomposition of $G - e_0$ in $O(r^3)$ time. Adding the complexity $O(r^3)$ over all choices of G_1 gives the $O(n^3)$ bound on the running time. $\square$

The following two results consider finding cycles going through certain special edges in a 3-connected graph.

Theorem 2 *Let G be a 3-connected graph with maximum degree d that has k special edges. Then one can find in polynomial time a cycle in G that has at least $ck^{1/(1+4(1+\log_2 3)\log_2(4d+1))}/d^{2/(1+\log_2 3)}$ special edges for some constant $c > 0$. The algorithm runs in time $O(n^3)$.*

Proof. Find a minimal subgraph H of G containing all k special edges and such that H is 2-connected. Find the tree decomposition of H into graphs H_i as in Theorem 1. For each H_i, there exists a path p whose edges are not in H joining a vertex of H_i or one of its descendants in the tree decomposition to a vertex of another H_j that is not a descendant of H_i, since G is 3-connected. We repeatedly add such paths p to H to obtain a subgraph R of G such that if paths with internal vertices of degree 2 are replaced by single edges in R, then we obtain a 3-connected graph R'. We note that the paths added to go from H to R form a forest whose trees have leaves at different H_i. The reason is that we add a path p to join the two sides of two vertices that separate H, in which case the various H_t between the H_i and the H_j joined by p will be combined into a single H_u', and we may do this in such a way as to enlarge into H_u' the root H_i, thus linking each time by a path into at least one new H_j.

If some path in R with internal vertices of degree 2 has $B \geq k^\epsilon$ special edges, where $\epsilon > 0$ will be chosen later, then G has a cycle of length B. Otherwise R' has at least $k^{1-\epsilon}/d$ vertices, and we may find in R' a cycle C of length at least $A = ck^{(1-\epsilon)/\log_2(4d+1)}$ for some constant $c > 0$ by the algorithm in Chen, Gao, Yu, and Zang [5]. Either at least $A/2$ of the edges in C correspond to edges in H or at least $A/2$ of the edges in C correspond to edges not in H.

Suppose at least $A/2$ of the edges in C correspond to edges not in H. Either (i) C contains a path p of length at least $\sqrt{A/2}$ not involving edges in H, or (ii) C contains at least $\sqrt{A/2}$ paths q not involving edges in H joining vertices in H. In case (i) we may extend $\sqrt{A/2}$ disjoint paths inside the tree in R that does not have internal vertices in H containing p, where these paths lead to different H_i from p. In case (ii) the paths q form the edges of a forest whose vertices are graphs H_i, so C reaches at least $\sqrt{A/2}$ different H_i.

Suppose instead at least $A/2$ of the edges in C correspond to edges in H. Either (iii) C contains edges in at least $\sqrt{A/2}$ different H_i, or (iv) C contains at least $\sqrt{A/2}$ edges in the same H_i. In case (iv), H_i is either 3-connected or a cycle, else C can only have two edges in H_i. If H_i is 3-connected, then each of the edges in $C \cap H_i$ must be paths that go through some special edge in a child component, otherwise they could have been removed from H while preserving 2-connectivity, by minimality of H; thus C has at least $\sqrt{A/2}$ special edges. If H_i is a cycle with at least $\sqrt{A/2}$ edges, then either at least $\sqrt{A/2}/3$ of these edges correspond to descendants of H_i containing at least one special edge, thus giving a cycle with at least $\sqrt{A/2}/3$ special edges within H; or there are at least $\sqrt{A/2}/3$ disjoint paths q coming out of H_i, where each path q has internal vertices not in H and leads to a different H_j.

Thus in all four cases (i), (ii), (iii), (iv), we have either $E = \sqrt{A/2}/3$ special edges in C, or E disjoint paths q starting at C (these paths may be of length 0) and ending in different H_i. Consider the subtree T of the tree decomposition of H extending from the root H_0 to all H_i reached by the paths q starting at C. Suppose some H_i in T is adjacent to at least F disjoint paths q where $F = \sqrt{E}$. We may select $F/(d+1)$ of the edges of H_i corresponding to these leaf paths of H_j, so that these edges do not share endpoints. The selected edges of H_i can be interconnected by a subtree T_i of H_i containing the selected edges. We may join $F/(d+1)^2$ pairs of these selected edges by disjoint paths inside T_i. These $F/(d+1)^2$ disjoint paths r correspond to paths joining pairs of the F vertices in H_j ending paths q coming from C, and each such r can be chosen to contain at least one special edge. We thus end up with $F/(d+1)^2$ disjoint paths t joining pairs of vertices in C, by following q, then r, then q, and each such t contains at least one special edge.

Suppose instead no H_i in T is adjacent to at least F of the paths q starting at C. Then we may consider the tree T' obtained from T by repeatedly removing leaves until every leaf of T' has a path q from from C. At least F of the H_i are reached by paths starting from C, including the leaf H_i in T', so at least $F/2$ of the H_i are reached and have at most one child. These reached H_i with at most one child may extend the paths to reach at least one special edge in H_i. We may then start at the leafs and go upwards to connect these H_i in pairs. In joining upwards at some H_i, at most d may reach together at the same vertex, so a factor of d is lost, giving F/d reaching joining components. There again another factor of $d+1$ is lost by matching, giving $F/(d+1)^2$ for the joining by paths r. Again we obtain paths t joining pairs of vertices in C, by following q, then r, then q, and each such t contains at least one special edge.

We shall show how to construct from this gadget with at least $G = F/(8(d+1))$ special edges a cycle going through at least $J = cG^\delta$ special edges for $\delta = 1/(1+\log_2 3)$. This will give the bound

$$J = c(F/d^2)^\delta = c(\sqrt{E}/d^2)^\delta = c(A^{1/4}/d^2)^\delta$$
$$= ck^{(1-\epsilon)/(4(1+\log_2 3)\log_2(4d+1))}/d^{2/(1+\log_2 3)},$$

or k^ϵ, whichever is smallest. Setting $\epsilon = 1/(1+4(1+\log_2 3)\log_2(4d+1))$ gives the desired bound.

So we have a cycle C and pairwise disjoint paths t joining pairs of vertices in C, where each such path contains at least one special edge, and the number of special edges in C plus the number of paths t is at least G. If the number of special edges in C is at least $G/2$ we are done. Otherwise the number of paths t is at least $G/2$. Number the paths i with internal vertices of degree 2 along the cycle $1, 2, \ldots, v$. Consider pairs of paths (i, j) along the cycle C with $i < j$ such that i, j separate the cycle plus paths t, and no (i, j') separate the cycle plus paths t for $i < j' < j$. We assume $(1, j''')$ is such a pair, if any, otherwise the following decomposition will not be needed and we proceed to the 3-connectivity analysis below. We cannot have two distinct such pairs (i, j), (i', j') with $i \leq i' < j \leq j'$. Let L be the maximum number of such pairs that pairwise satisfy $i < i' < j' < j$ or $i' < i < j < j'$. For each $0 \leq h \leq L$, consider the paths t with one endpoint between i and $i+1$ and the other endpoint between j and $j+1$ with $i < j$, such that the maximum number of pairs (i', j') with $i+1 \leq i' < j' \leq j$ such that two such pairs taken pairwise (i', j'), (i'', j'') satisfy $i' < i'' < j'' < j'$ or $i'' < i' < j' < j''$ is h. These paths t, together with the cycle C, have a tree decomposition consisting of a cycle R_0 at the root, children R_i that are 3-connected cubic graphs or three parallel edges. If R_i consists of r_i paths t containing a special edge then we can find in R_i a cycle with $r_i^{1/\log_2 3}$ special edges by the result of Feder, Motwani, and Subi [8], which may be made to go through the edge connecting to the cycle R_0 while reducing the number of special edges to $r_i^{1/\log_2 3}/2$. Thus if all the R_i for h fixed have r_h paths t, we obtain a cycle with at least $r_h^{1/\log_2 3}/2$ special edges. Since the r_h add up to at least $G/2$, we have a bound $(G/(2L))^{1/\log_2 3}/2$ on the number of special edges in the obtained cycle..

We may choose one path t for each $0 \leq h \leq L$ and remove all other paths t, so that now the cycle C has vertices $0, 1, 2, \ldots, 2L, 2L+1$ and the paths t have endpoints $(i, 2L-i)$. Since G is 3-connected, there are three disjoint paths from $2L+1$ to L. Remove a maximal number of edges from G other than C and the L paths t so that if we remove one more such edge then there are not three disjoint paths from $2L+1$ to L. Every edge e that was added to obtain three disjoint paths and still remains participates in a cut with two vertices u, v, where u is on the path going along $U = 2L+1, 0, 1, \ldots, L$ and v is on the path going along $V = 2L+1, 2L, 2L-1, \ldots, L$. Thus the three disjoint paths p, q, r from $2L+1$ to L use all such added edges e.

Define sector i to consist of the four paths $(i, 2L-i)$, $(i+1, 2L-(i+1))$, $(i, i+1)$, and $(2L-i, 2L-(i+1))$ for $0 \leq i < L-1$ for the paths described above. For each sector i, at least one of the three disjoint paths p, q, r must visit sector i. Otherwise the three disjoint paths contain edges e_p, e_q, e_r on paths that join the parts of C and the L paths t in sectors before sector i and after sector i. So for two of these three edges, say e_p and e_q, the corresponding cut vertices u_p, u_q along U and v_p, v_q along V are either all before or all after sector i, say all before sector i. Then one of e_p, e_q starts in U and the other one in V, say e_p in U and e_q in V. Therefore u_q precedes u_p in U, and v_p precedes v_q in V, and so path r must visit both u_p and v_q, say in that order, which is not possible since u_p comes after the cut e_q, u_q, v_q. Thus sectors i and $i+2$ are both visited by p and q, and so it is possible to join two vertices on p, q, r by a path contained in sectors $i, i+1$ visiting at least one of the special edges in these sectors.

Similarly, it is not possible for any one of the paths p, q, r, say p, to visit first a vertex in sector $i+2$ or later, and then visit a vertex in sector $i-2$ or earlier. Otherwise sector i is traversed forward by p, say possibly with an edge e_p, then backwards by p, say from $i+1$ to i, and then again forward by p, say possibly with an edge e'_p, while sector i is traversed by q involving possibly edges e_q, e_r, so again either two of e_p, e_q, e_r or two of e'_p, e_q, e_r must occur in cuts involving four vertices that occur all before sector i or all after sector i, which is not possible as before.

Thus if we consider the selected paths with at least one special edge joining two vertices in sectors $6i, 6i+1, 6i+2$ and on the paths p, q, r, for $0 \leq i \leq (L-3)/6$, we have at least $L/6$ such paths, which can be grouped into six cases, namely both endpoints in the same path, one of p, q, or r, or both endpoints in different paths, p and q, p and r, or q and r. At least one of these six cases occurs in at least $L/36$ of the chosen paths with a special edge. If one of the first three cases occurs, say these $L/36$ special edges occur joining vertices in p, then we may visit them by going forward along p and then backward along q, thus obtaining a cycle with at least $L/36$ special edges. If one of the last three cases occurs, say these $L/36$ special edges occur joining p and q, then we may visit them by going forward alternating between p and q, and then backward along r, thus obtaining again a cycle with at least $L/36$ special edges.

The bound is thus the maximum of $L/36$ and $(G/(2L))^{1/\log_2 3}/2$, and the tradeoff happens at $L = G^{1/(1+\log_2 3)}$. The running time $O(n^3)$ is verified by showing that deciding which edges to remove to get H from G takes $O(n^2)$ time per vertex v whose edges we wish to remove, by examining the biconnected components of G minus v. Then a tree decomposition for H is obtained in time $O(n^3)$, and the at most n trees added for R are obtained in $O(n^2)$ time each. The construction for cases (i),(ii),(iii),(iv) requires $O(n)$ traversals that can be done in $O(n^2)$ time each. Finally, removing edges after finding the three disjoint paths p, q, r takes $O(n)$ time per edge, as the graph in this case has constant degree. □

A slight modification of the preceding argument gives the following bound.

Theorem 3 *Let G be a 3-connected graph with maximum degree d that has k special edges. Then one can find in polynomial time a cycle in G that has at least $k^{1/(c \log d)}$ special edges for some constant $c > 0$. The algorithm runs in time $O(n^3)$.*

Proof. Consider again the proof of Theorem 2. If $E \geq d^8$, then $\sqrt{E}/d^2 \geq E^{1/4}$, and we may remove the power of d in the denominator, so the result follows.

An edge e that is not special, at the end of a path p with internal vertices of degree 2, and attached at the endpoint z of p may not be contracted if and only if some H_i in the tree decomposition has an edge zt as the edge corresponding to the parent of H_i, and the forests attached to the vertices of H_i other than z, t only reach the neighbor u of z in p out of all vertices in H outside of H_i. We proceed to contract the edges e of H that are not special that may be contracted according to this rule if at least one of the endpoints of p is a vertex in a 3-connected graph H_j. This requires contracting per edge of H_j at most d paths which must be of length at most d^{32}, otherwise $A \geq d^{32}$ and $E = A^{1/4} \geq d^8$. This increases the degree to at most d^{34}.

Suppose instead $E < d^8$. Then $k \leq r^{c \log d}$ for some constant $c > 0$ and $r \leq d$. If a tree in the forest that was added to H has a path P of length at least d^{64} joining two vertices x, y in H, then we may find a cycle C' in H going through x, y, and either C' has length at least d^{32}, or C' has length at most d^{32}, in which case some subpath P' of P of length d^{32} does not have any intermediate vertices in C', so P' can be extended to a cycle of length at least d^{32}. This then gives $A \geq d^{32}$ and thus $E = A^{1/4} \geq d^8$. We may thus assume that the added forests have diameter at most d^{64}.

A well know fact is that an edge of a 3-connected graph with at least five vertices may be either contracted or removed (while dropping intermediate vertices of degree 2 in a path after removal) while preserving 3-connectivity. Since the edges e of the forests that were added to H may not be removed, these edges e may be contracted, while preserving 3-connectivity. Since the forests F that were attached to H have diameter at most d^{64}, we may contract paths p' in such forests F, increasing the degree to at most d^{86}. If F has f vertices, a set S with at most cd vertices separates F into components with at most f/d vertices. We may thus select a tree T spanning S in F, with T consisting of at most d^{65} vertices, and contract T, increasing the degree to at most d^{87}. Repeating this process inductively on the components with f/d vertices gives a new forest F' with diameter at most $r = c \log_d k$ for some constant $c > 0$, with the degrees bounded by $d' = d^{87}$. By the algorithm of Chen, Gao, Yu, and Zang [5], we can find a cycle of length at least $k^{c'/\log d'} = k^{c''/\log d}$, and this cycle has at least $t = k^{c''/\log d}/r = e^{c''r}/r \geq e^{c'''r} = k^{c'''/\log d}$ edges that are not in forests F' and are thus in the graph H, for constants $c', c'', c''' > 0$.

We then proceed as in the proof of Theorem 2 with these t edges of H, except that the situation where we twice divide by $d' + 1$ does not arise. The reason is that this situation arose when we had some number f of these edges e in F hanging paths of the tree decomposition attached to a 3-connected component H_j. Each such hanging path containing e has a cycle with at least one special edge containing e, unless e has an endpoint in H_j and could not be contracted. In that case, the edge $e = zt$ has an associated component H_i joined at t by a forest, and there is a cycle containing zt and a special edge in H_i, which involves only possibly one hanging path containing H_i, so the bound f is achieved without dividing twice by $d'+1$, and the result follows as in Theorem 2. □

The case with special edges in a 3-connected graph is now generalized to the case with edges of various weights in a 3-connected graph.

Theorem 4 *Let G be a 3-connected graph with maximum degree d where edge e_i has weight $w_i \geq 0$. Then one can find in polynomial time a cycle in G of total weight at least $L_b(w) = \left(\sum w_i^b\right)^{1/b}$, where $b = c \log d$ for some constant $c > 0$. The algorithm runs in time $O(n^3)$.*

Proof. By Theorem 3, we can find a cycle that has at least k^a special edges for $a = 1/(c' \log d)$ with $c' > 0$ constant in a 3-connected graph of maximum degree d with k special edges.

Given a 3-connected graph G of maximum degree d and edge weights w_i, we may divide all edge weights by the weight of the second largest edge weight w_j. As a result, the second largest edge weight is now $w_j = 1$. Group the edges into sets $S_0, S_1, S_2, \ldots$, as follows. The set S_0 contains the two edges of largest weight, namely $z \geq 1$ and 1. The set S_i with $i \geq 1$ contains all edges of weight $1/2^i < w_j \leq 1/2^{i-1}$.

Apply Theorem 3 to each problem on G having the set S_i as special edges. If S_i has s_i special edges, then we find a cycle with $r_i \geq s_i^a$ special edges. For S_0, we have $r_i = s_i = 2$ and the solution has total weight $z + 1$. For S_i with $i \geq 1$, the solution has total weight at least $r_i/2^i \geq s_i^a/2^i$. Let $b = c/a$ for some constant $c > 0$ to be chosen later. If the solution found has total weight f, then f^b is at least the maximum of $(z+1)^{c/a}$ and $s_i^c/2^{ic/a}$. Define $x \geq z$ and $t_i \geq s_i$ so that $f^b = (x+1)^{c/a} = t_i^c/2^{ic/a}$ for all $i \geq 1$. Then, for $c \geq 2$

$$
\begin{aligned}
(L_b(w))^b &\leq x^{c/a} + 1 + \sum_{i \geq 1} t_i/2^{(i-1)c/a} \\
&= x^{c/a} + 1 + \sum_{i \geq 1} \left(f^b 2^{ic/a}\right)^{1/c}/2^{(i-1)c/a} \\
&= x^{c/a} + 1 + \sum_{i \geq 1} \left(f^b\right)^{1/c}/2^{((c-1)i-c)/a} \\
&\leq x^{c/a} + 1 + \left(f^b\right)^{1/c}(2^{1/a} + 1 + 2^{-1/a} + \ldots) \\
&\leq x^{c/a} + 1 + \left(f^b\right)^{1/c}(2^{1/a} + 2) \\
&= x^{c/a} + 1 + (x+1)^{1/a}(2^{1/a} + 2) \\
&\leq (x+1)^{c/a} = f^b
\end{aligned}
$$

where the last inequality holds for c constant large enough. Thus the solution found has total weight at least $f \geq L_b(w)$, completing the proof of the bound. The running time is dominated by a single execution of the $O(n^3)$ algorithm in Theorem 3, as the bounds for each S_i can be evaluated to find the S_i giving the maximum lower bound before executing the algorithm. □

Finally, the case of 3-connected graphs with edge weights is applied to give the result for 3-cyclable graphs, or graphs with a large 3-cyclable minor.

Theorem 5 *Let G be a graph with maximum degree d that has a 3-cyclable minor H with k vertices (or in particular, a cycle with k vertices). Then one can find in polynomial time a cycle in G of length at least $k^{1/(2c\log d)}$ for the constant $c > 0$ from Theorem 4. The algorithm runs in time $O(n^3)$.*

Proof. Let G be a graph in which we wish to find a long cycle. We assume G is 2-connected, since every 3-cyclable minor lies in a 2-connected block. Find as in Theorem 1 the tree decomposition of G into graphs G_i such that each G_i is either (1) 3-connected, (2) a cycle, or (3) a multigraph consisting of two vertices u, v joined by multiple parallel edges.

Assume first G itself is 3-cyclable. We assign a weight w_i to each G_i in the tree decomposition, so that if G_i and all its descendant G_j have at least $n_i + 1$ vertices, then $w_i \geq {n_i}^a$, where $a = 1/b = 1/c\log d$. We do this inductively, starting at the leaves. So suppose for a given G_i, we have assigned weights w_j to the children graphs G_j. This assigns weight w_j to the edge of G_i corresponding to G_j; an edge of G_i not corresponding to a child of G_i is assigned weight 1, and the edge of G_i corresponding to the parent of G_i is assigned weight 0. If G_i is 3-connected, apply then Theorem 4 to G_i, obtaining a cycle of weight $w \geq \left(\sum w_j^b\right)^{1/b} \geq \left(\sum n_j\right)^{1/b} = n_i^a$. If G_i is a cycle, then we obtain a cycle of weight $w = \sum w_j \geq \sum n_j^a \geq n_i^a$. If G_i consists of multiple parallel edges, then G_i has at most one child G_j, otherwise G would not be 3-cyclable; we obtain a cycle of weight $w = w_j \geq n_j^a = n_i^a$. Assign weight w to G_i, completing the induction.

Unfortunately, the cycles that we selected for each G_i to define the weights of G_i do not necessarily connect, since the cycle for a child G_j may not go through the special edge e that links it to its parent in the tree. We shall define cycles that do go through e so that if G_i has weight w, then the cycle through e visits at least $w^{1/2} \geq n_i^{a/2}$ edges other than e, and this will complete the proof.

Consider a G_i whose weight w was defined by finding a cycle C of weights w_j, and suppose for each j we have found a path inside the corresponding G_j of length at least ${w_j}^{1/2}$. We write $w_j = \epsilon_j w$, with $\sum \epsilon_j = 1$. Clearly, if C goes through e, we can just select C and have a cycle through e of length at least $\sum {w_j}^{1/2} \geq w^{1/2}$, since $\sum {\epsilon_j}^{1/2} \geq 1$ where the sums are over the weights in C.

Suppose C does not go through e. By 2-connectivity, we can obtain two disjoint paths joining the two endpoints of e to C, respectively, such that the two disjoint paths enter C at two distinct vertices, thus decomposing the weights of C into two subsets S and T delimited by these two vertices. If $\sum {\epsilon_j}^{1/2} \geq 1$ when the sum is taken over the weights in either S or T, then we are done. If this is not the case, then we will show that the largest weight $w_1 = \epsilon_1 w$ in S and the largest weight $w_2 = \epsilon_2 w$ in T satisfy ${\epsilon_1}^{1/2} + {\epsilon_2}^{1/2} > 1$. If $w_2 = 1$, then the cycle through e and S goes through the edge of weight w_1 and at least one other edge (of weight at least $w_2 = 1$). So we can assume $w_1, w_2 > 1$, and obtain a cycle through e and the edges of weight w_1, w_2 by 3-cyclability.

It remains to prove the claim about the sum of the $\epsilon_j^{1/2}$. Notice that if $w_j \geq w_{j'}$ and $0 \leq \delta \leq w_{j'}$, then ${w_j}^{1/2} + {w_{j'}}^{1/2} \geq (w_j + \delta)^{1/2} + (w_{j'} - \delta)^{1/2}$. For the argument, we modify the weights by choosing the appropriate values δ, as follows. We can ensure that S will have at most one non-zero weight smaller than w_1, and similarly that T will have at most one non-zero weight smaller than

w_2. Thus S has $s \geq 1$ weights equal to w_1 and one weight $0 \leq w_1' < w_1$; similarly T has $t \geq 1$ weights equal to w_2 and one weight $0 \leq w_2' < w_2$.

We thus have $s\epsilon_1 + \epsilon_1' + t\epsilon_2 + \epsilon_2' = 1$, with $s\epsilon_1^{1/2} + {\epsilon_1'}^{1/2} < 1$ and $t\epsilon_2^{1/2} + {\epsilon_2'}^{1/2} < 1$. We write $\epsilon_1' = \lambda\epsilon_1$ with $0 \leq \lambda < 1$, and let $s' = s + \lambda$. Similarly, we write $\epsilon_2' = \mu\epsilon_2$ with $0 \leq \mu < 1$, and let $t' = t + \mu$. Then $s'\epsilon_1 + t'\epsilon_2 = 1$. Also, $s'\epsilon_1^{1/2} \leq (s + \lambda^{1/2})\epsilon_1^{1/2} < 1$, and similarly $t'\epsilon_2^{1/2} < 1$. But then $s'\epsilon_1 < \epsilon_1^{1/2}$ and $t'\epsilon_2 < \epsilon_2^{1/2}$, so $\epsilon_1^{1/2} + \epsilon_2^{1/2} > 1$. This completes the proof for the case where G is 3-cyclable.

Suppose G is not 3-cyclable. We proceed to simplify the graph as follows, from the leaf graphs G_j up to the root. If G_i consists of multiple parallel edges, we keep only one child G_j, the one with most vertices, so that for other children we are left with an edge in G_i that corresponds to no child.

As we go up the tree, we must also make sure we will be able to obtain, from a cycle in a graph G_i, a new cycle going through the special edge e. This is not possible if the two edges f_1 and f_2 with weights w_1 and w_2, together with e, separate the 3-connected graph G_i, or if e, f_1, f_2 are all incident to the same vertex v. For one of f_1, f_2, the one corresponding to a child G_j of smaller size, we remove the child G_j.

If the 3-cyclable minor of size k reaches the root of the tree, then the simplified graph will have $n \geq k$ vertices, by induction from the leaves up to the root of the tree of G_i The reason is that when f_1 or f_2 is removed or reduced to a single edge, the 3-cyclable minor of size k necessarily does have vertices in one of the two subgraphs corresponding to f_1 and f_2. We then get a cycle of length at least $n_i^{a/2}$ with the previous algorithm.

If the 3-cyclable minor of size k does not reach the root of the tree consider the G_i closest to the root that it reaches. The 3-cyclable minor does not have vertices corresponding to the special edge e connecting this G_i to its parent. If G_i is 3-connected, we do not simplify the graph for pairs of edges f_1, f_2 which together with e disconnect the graph of G_i component (neither in the case of P_3 nor in the case of a 3-connected graph); we assign to e weight 1. If G_i consists of multiple parallel edges, then we keep the two children of G_i with the most vertices. As before, we have $n_i \geq k$ vertices by induction from the leaves of the tree up to this G_i, and get a cycle of length at least $n_i^{a/2}$ with the previous algorithm.

The tree decomposition can be found in time $O(n^3)$, and the running time for each G_i is dominated by the cubic time algorithm of Theorem 4. □

3 Long Cycles in Hamiltonian Graphs

We infer from Theorem 5 that one can find long cycles in Hamiltonian graphs.

Theorem 6 *Let G be an n-vertex graph that has a Hamiltonian cycle. Then one can find in G a cycle of length at least $n^{1/(c \log\log n)}$ for some constant $c > 0$ in $O(n^3)$ time.*

Proof. The first stage of the algorithm finds a spanning tree of G with vertices of degree $O(\log n)$. Suppose we have found a spanning forest F consisting of r trees with vertices of degree at most d. Initially F consists of the n vertices and no edges. Find a maximal matching M of trees in F, where two trees t_1, t_2, may be matched if they are joined by an edge e, which can be added to F. This partitions F into two sets of trees F_1, F_2, where the trees in F_1 are matched, while there are no edges joining trees in F_2. There must exist a matching of the trees in F_2 into vertices in F_1 given by edges joining F_2 to F_1 in the Hamiltonian cycle. Find such a matching M'. After adding the two matchings M and M' to F, we have a forest F' with at most $r/2$ trees and vertices of degree at most $d+2$. Repeating this process $\log_2 n$ times, we obtain a single spanning tree with vertices of degree at most $2\log_2 n$.

The second stage of the algorithm finds a spanning 2-connected subgraph of G with vertices of degree $O(\log n)$. Suppose we have found a spanning subgraph H with r_1 blocks, of which r_2 do not have exactly 2 cutpoints, and with r_3 cutpoints, of which r_4 do not belong to exactly 2 blocks. Let $r' = r_2 + r_4$, and let d be the maximum degree in H. Initially H is the spanning tree found before. Consider the maximal sequences $b_1, c_1, b_2, c_2, b_3, \ldots, ...$ where b_1 is a block with only cutpoint c_1, each c_i is a cutpoint belonging only to the two blocks b_i and b_{i+1}, and each $b_i, i \geq 2$ is a block with only cutpoints c_{i-1} and c_i. For each such sequence, say $a_1, a_2, a_3, \ldots$ proceed as follows. Join a_1 to the last a_i that a_1 has an edge to, then join one of $a_2, \ldots, a_{i-1}$ to the last a_j with $j > i$ that there is an edge to, then join one of $a_i, \ldots, a_{j-1}$ to an $a_{j'}$ with $j' > 1$, and so on. This combines an initial set of blocks $b_1, b_2, \ldots b_{j''}$ into a single block b' that does not have edges to subsequent blocks in the sequence, while increasing the degrees by 2. Assume thus that each b_1 has this property stated for b'.

We now proceed as for F above. Find a maximal matching M of components b_1 starting the above sequences. The unmatched components b_1 can be matched by M' to vertices not in the corresponding sequence $b_1, c_1, \ldots$, with the following possible exception. The possible exception is the cutpoint belonging to at least 3 blocks after the sequence, and not in other choices of unmatched components b_1, where each vertex is used for at most 2 choices of b_1, since the Hamiltonian cycle contains such an M'. Adding the edges of M and M', at most 3 edges per vertex, we are left with the case where the sequences $b_1, c_1, \ldots$, consist of just b_1. The same process again matches each b_1 to vertices not in b_1, reducing the number of such b_1 by half and adding at most 3 edges per vertex. Reducing the number of b_1 by half also reduces the value r' defined above to at most $4r'/3$. Since this process adds a total of 6 edges to each vertex and can be repeated at most $\log_{4/3} n$ times, the number of edges added to each vertex is at most $6\log_{4/3} n$. In the end, we have obtained a 2-connected spanning subgraph H with degrees at most $2\log_2 n + 6\log_{4/3} n$.

If the 2-connected spanning subgraph H with vertices of degree $O(\log n)$ is 3-cyclable, then we can apply Theorem 5 and obtain a cycle of length at least $n^{1/(c \log\log n)}$. We shall satisfy the degree bound $O(\log n)$, and obtain a 2-connected spanning subgraph H' that is not necessarily 3-cyclable, but satisfies the conditions for the algorithm of Theorem 5, thus obtaining a cycle of length at least $n^{1/(c' \log(c \log n))} = n^{1/(c'' \log\log n)}$. Obtain the tree decomposition of H into graphs H_i. This decomposition does not satisfy the conditions implied by 3-cyclability needed for Theorem 5 if there are graphs H_i with edge $e = (u, v)$ containing either (i) a set of at least two other edges f_j incident to u such that each such f_j correspond to a child H_j of H_i; (ii) same as (i) for v; (iii) two edges g_1, g_2 that together with e separate H_i.

Let T be the subtree of the tree decomposition containing the root H_0 and every H_i that does not satisfy the conditions, such that every leaf of H_i does not satisfy the conditions. The leaves H_i of T are in paths $H_1, H_2, \ldots, H_t$ such that each H_i for $2 \leq i \leq t$ has only the child H_{i-1} in T. We shall take care of satifying the conditions for each H_i that does not satisfy the conditions in such paths with only an increase in degree by a constant, thus not including such paths from T. Since this needs to be done at most $O(\log n)$ times as above, the total increase in degree is $O(\log n)$.

For each such path $H_1, H_2, \ldots, H_t$, join H_1 to the latest H_i it has an edge to, then join an H_j with $j \leq i$ or one of its descendants not in H_{j-1} to the latest $H_{i'}$ it has an edge to, otherwise proceed to H_{i+1}, and so on. This increases the degrees by at most 2, and shortens the path so that in the resulting path no two H_i are joined by an edge. Consider the earliest H_p in this new path that has an edge going out of the path in the Hamiltonian cycle, and such that H_p does not satisfy the conditions. Suppose we have taken care of each H_i with $i < p$ by adding edges for H_i and its descendants not in H_{p-1} with total increase in degree $O(\log^{c-1} n)$. It suffices then to include an edge from H_j to a vertex not in the path. This can be done as before by first finding a maximal matching among such H_j in different paths, possibly coming out of the descendants of H_j, and then matching the unmatched H_j to vertices not in their path or in other unmatched H_j, with an increase of at most 3.

It thus suffices to take care of satisfying the conditions for each H_i with $i < p$ with edges within H_i and its descendants not in H_{i-1}. This can be done in such a way that takes care of all H_i with $i < j'$ for some j' that will be guaranteed to satisfy $j' \geq p$. We first do this for case (i) for H_i that has edge $e = (u, v)$ connecting to its parent. There is thus a set of at least two other edges f_j incident to u such that each such f_j correspond to a child H_j of H_i, and say f_0 corresponds to the child H_{i-1} on the path. Since there are no edges coming out of H_{i-1}, the Hamiltonian cycle must have edges coming out of H_j with $f_j \neq f_0$. Again we find a maximal matching among these H_j, and then match the remaining unmatched H_j to vertices not in any unmatched H_j and contained in H_i or its descendants other than H_{i-1}, with an increase in degree of at most 3. Note that the vertices u and v of the edge $e = (u, v)$ may occur in several such H_i, so we try to avoid edges to both u and v. If this cannot be done, then the Hamiltonian cycle is required to have such edges. In that case, we go up the path $H_1, \ldots, H_t$ and choose in each case whether to avoid u or v, with no u or v being used for more than 2 graphs H_i, since these vertices have only 2 edges incident to them in the Hamiltonian cycle. When this can be done no longer, we have taken care of all H_i with $i < j'$ with $j' \geq p$. Note that in the special case of $H_i = H_1$, there is no particular edge f_0 corresponding to a special child H_{i-1}, so the above matching procedure can leave out any one f_j for which H_j remains unmatched. Thus for case (i), and similarly case (ii), the degree goes up only by a constant.

For case (iii), two edges g_1, g_2 that together with $e = (u, v)$ separate H_i. The general case has pairs of edges g_{1q}, g_{2q} with $1 \leq q < s$ that together with e separate H_i into components $R_1, R_2, \ldots, R_s$, such that e joins R_1 to R_s, and the edges g_{1q}, g_{2q} join R_q to R_{q+1}. An edge g_{jq} may correspond to a child graph H_{jq} that is a cycle, in which case g_{jq} represents a path $g_{jq1}, g_{jq2}, \ldots, g_{jqr}$. In such a path, we may join g_{jq1} to the latest g_{jqh} that it has an edge to, then join one of $g_{jq1}, \ldots, g_{jqh}$ to the latest $g_{jqh'}$ it has an edge to, and so on, with a constant increase in degree that reduces the number of edges g_{jqh} on the path and guarantees that no two g_{jqh} on the path are joined by edges. If there is an edge joining a g_{1qh} to a $g_{2qh'}$, then we add such an edge, thus introducing an intermediate R_j between R_q and R_{q+1} with g'_{1q}, g'_{2q} joining R_q to R_j and g''_{1q}, g''_{2q} joining R_j to $R_q{+}1$, and so on, with a constant increase in degree after which it is guaranteed that no edge joins a g_{1qh} to a $g_{2qh'}$ as well. Finally, consider the sequence $R_1, (g_{11}, g_{21}), R_2, (g_{12}, g_{22}), R_3, (g_{13}, g_{23}), \ldots, R_s$. Denote this sequence by $A_1, A_2, \ldots, A_{2s-1}$. Join A_1 to the latest A_j it has an edge to. If A_j is (g_{1j}, g_{2j}), then join to both g_{1j} and g_{2j} if possible, and also join as late in the corresponding paths of g_{1jh} and g_{2jh}. Then join one of $A_1, \ldots, A_j$ to the latest $A_{j'}$ with $j' > j$. If such an edge can only be made to come out of g_{1j} or g_{2j}, then if g_{1jh} and $g_{2jh'}$ were reached, then the edge must be chosen coming out of no later than $g_{1j(h+1)}$ or $g_{2j(h+1)}$. Proceed then similarly with an edge out of one of $A_1, \ldots, A_{j'}$. By the time we reach A_s, with a constant increase in degree, there are no pairs of edges $g_{1jh}, g_{2jh'}$ which form a cut together with e, thus taking care of H_i. As in case (i), the vertices u, v from the edge e may be used for multiple H_i, so again we must avoid them if possible. We thus go up the path $H_1, \ldots, H_t$ while guaranteeing that each u, v is used in at most 2 graphs H_i, since these vertices have at most 2 incident edges in the Hamiltonian cycle. When this can be done no longer, we have taken care of all H_i for $i < j'$ and with $j' \geq p$ as before.

This completes the algorithm that finds a graph with degrees $O(\log n)$, so that Theorem 5 can be applied to find a cycle of length at least $n^{1/(c \log \log n)}$ for some constant $c > 0$. Each iteration requires finding matchings for the H_i in time $O(n^{2.5})$, for a total of $O(n^{2.5}\log n)$ time over the $O(\log n)$ iterations to reduce to the problem of Theorem 5, whose solution takes $O(n^3)$ time. ☐

In the above proof, the degree of the spanning tree can be improved to 3 in polynomial time, see Fürer and Raghavachari [10], and the degree of the 2-connected spanning graph can be improved to $O(\log n/\log\log n)$ in time $n^{O(\log n/\log\log n)}$, see Ravi, Raghavachari and Klein [17]. For more details along these lines, see Feder, Motwani, and Zhu [9].

Theorem 6 generalizes to the case of graphs with very long cycles as follows. The proof is a special case of a more general result that appears in [9].

Theorem 7 *Let G be an n-vertex graph that has a cycle of length k. Then one can find in G a cycle of length at least $k^{1/(c(\log(n/k)+\log\log n))}$ for some constant $c > 0$ in $O(n^3)$ time.*

Proof. We first find a tree in G of maximum degree $O(r(n/k)\log^2 n)$ that contains at least $k(1-1/r)$ of the vertices of the cycle K of length k, for any $r \geq 2$. Let $t = r\log n$. We consider a series of $\log n$ phases $i = 0, 1, \ldots, \log n$ during which we select edges to form a subgraph H. In phase i, we consider components of H that have between 2^i and 2^{i+1} vertices. By the end of the phase, these components will be combined into components with at least 2^{i+1} vertices, or not considered as part of H. The number of vertices that are thus removed from H but belong to K in each phase will be at most k/t, and the degree increase in each phase will be at most $2tn/k$. Thus over all $\log n$ phases we have removed from H a total of $(k/t)\log n = k/r$ vertices that belong to K, and the maximum degree is at most $(2tn/k)\log n = (2rn/k)\log^2 n$, as required.

Consider H as selected at the beginning of phase i. The components that have between 2^i and 2^{i+1} can be joined in pairs by a maximal collection of disjoint paths that do not go through H and are added to H. Eventually, we are left with such components that can not be combined in pairs by paths not going through H. Suppose there are at least k/t vertices of K that belong to such components. Then the number of such components that contain vertices of K is at least $s = \lceil k/(t2^{i+1})\rceil$. These components can be joined together by the cycle K. These s components have s representative vertices in K that can be joined in pairs as disjoint paths. These disjoint paths necessarily go through other components of H that have at least 2^{i+1} vertices, so we can set a flow problem where the sources are the components having between 2^i and 2^{i+1} vertices and the sinks are the components having at least 2^{i+1} vertices, thus obtaining at least $s \geq \lceil k/(t2^i)\rceil$ paths. Since the total number of such components to be joined is at most $n/2^i$, the computation of disjoint paths by flow happens at most $(n/2^i)/(k/(t2^i)) = tn/k$ times, for an increase of degree at most tn/k during a phase. When such a flow can no longer be found, the phase is complete, which happens only when fewer than k/t vertices of K belong to such components that have between 2^i and 2^{i+1} vertices. The time complexity is dominated by pushing $O(n)$ units of flow in $O(m)$ time each, for a total $O(nm)$ time.

We may assume G is 2-connected by considering individual 2-connected components. We find a 2-connected subgraph of G of maximum degree $O(r(n/k)\log^6 n)$ that contains at least $k(1-1/r)$ of the vertices of the cycle K of length k, for any $r \geq 2$. Let $t' = r\log^3 n$. We initially find the tree of degree $O(r(n/k)\log^2 n)$ above. We shall gradually add paths to this tree H until H becomes 2-connected. At any stage, H consists of a collection of disjoint 2-connected components joined by single vertices in a tree structure. The degree of a component is the number of other components joined at single vertices to it. Consider removing all paths joining leaf components to components of degree at least 3, going through components of degree 2. If we repeatedly remove these paths, the number of paths going through components of degree 2 in H is halved each time, so we may remove such hanging paths at most $\log n$ times. This sets up $\log n$ phases, where each phase considers the hanging paths of components.

A phase that deals with hanging paths of components first begins by consider each such path, one at a time. It starts with the leaf component and finds the path not going through H that connects to the latest component on the path of components starting at this leaf, thus forming a single 2-connected component that constitutes a new leaf, and the operation is performed again. Since vertices in the earlier leaf are not used to start a new path, this increases degrees by at most 2. When the leaf component can no longer be connected, we proceed to the parent of the leaf component as we had with the leaf. In the end, we have reduced the number of components on the path as much as possible by paths not going through H, and increased the degree by at most 2.

The hanging paths of components are then taken care in $\log n$ sub-phases by considering paths of components with between $n/2^{i+1}$ and $n/2^i$ vertices in phases $0 \leq i < \log n$. For each such path of components, we consider the components containing the bottom half of vertices closer to the leaf. As with the construction of the tree before, we first connect these to one another by disjoint paths not going through H, and when this is no longer possible, find disjoint paths for the remaining ones joining them to the rest of the graph. We divide the at most 2^i bottom halves into groups of size 2^j with $1 \leq j \leq i$, and attempt in sub-sub-phase j flows joining the bottom of one part of size 2^{j-1} to the top of the other part of size 2^{j-1} in successive flows giving disjoint paths. Each sub-sub-phase may not join in flows at most of $(k/t')/(n/2^i)$ bottom halves, or $(k/t')/(n/2^j)$ bottom halves per group. Since the number of bottom halves per group is 2^j, the degrees increase by at most $t'n/k$ in each sub-sub-phase, or $(t'n/k)\log^3 n = O(r(n/k)\log^6 n)$ over all $\log^3 n$ sub-sub-phases. The k/t' vertices of K that may not be joined in a sub-sub-phase give a total of $(k/t')\log^3 n = k/r$ vertices over all $\log^3 n$ sub-sub-phases. The time complexity is dominated by pushing $O(n)$ units of flow in $O(m)$ time each, for a total $O(nm)$ time.

Finally, we strengthen the conditions on the 2-connected subgraph found by requiring that the conditions implied by 3-cyclability that are used in Theorem 5 hold. These conditions for a 2-connected graph with tree decomposition into G_i are: (1) there is at most one edge incident to u in G_j other than uv that corresponds to a child graph $G_{j'}$ of G_j; (2) there is at most one edge incident to v in G_j other than uv that corresponds to a child graph $G_{j'}$ of G_j; (3) if three edges uv, e, f separate G_j into two parts, then at most one of u, v corresponds to a child graph $G_{j'}$ of G_j.

We proceed as we did to obtain the 2-connected subgraph from the tree, but instead of starting with the tree, we begin with the tree decomposition of the 2-connected subgraph. We thus proceed to $\log n$ phases for the hanging paths of the tree decomposition, with each phase grouped into $\log n$ sub-phases and each sub-phase grouped into $\log n$ sub-sub-phases. Again, we attempt in each sub-sub-phase to link the bottom half of the hanging paths of G_j so that the G_j in the top halves are not separated by just 2 vertices. However, we measure the number of vertices to be obtained for a solution from such a G_j not by the total number of vertices in G_j, but by the number of vertices that would have to be removed to satisfy conditions (1),(2),(3) if G_j remained only 2-connected, that is, if we counted only the vertices corresponding to only one edge incident to u other than uv, counted only the vertices corresponding to only one edges incident to v other than uv, and for every pair of edges e, f such that uv, e, f separate G_j counted only the vertices for one of e, f.

Before considering connecting the bottom halves of paths among one another and to the rest of the graphs, we process each path individually, by going up the path from the leaves G_j, joining G_j together as much as possible. In addition, to attempting to join G_j as close as possible to the leaf as possible to an ancestor on the path, we consider taking care of each G_j encountered by going up the path individually, in order to partially satisfy conditions (1),(2),(3) within G_j. For condition (3), the pairs of edges e_i, f_i that together with uv separate G_j into two parts decompose G_j into subgraphs $H_1, \ldots, H_s$ such that uv joins H_1 to H_s, and e_i, f_i join H_i to H_{i+1}, and each e_i, f_i corresponds to a path of edges in a child of G_j. We may thus consider the sequence $H_1, (e_1, f_1), H_2, (e_2, f_2), H_3, \ldots, H_{s-1}, (e_{s-1}, f_{s-1}), H_s$, and proceed as follows. First join H_1 to the last item in this sequence by a path, and if this last item is (e_i, f_i) then join H_1 to the last possible edge on the two paths for e_i and for f_i. Then proceed similarly starting anywhere up to this last item, which is now part of H_1. If it is not possible to proceed from H_1, we proceed through the edges in the paths for e_1, f_1 by connecting them to the last possible edge similarly. Each time we add intermediate vertices of maximum degree 3, the degree of the vertices from which the paths are started increases by 2, and the degree of the vertices where the paths are ended increases by 1, for a total increase of 3 in the degrees. For the remaining (e_i, f_i) that cannot be taken care of in this way, we will have to remove the vertices corresponding to e_i or f_i, whichever has the least number of vertices. This takes care of condition (3).

For conditions (1) or (2) of G_j, say condition (1), we consider each descendant $G_{j'}$ of G_j attached at the vertex u from the edge uv in G_j corresponding to the parent of G_j. In each $G_{j'}$, a path starting at u with edges $e_1, \ldots, e_a$ is subdivided into new graphs G_i corresponding to paths $e_1, \ldots, e_i$ for $1 \leq i \leq a$. Thus each edge e other than uv incident to u in G_j corresponds to a path of graphs $G_{e1}, \ldots, G_{ea}$ from the leaves going up to G_j, and we proceed from the leaf G_{e1} to connect to the highest G_{ei} and so on up the path, as from before, to reduce the number of G_{ei} on the path corresponding to e. Then we proceed to join the paths of corresponding to different e to one another and to the rest of G_j, by considering the bottom halves of such paths, again by counting the number of vertices that might be included from each G_{ei} by joining it to the rest of G_j. If G_j has n_j vertices not in the child G_j, then instead of attempting to join at least $(k/t')/(n/2^i)$ bottom halves per flow as before in sub-sub-phases, we consider joining at least $(n_j/n)(k/t')/(n_j/2^i)$ bottom halves per flow, or $(n_j/n)(k/t')/d$ vertices, so the degree goes up by at most $4t'n/k$, and the number of vertices not joined is at most $\sum_j (n_j/n)(k/t') \leq k/t$ over all G_j. Here this is incurred in $\log n$ sub-phases corresponding to the different values of $1 \leq 2^i \leq n_j$ for each G_j. This completes the cases for conditions (1),(2) and (3) and the construction of the almost 3-cyclable subgraph. The time complexity is dominated by pushing $O(n)$ units of flow in $O(m)$ time each, for a total $O(nm)$ time.

Applying Theorem 5 to the 2-connected subgraph meeting the sufficient conditions implied by 3-cyclability, having at least $k/2$ vertices, and degree at most $d' = O((n/k)\log^6 n)$ gives the bound $k^{1/(c \log d')} \geq k^{1/(c'(\log(n/k)+\log\log n))}$ on the length of the cycle found in $O(n^3)$ time, for constants $c, c' > 0$. □

The following generalization of Theorem 7 appears in [9].

Theorem 8 *If a graph G with n vertices has a 3-cyclable minor K with k vertices of vertex degree at most d, then for we can find in $O(n^3)$ time a cycle in G of length at least $k^{1/(c(\log d+\log(n/k)+\log\log n))}$ for some constant $c > 0$.*

4 Long Cycles in Graphs of Fixed Euler Genus

We now show how to find long cycles in graphs of fixed Euler genus

Theorem 9 *Let G be a 3-connected graph of fixed Euler genus g that has k special edges, so that every vertex of G is incident to at most two special edges. Then one can find in polynomial time a cycle in G that has at least $f'(g)k^{\log_3 2/(1+\log_2 3)}$ special edges, with $f'(g) > 0$. The algorithm runs in time $O(n^2)$ for planar graphs.*

Proof. Let G be a 3-connected graph. Say that an edge e in G is *contractable* if the graph obtained from G after contracting edge e is 3-connected, and *removable* if the graph obtained from G, after removing e and then replacing any resulting vertex v of degree 2 and incident edges uv, vw with a single edge uw, is 3-connected. A well known fact is that every edge of a 3-connected graph with at least five vertices is either contractable or removable.

Select a set K of at least $k/3$ special edges that do not share endpoints. Repeatedly contract or remove from G edges $e = (u, v)$ such that neither u nor v is in K, depending on whether e is contractable or removable. This results in a 3-connected graph G' such that the vertices v that are not incident to an edge in K have all their neighbors incident to an edge in K. Find in G' a cycle C of length at least $r = f(g)k^{\log_3 2}$ by the result from [4] in polynomial time, or in time $O(n^2)$ if G' is planar by the result from [7]. The cycle C has at least $r/2$ vertices incident to an edge in K, which can be decomposed as $2s+t \geq r/2$, where s is the number of edges of K with both endpoints in C, and t is the number of edges of K with exactly one endpoint in C. Let L be the set of these t edges.

Repeatedly contract or remove from G edges $e = uv$ such that neither u nor v is in C or incident to an edge in L, depending on whether e is contractable or removable. This results in a 3-connected graph G' such that the vertices v that are neither in C nor incident to an edge in L have all their neighbors either in C or incident to an edge in L. By the result from [19], G'' has a 2-connected spanning subgraph H of maximum degree $6 + 2g$. The vertices v not in C incident to an edge $e = uv$ in L can be joined by paths in H of length at most 2 not going through u to other vertices w that are either in L or in C. Each such path of length at most 2 meets at most cg^2 other such paths by the $6 + 2g$ degree bound, for some constant $c > 0$. Therefore there exist at least $t/(cg^2)$ disjoint such paths, which can be found in $O(n^2)$ time by flow.

We thus have $\max(s, t/(cg^2)) \geq r/(c'g^2)$ disjoint paths containing at least one special edge joining two endpoints in C for some constant $c' > 0$. We can then find as in Theorem 2 a cycle containing at least $c''(r/g^2)^{1/(1+\log_2 3)} = c''(f(g)k^{\log_3 2}/g^2)^{1/(1+\log_2 3)} = f'(g)k^{\log_3 2/(1+\log_2 3)}$ special edges with $0 < f'(g) \leq 1$. The complexity is $O(n^2)$ as 3-connected graphs can be recognized in linear time [12]. □

The argument that derives Theorem 4 from Theorem 3 carries over to deriving the following from Theorem 9.

Theorem 10 *Let G be a 3-connected graph of fixed Euler genus g where edge e_i has weight $w_i \geq 1$, and every vertex is incident to at most two edges of weight $w_i > 1$. Then one can find in polynomial time a cycle in G of total weight at least $f''(g)L_c(w) = f''(g)(\sum w_i^c)^{1/c}$ for some constant $c \geq 1$, with $f''(0) = 1$ and $0 < f''(g) \leq 1$. The algorithm runs in time $O(n^2)$ for planar graphs.*

We finally show the main result of this section.

Theorem 11 *Leg G be a graph of fixed Euler genus g that has a 3-cyclable minor H with k vertices (or in particular, a cycle with k vertices). Then one can find in polynomial time a cycle in G of length at least $f(g)k^{1/(2c)}$ for the constant $c \geq 1$ from Theorem 10 and $0 < f(g) \leq 1$. The algorithm runs in time $O(n^2)$ for planar graphs.*

Proof. The argument that derives Theorem 5 from Theorem 4 carries over to deriving the result from Theorem 10. This follows first by observing that a 3-cyclable minor can only lead to at most two special edges incident to a vertex in a 3-connected component G_i, so Theorem 10 applies. Second, the children G_j of the children of a 3-connected component G_i share at most one vertex with G_i, so the Euler genus of the union of G_i and G_j is at least the sum of the Euler genus of each of G_i and G_j. Therefore there are at most $c'g$ components G_i of genus $g_i \geq 1$ on a path from the root component G_0 to a leaf component G_i, so in going up to the root we multiply by $f''(g)$ from Theorem 10 at most $c'g$ times. This gives the bound $f(g)k^{1/(2c)}$ for $f(g) = f''(g)^{c'g}$. $\square$

References

[1] C. Bazgan, M. Santha, and Z. Tuza. "On the Approximability of Finding A(nother) Hamiltonian Cycle in Cubic Hamiltonian Graphs." *Journal of Algorithms* 31 (1999), pp. 249–268.

[2] A. Björklund and T. Husfeldt, "Finding a path of superlogarithmic length." *SIAM Journal on Computing* 32–6 (2003), pp. 1395–1402.

[3] A. Björklund, T. Husfeldt, and S. Khanna. "Approximating longest directed path." *ICALP* (2004).

[4] T. Böhme, B. Mojar, and C. Thomasen. "Long cycles in graphs on a fixed surface." *Journal of Combinatorial Theory, Series B* 85 (2002), pp. 338–347.

[5] G. Chen, Z. Gao, X. Yu, and W. Zang. "Approximating longest cycles in graphs with bounded degree." *SIAM Journal on Computing* 36 (2006), pp. 635–656.

[6] G. Chen, J. Xu, and X. Yu. "Circumference of graphs with bounded degree." *SIAM Journal on Computing* 33–5 (2004), pp. 1136–1170.

[7] G. Chen and X. Yu. "Long cycles in 3-connected graphs." *Journal of Combinatorial Theory, Series B* 86 (2002), pp. 80–99.

[8] T. Feder, R. Motwani, and C. Subi. "Approximating the longest cycle problem in sparse graphs." *SIAM Journal on Computing* 31–1 (2003), pp. 1596–1607.

[9] T. Feder, R. Motwani, and A. Zhu. "k-connected spanning subgraphs of low degree." Electornic Colloquium on Computational Complexity (ECCC) TR06-41 (2006).

[10] M. Fürer and B. Raghavachari, "Approximating the minimum degree spanning tree to within one from the optimal degree." In: *Proceedings of the Third Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 1992, pp. 317–324.

[11] H.N. Gabow. "Finding paths and cycles of superpolylogarithmic length." In: *Proceedings of the Thirty-Sixth Annual ACM Symposium on Theory of Computing (STOC)*, 2004, pp. 407–416.

[12] J.E. Hopcroft and R.E. Tarjan. "Dividing a graph into triconnected components." *SIAM Journal on Computing* 2–3 (1973), pp. 135–158.

[13] B. Jackson. "Longest cycles in 3-connected cubic graphs." *Journal of Combinatorial Theory, Series B* 41 (1986), pp. 17–26.

[14] B. Jackson and N.C. Wormald. "Longest cycles in 3-connected graphs of bounded maximum degree." In: **Graphs, Matrices, and Designs,** R.S. Rees (editor), Marcle and Dekker (1993), pp. 237–254.

[15] D. Karger, R. Motwani, and G.D.S. Ramkumar. "On approximating the longest path in a graph." *Algorithmica* 18 (1997), pp. 82–98.

[16] A.S. LaPaugh and R.L. Rivest. "The subgraph homeomorphism problem." *Journal of Computer and System Sciences* 20 (1980), pp. 133–149.

[17] R. Ravi, B. Raghavachari, and P. Klein. "Approximation through local optimality: designing networks with small degree." In *Proceedings of the 12th Conference on Foundations of Software Tech. and Theoret. Comp. Sci, Lect. Notes in Comp. Sci* 652 (1992) pp. 279–290.

[18] N. Robertson and P.D. Seymour. "Graph Minors XIII: The disjoint paths problem." *Journal of Combinatorial Theory, Series B* 63 (1995), pp. 65–110.

[19] D.P. Sanders and Y. Zhao. "On 2-connected spanning subgraphs with low maximum degree." *Journal of Combinatorial Theory, Series B* 74 (1986), pp. 64–86.