Finding Paths and Cycles of Superpolylogarithmic Length

Harold N. Gabow Department of Computer Science
University of Colorado at Boulder
Boulder, Colorado 80309
hal@cs.colorado.edu

ABSTRACT

Let ℓ be the number of edges in a longest cycle containing a given vertex v in an undirected graph. We show how to find a cycle through v of length $\exp(\Omega(\sqrt{\log \ell / \log\log \ell}))$ in 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 $\Omega((\log \ell)^2 / \log\log \ell)$ due to Björklund and Husfeldt. Our approach, which builds on Björklund and Husfeldt's, uses cycles to enlarge cycles. This self-reducibility allows the approximation method to be iterated.

Categories and Subject Descriptors

F.2 [**Theory of Computation**]: Analysis of Algorithms and Problem Complexity; G.2.2 [**Mathematics of Computing**]: Discrete Mathematics—*Graph Theory*

General Terms

Algorithms, Design, Theory

Keywords

approximation algorithms, graph algorithms, path, cycle

1. INTRODUCTION

Interest in approximating the longest path of a graph was rekindled by Karger, Motwani and Ramkumar [9] who were motivated by the large gap between known performance guarantees and hardness results. We make some progress in reducing the gap by presenting the best known polynomial time approximation algorithm. In particular we show how to find paths of greater than polylogarithmic length.

Previous work. Monien [11] investigated fixed parameter algorithms for long paths and cycles, showing how to find a path of length (exactly) k, if one exists, in time $O(k!nm)$. He also proved a fact about undirected cycle length that is useful in finding cycles of length $\geq k$; a variant of this fact is stated below. In other early work Fellows and Langston showed how to find undirected cycles of length $\geq k$ using Robertson-Seymour theory [6]. Bodlaender [2] used dynamic programming to find long undirected paths and cycles, improving Monien's bounds.

Karger et.al.[9] showed getting a constant factor approximation to the longest undirected path is NP-hard. Furthermore for any $\epsilon > 0$, approximating to within a factor $2^{O(\log^{1-\epsilon} n)}$ is quasi-NP-hard. Stronger hardness results for directed graphs are given in [4].

Alon et. al. [1] introduced the technique of color coding to find paths, cycles and other small subgraphs. For example color coding finds a path of length $\log n$ in polynomial time if one exists; if not it finds a longest path. The same holds for vw-paths.

Björklund and Husfeldt [3] find an undirected path of length $\Omega((\log \ell / \log\log \ell)^2)$ in polynomial time, where ℓ is the longest path length. [7] observes that the length guarantee is actually $\Omega(\log^2 \ell / \log\log \ell)$. This is the best known bound to date.

Gabow and Nie [7] investigate long directed cycles. They also prove a variant of Monien's undirected cycle structure theorem: If a connected undirected graph has a cycle of $\geq k$ edges, either every dfs tree has a fundamental cycle of $\geq k$ edges or some cycle has between k and $2k$ edges. This result, or Monien's, allows color coding to be applied to the problem of finding a cycle of $\geq k$ edges.

Our algorithm requires finding a cycle through three given points. Robertson and Seymour showed that more generally, the fixed vertex subgraph homeomorphism problem can be solved in polynomial time [12]. However huge constants are involved and even for our case of triangles the exact algorithm is unknown. LaPaugh and Rivest [10] give a linear time algorithm for our case of triangles, involving no large hidden constants.

Our contribution. A *v-cycle* is a cycle containing vertex v. We take the basic problem to be approximating the longest v-cycle, where v is a given vertex of degree 2. It is easy to reduce longest cycle, longest vw-path and longest path to this problem.

Let ℓ denote the length of a longest v-cycle in a given undirected graph. We show that a v-cycle whose length is $\exp(\Omega(\sqrt{\log \ell / \log\log \ell}))$ can be found in polynomial time. This implies the same bound for the longest cycle, longest vw-path and longest path. This improves the previous best bound of Björklund and Husfeldt for longest path given

STOC'04, June 13–15, 2004, Chicago, Illinois, USA.

above. It also improves the previous best bound for longest vw-paths and cycles, which was only $\log n$ using color coding.

Our approach builds on Björklund and Husfeldt's idea of using cycles to enlarge paths. We use cycles to enlarge cycles, giving a self-reducibility property that allows the construction to be iterated. We note that the hardness results of Karger et.al.[9] are based on a self-improvability property of longest path involving graph products; it is unclear if this has any relation to our self-reducibility.

Our results hinge on properties of biconnected components, cutpoints and separating pairs. The self-reducibility gives rise to a family of recursive algorithms. We can only recur a limited number of times because of the need to keep the graphs large.

Section 2 presents the facts on cutpoints and separation pairs that underlie our algorithm. Section 3 presents the algorithm. The final two sections give the analysis, Section 4 proving the length guarantee and Section 5 giving implementation details and proving the polynomial time bound. We conclude this section with our terminology.

Terminology. A fraction a/bc is always an abbreviation of $a/(bc)$, e.g., $a/2k$. All logarithms are base 2 unless noted otherwise. When used as a number, e is the base of natural logarithms. $\exp(x)$ denotes e^x.

Our graph terminology is consistent with [14]. All graphs in this paper are undirected and simple. $G[X]$ denotes the subgraph induced by vertex set X. For X and Y disjoint vertex sets, $E[X,Y]$ consists of all edges joining X and Y. Furthermore writing $xy \in E[X,Y]$ means $x \in X$ and $y \in Y$.

All paths and cycles in this paper are simple. In contrast a walk can have repeated vertices. (So a path is a simple walk.) Let x and y be vertices. An *xy-path* is a path from x to y. For $x \neq y$, $d(x,y)$ denotes the length of a shortest xy-path. If the graph of interest is unclear in some notation we include it as a subscript, e.g., $d_G(x,y)$.

We represent a path as a list of vertices, e.g., x, y, z. We also allow paths in the list, e.g., if xy is an edge and Z is a path starting at y then x, Z denotes a path that is 1 edge longer than Z. Sometimes we write x, y, Z for the same path to remind the reader of the first edge xy. This will not cause any confusion.

If P is a path containing x and y with x preceding y, then $P[x,y]$ denotes the subpath of P from x to y. We occasionally write $P[y,x]$ to refer to the subpath from y to x in the reverse path of P; however to prevent confusion we always indicate when this extended notation is being used. If C is a cycle containing x, y and v then $C_v[x,y]$ ($C_{\overline{v}}[x,y]$) denotes the subpath of C from x to y that contains (avoids) v, respectively. We extend the subpath notation to allow open-sided intervals, e.g., $P(v,w]$ is the path $P[v,w] - v$.

If P is a path we use P to denote a set of vertices or edges, as is convenient; the exact meaning will be clear from context. $|P|$ always denotes the number of edges in P.

2. APPROACH

Throughout this paper G is a given connected undirected graph. Our main algorithm finds a long v-cycle, where v is a given vertex of degree 2. This algorithm can be used to find a long xy-path, by adding a new vertex v with edges vx, vy. Letting x and y vary we can approximate the longest path in the graph. Similarly we can approximate the longest cycle.

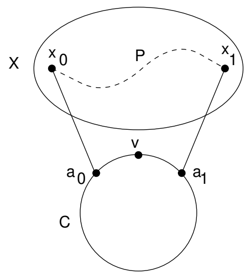

Figure 1: A v-cycle C and an a_0a_1-path P passing through a connected component X of $G - V(C)$.

The overall approach is due to Björklund and Husfeldt [3]. They use a cycle to find a long path. Since our algorithm seeks a long cycle rather than a long path, the approach must be modified. This section presents the principles on which our algorithm is based.

The setting for our algorithm is illustrated in Fig.1: C is a v-cycle. X is a connected component of $G - V(C)$. We sometimes write $G[X]$ to emphasize that we are dealing with the graph induced by vertices X. P is an a_0a_1-path through X, more precisely for distinct vertices $a_0, a_1 \in C$, $P = a_0, P[x_0, x_1], a_1$ with $P[x_0, x_1] \subseteq X$.

Here is how the figure relates to the algorithm of the next section. Let C^* be a longest v-cycle. C will be a v-cycle already found by the algorithm. The algorithm is attempting to enlarge C to a longer v-cycle. P will be a path that our algorithm has found recursively (in Lemma 2.1(i) below) or a subpath of C^* (in Lemma 2.1(ii)).

We use two main techniques to enlarge C. The first is similar to Björklund and Husfeldt's idea to use cycles to extend paths:

LEMMA 2.1. *(i) $P, C_v[a_1, a_0]$ is a v-cycle of length $> |P| + d_C(a_0, v)$.*

(ii) For any vertex $c \in C$ adjacent to X, there is a path $Q \subseteq X$ and a vertex $c' \in C - c$ such that c, Q, c' is a path of length $\geq |P|/2 + 1$.

PROOF. (i) Note $|C_v[a_1, a_0]| > d_C(a_0, v)$ since a_1 is distinct from the degree 2 vertex v.

(ii) Choose $x \in X$ a neighbor of c as follows. If $c = a_i$ for $i \in \{0,1\}$ then $x = x_i$. Otherwise x is arbitrary.

Take any minimal path R from x to P in X. Let R end at vertex $r \in P$. Choose index $j \in \{0,1\}$ so $|P[r, x_j]| \geq |P[r, x_{1-j}]|$. (Note one of these two subpaths of P actually involves the reverse of P.) The desired path c, Q, c' is $c, R, P[r, x_j], a_j$. This walk is simple since the choice of x guarantees $c \neq a_j$. The path's length is $\geq 1 + (|P| - 2)/2 + 1 = |P|/2 + 1$. ∎

When $d_C(a_0, v)$ is large part (i) allows us to make good progress in enlarging C. When $d_C(a_0, v)$ is small another method is needed. The idea, illustrated in Fig.2, is to take 2 recursively found paths (P in Fig.2) and find a v-cycle that contains both paths. Of course 2 arbitrary paths need not be contained in a v-cycle. To guarantee this v-cycle exists, for each component X we find a separation pair of G (r_0, r_1 in Fig.2) both vertices of which are contained in C^*. Then no matter what r_0r_1-paths the algorithm chooses, they

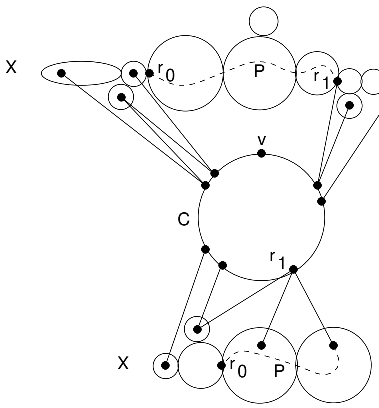

Figure 2: A v-cycle C and 2 connected components X of $G - V(C)$, with their separation pairs r_0, r_1 and r_0r_1-paths P.

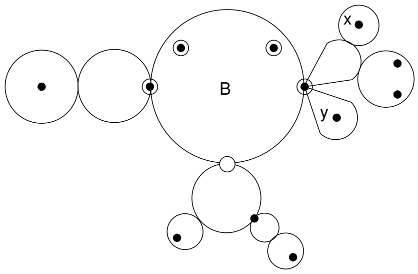

Figure 3: A bicomponent B and 12 vertices (drawn solid) that determine 5 distinct vertices v_B (circled or drawn hollow).

will be contained in a v-cycle (gotten from C^*). Of course the algorithm does not know C^*, so some guessing will be involved, and the separation pairs need not even exist, so some other cases will be involved.

The next several lemmas give tools that allow us to either enlarge C or to find these separation pairs r_0, r_1. Our discussion uses connected components, biconnected components, and a form of triconnected components. For clarity we use terminology that explicitly differentiates all these types of components.

A *bicomponent* is the set of edges of a biconnected component. Recall that two edges are in the same bicomponent iff some cycle contains both of them [13]. We are interested in when a bicomponent B separates 2 vertices. By definition the edge set B separates vertices x and y if every xy-path includes an edge of B. (Note that it is possible for two vertices to be separated by a vertex of $V(B)$ but not separated by B, e.g., x and y in Fig.3.)

The following notation, illustrated in Fig.3, elucidates the concept: For any vertex v and bicomponent B, v_B denotes the vertex of $V(B)$ that is the end of every minimal path from v to $V(B)$. For example a vertex $v \in V(B)$ has $v_B = v$. Assuming the graph is connected, any v_B is unique. In proof suppose not. So there are two minimal paths from v to $V(B)$, say P_1 and P_2, with P_i ending in vertex $b_i \in V(B)$ and $b_1 \neq b_2$. Let Q be a b_1b_2-path in B. Then $P_1 \cup Q \cup P_2$ contains a cycle that includes both edges of B and edges not in B. This contradicts the definition of B.

The vertices v_B give this alternate characterization of separation: v and w are separated by B if and only if $v_B \neq w_B$. The only if direction is clear. For the if direction note that a vw-path avoiding B would contradict the uniqueness of v_B.

We now give some basic properties of separation by a bicomponent. Say that vertex s *weakly separates* sets $A, B \subseteq V$ if every path from A to B contains s. (It is possible that s belongs to A or B.)

LEMMA 2.2. *Let G be connected, and B be a bicomponent.*
(i) Any two distinct vertices in $V(B)$ are separated by B.
(ii) Any two distinct vertices x, y are separated by B if some xy-path contains an edge of B.
(iii) For any set of vertices W, either B separates two vertices of W or some vertex of $V(B)$ weakly separates W and $V(B)$.

PROOF. (i) and (ii) follow immediately from the v_B characterization of separation. For (iii) note that if B does not separate any two vertices of W then every $w \in W$ has the same vertex w_B. By definition this vertex weakly separates W and $V(B)$. ∎

For any real value $b > 2$, a bicomponent is *b-round* if it contains a cycle of $\geq b$ edges. $D(x, y)$ denotes the length of a longest xy-path.

LEMMA 2.3. *Consider a connected graph and two distinct vertices x, y.*
(i) If x and y are separated by a b-round bicomponent then $D(x, y) \geq b/2$.
(ii) $D(x, y) > d(x, y)$ implies x and y are separated by a $(D(x, y)/d(x, y) + 1)$-round bicomponent.

Remark: We will only use this weaker version of (ii): If $b = D(x, y)/d(x, y) > 2$ then x and y are separated by a b-round bicomponent.

PROOF. (i) Let B be a bicomponent containing a cycle A of length $\geq b$. First suppose x and y are distinct vertices of $V(B)$. Biconnectedness implies there are distinct vertices $r, s \in A$ with an xr-path and a ys-path that are disjoint and also both disjoint from A until their last vertex. (This fact is well-known [14, Ex.4.2.9]. It is also easy to see from the definition, by adding edge xy to the graph if it is not already present.) Piece these paths together to form an xy-path of length $\geq b/2$. (Specifically, follow the xr-path, then a path $A[r, s]$ of length $\geq b/2$, and then the ys-path reversed.)

Now suppose x and y are arbitrary vertices separated by B. Take an xy-path. It contains an xx_B-subpath and a yy_B-subpath. We have $x_B \neq y_B$, so the subpaths are disjoint. Form the desired xy-path from the two subpaths plus the x_By_B-path of length $\geq b/2$ constructed above (note this path is contained in B).

(ii) Let L (S) be a longest (shortest) xy-path. Let the vertices of $L \cap S$ be $x = s_0, s_1, \ldots, s_r = y$, ordered as they occur in S. (This need not be their order in L.) For each consecutive pair s_i, s_{i+1}, paths $L[s_i, s_{i+1}]$ and $S[s_i, s_{i+1}]$ are internally vertex disjoint. ($L[s_i, s_{i+1}]$ may actually be a subpath of the reverse of L.) The various paths $L[s_i, s_{i+1}]$ can share vertices and edges, but certainly some $L[s_j, s_{j+1}]$ has length $\geq |L|/r \geq |L|/|S|$. Thus $L[s_j, s_{j+1}], S[s_{j+1}, s_j]$ is a cycle A.

(A is simple, and its length is $\geq 1 + \lceil |L|/|S| \rceil \geq 3$.) A is contained in some bicomponent B. A makes B $(|L|/|S|+1)$-round. Since S contains an edge of B, B separates x and y. ∎

Recall that two edges are *independent* if all four endpoints are distinct. A set of edges forms a *star* if some vertex (its *center*) is incident to all of them. For an edge $e \in E[X,Y]$, $X(e)$ denotes the endpoint of e in X. For $F \subseteq E[X,Y]$, $X(F)$ denotes $\{X(e) : e \in F\}$.

LEMMA 2.4. *Let the vertex set of graph G be partitioned into sets X and Y, with $G[X]$ connected. Let B be a bicomponent of $G[X]$ and let $F = E[X,Y]$.*

Suppose no two independent edges $e_1, e_2 \in F$ have their ends $X(e_1), X(e_2)$ separated by B, in graph $G[X]$. Then either F is a star or some vertex of $V(B)$ weakly separates $X(F)$ from $V(B)$, in graph $G[X]$.

PROOF. Suppose F is not a star. Then there are independent edges $yx, y'x' \in E[Y,X]$. The lemma's hypothesis implies $x_B = x'_B$. (All vertices v_B in this argument are calculated in graph $G[X]$.) Consider any edge $wz \in E[Y,X]$ with $z \neq x, x'$. This edge is independent with either yx or $y'x'$ (or both). Hence the lemma's hypothesis implies $z_B = x_B$. Thus $X(F)$ is weakly separated from $V(B)$ by x_B in $G[X]$. ∎

The next two lemmas are used by the algorithm to find separation pairs.

Consider graph G with vertex set partitioned into X and Y. For any real value $b > 2$ a set of edges $F \subseteq E[X,Y]$ is *b-close* if no two independent edges $e_1, e_2 \in F$ have their ends $X(e_1), X(e_2)$ separated by a b-round bicomponent in $G[X]$.

LEMMA 2.5. *Let the vertex set of graph G be partitioned into sets X and Y, with $G[X]$ connected. Let $F = E[X,Y]$ be b-close for some $b > 2$. Let P be an x_0x_1-path in $G[X]$, with $x_0 \in X(F)$.*

Suppose $|P| \geq d_X(x_0,x_1)b > 0$. Then either F is a star or some vertex $r \in P$ weakly separates $X(F)$ from $V(P(r,x_1])$ in graph $G[X]$, and furthermore $|P[x_0,r]| < d_X(x_0,x_1)b$.

Remark: The lemma is illustrated 4 times in Fig.2. First discard all edges incident to C except the 4 in the upper left. These 4 edges constitute set F. Extend P of the figure to the left so it begins at one of the 3 vertices of $X(F)$. Now r_0 is vertex r of the lemma. Similarly for r_0 in the lower left. In the upper right r_1 is r of the lemma, this time a weak separator. The lower right illustrates the case F a star.

PROOF. Suppose F is not a star. Let x'_1 be the first vertex in $P(x_0,x_1]$ having $|P[x_0,x'_1]| \geq d_X(x_0,x'_1)b$. Lemma 2.3(*ii*) then shows x_0 and x'_1 are separated (in $G[X]$) by a b-round bicomponent B. The hypothesis of Lemma 2.4 holds by b-roundness of B and b-closeness. So it shows some vertex $r \in V(B)$ weakly separates $X(F)$ from $V(B)$, in graph $G[X]$. Clearly $r \in P - x'_1$ ($P[x_0,x'_1]$ contains an edge of B). The choice of x'_1 shows $|P[x_0,r]| < d_X(x_0,r)b$.

Let r' be the vertex following r in P. Since P contains an edge of B, $rr' \in B$. The path $P[r',x_1]$ avoids r. So r weakly separates $X(F)$ from $V(P[r',x_1])$. In particular r separates x_0 and x_1, so $d_X(x_0,r) \leq d_X(x_0,x_1)$. This implies $|P[x_0,r]| < d_X(x_0,x_1)b$. ∎

For a set F as in Lemma 2.5, define r to be the *separating vertex for F* if either

(i) F is a star and r is its center, or

(ii) r is the vertex given by Lemma 2.5.

If more than one vertex qualifies as r (this can happen in both cases) the choice is arbitrary. Note also that if F is a star with center $r \in X$ then r trivially has all the properties of the separating vertex of Lemma 2.5.

Recall that a *separation pair* is a set of two vertices that separates two other vertices.

LEMMA 2.6. *Let G be a biconnected graph whose vertex set is partitioned into sets X and Y, with $G[X]$ connected. Let $E[X,Y] = F_0 \cup F_1$ with $Y(E[X,Y]) \subset Y$. For some $b > 2$ let both sets F_i be b-close. Let x_i, $i = 0,1$ be distinct vertices with $x_i \in X(F_i)$. Let P be an x_0x_1-path in $G[X]$.*

Suppose $|P| \geq 2d_X(x_0,x_1)b + 2$. Let r_i be the separating vertex for F_i. Then r_0, r_1 is a separation pair of G.

PROOF. First note that Lemma 2.5 applies to both sets F_i, so the separating vertices r_i exist.

We first enlarge P to a path that includes both vertices r_i: If F_0 is a star centered at $r_0 \in Y$ then enlarge P to the path r_0, P. Do the same for F_1 at the x_1 end. The resulting path, which we continue to call P, includes r_0, r_1 in all cases. (Note that if both $r_i \in Y$ we have $r_0 \neq r_1$, by biconnectivity. So P is indeed a path.)

Take a vertex $x \in P(r_0,r_1)$. We show that x exists: First recall $r_i \in X$ implies $|P[x_i,r_i]| < d_X(x_0,x_1)b$. If both $r_i \in X$ this inequality plus the assumed bound on $|P|$ implies r_0 precedes r_1 in P and $V(P(r_0,r_1)) \neq \emptyset$. If some $r_i \in Y$ then we can take $x = x_i$; we have $x_i \neq r_{1-i}$ again by the length bounds.

We show r_0, r_1 separates x and $Y - Y(E[X,Y])$. Let Q be a minimal path from $Y - Y(E[X,Y])$ to x. Q starts with an edge $yz \in E[Y,X]$. For definiteness let $yz \in F_0$. If F_0 is a star then clearly Q contains its center r_0. Suppose F_0 is not a star. Lemma 2.5 shows r_0 weakly separates z and $x \in P(r_0,x_1]$ in $G[X]$. Minimality implies $Q[z,x]$ is contained in X. Hence $r_0 \in V(Q)$. ∎

Here is how the algorithm uses separation pairs (recall Fig.2). Say that a path *traverses* a subgraph H if it contains a subpath of ≥ 3 edges, where the first and last subpath edge do not belong to H but all others do. Let a, b be a separation pair. An *a,b-tricomponent* T is a maximal set of edges, any two of which are joined by a path that does not contain a or b internally [8]. It is easy to see that any path P traversing an a,b-tricomponent T contains both a and b. Furthermore P traverses T only once, i.e., $E(P) \cap T = E(P[a,b])$.

Let C^* be a v-cycle. Let r_0, r_1 be a separation pair of G contained in C^*, and abbreviate $C^*_{\overline{v}}[r_0,r_1]$ to C^*_1. Let T_1 be the r_0,r_1-tricomponent of G that contains C^*_1. Let P_1 be an arbitrary r_0r_1-path contained in T_1. Next define C^*_2, T_2 and P_2 similarly from a separation pair r'_0, r'_1 contained in C^*. Assume C^*_1 and C^*_2 are vertex disjoint.

The *Gluing Principle* states that regardless of the choice of P_i, a v-cycle containing paths P_1 and P_2 exists. In proof, the edge set $A = (C^* - \cup_{i=1}^2 C^*_i) \cup \cup_{i=1}^2 P_i$ is such a cycle. This hinges on the fact that A is guaranteed to be simple, because C^* traverses each tricomponent T_1, T_2 only once.

The Gluing Principle forms a major strategy of the algorithm.

3. ALGORITHM

This section presents an algorithm to approximate the longest cycle C^* through a given vertex v of degree 2 in a given graph G. Section 4 proves the length guarantee of the algorithm. Section 5 gives some final details of the algorithm and proves the polynomial time bound.

Let ℓ be the length of C^*. For every integral power $p \geq 1$ we give an algorithm $\mathcal{A}_p$ that finds a cycle of length $\Omega((\log \ell / \log\log \ell)^p)$. $\mathcal{A}_p$ uses $\mathcal{A}_{p-1}$ as a subroutine, and the hidden constant depends on p. We choose an appropriate value p^* of p to get the desired overall result on superpoly-logarithmic cycles.

For brevity the presentation of $\mathcal{A}_p$ is not optimized. Slight asymptotic improvements are possible by using more detailed versions of $\mathcal{A}_1$ and $\mathcal{A}_2$. Also we make no attempt to keep the constants small, preferring instead to keep the arithmetic simple.

We will guess a value ℓ^* as the length of C^*. Write

$$a^* = \log \ell^*, \quad k^* = \log a^* = \log\log \ell^*.$$

For $p \leq p^*$, algorithm $\mathcal{A}_p$ will be given a smaller graph derived from the given one, along with a degree 2 vertex v. When describing $\mathcal{A}_p$ we will use C^* to denote the longest cycle through v in this recursive call. $\mathcal{A}_p$ uses variables a and k that play roles similar to a^* and k^* above, where

$$a = \frac{pa^*}{p^*}, \quad k = k^*.$$

The values of a decrease slowly with p. This allows us to find the longest cycle possible. The reader should think of variable k as $\log a$, in analogy with the definition of k^*. However since $\log a$ is difficult to compute, we actually define k to be the slightly larger value k^*. Algorithm $\mathcal{A}_p$ has this guarantee: If $|C^*| \geq 2^a$ then $\mathcal{A}_p$ returns a cycle of length at least

$$\alpha_p \left(\frac{a}{k}\right)^p.$$

Here $\alpha_p \leq 1$ is a factor depending on p that we will derive.

We start with a driving routine. It ensures that a is large, in all calls to routines $\mathcal{A}_{p^*}, \mathcal{A}_{p^*-1}, \ldots, \mathcal{A}_1$.

Main Routine

Step 1. If $|C^*| \leq 2^{2^8}$ use color coding to find a longest v-cycle, and return it.

Step 2. For k^* taking on consecutive integral values from 8 to $\lfloor \log\log n \rfloor$ set $a^* = 2^{k^*}$, $p^* = \lfloor \sqrt{a^*/24k^*} \rfloor$ and call $\mathcal{A}_{p^*}$. Return the longest cycle found.

The test on $|C^*|$ in Step 1 is implemented using the gap theorem of [11] or [7].

Algorithm $\mathcal{A}_1$ can be based on color coding [1] and again the gap theorem. Using these it is easy to find a v-cycle of length $\min\{|C^*|, \log n\}$ in polynomial time (for details see [7]). This implies our length guarantee for $\mathcal{A}_1$ is satisfied with $\alpha_1 = 1$, since clearly we can assume $a \leq \log n$.

For $p^* > p \geq 1$ algorithm $\mathcal{A}_{p+1}$ uses $\mathcal{A}_p$. We give an overview of $\mathcal{A}_{p+1}$ before stating it precisely. $\mathcal{A}_{p+1}$ begins by using $\mathcal{A}_p$ to find a v-cycle C. Then $\mathcal{A}_{p+1}$ recurses on each connected component X of $G - V(C)$ (Fig.1). The idea is to enlarge one of these recursively found cycles, using C or another recursively found cycle, to get a longer cycle $\overline{C}$. Specifically each level of recursion in $\mathcal{A}_{p+1}$ enlarges the cycle of the previous level by the additive "increment" I defined by

$$\widetilde{a} = \frac{pa}{p+1}, \quad I = \frac{\alpha_p}{2}\left(\frac{\widetilde{a}}{k}\right)^p.$$

Observe (from the definition of a) that $\widetilde{a}$ is the value of a in recursive calls from $\mathcal{A}_{p+1}$ to $\mathcal{A}_p$. The enlarged v-cycle $\overline{C}$ gets returned by $\mathcal{A}_{p+1}$.

$\mathcal{A}_{p+1}$ uses several different strategies to construct $\overline{C}$. The first strategy (Step 3 below) is the analog of Björklund and Husfeldt's algorithm [3] (their algorithm also provides the organization described in the previous paragraph): We apply Lemma 2.1(i), using the recursive call to find a long path c, Q, c' through X and combining this path with C to get $\overline{C}$. This method is effective if $d_C(c, v)$ is large.

The second major strategy involves piecing together two recursively found paths into $\overline{C}$, as in Fig.2. This is the most difficult case to implement within polynomial time. The algorithm can only make a limited number of recursive calls, and we cannot identify beforehand the case that will actually yield $\overline{C}$. This forces us to organize all the cases like this case, as follows.

Partition the edges of C^* into maximal subpaths that are internally disjoint from C. Call each of these subpaths a *segment* of C^*. Both edges incident to v are segments. In general a segment is either an edge or chord of C or a path traversing a connected component X of $G - V(C)$ (i.e., a path through X plus two connecting edges). Note that more than one segment may traverse a given component X.

We shall apply Lemma 2.6 to a segment S and its connected component X. This means that in the lemma we define Y as the complement of X, and P as $S[x_0, x_1]$ for x_0 and x_1 the second and penultimate vertices of S respectively. (The F_i and b are defined below.)

Consider two (long) segments of C^* and their corresponding connected components X. We will apply Lemma 2.6 (to X and the segment) to find a separation pair r_0, r_1 and corresponding r_0r_1-tricomponent T in X that contains a (long) portion of C^*. We will use recursive calls to approximate $C^*[r_0, r_1]$. To be precise for $i = 1, 2$ let X_i be the two components X, let C^*_i be the subpath $C^*[r_0, r_1]$ in X_i, and let P_i be the r_0r_1-path returned by the recursive call (on the identified r_0r_1-tricomponent T). The Gluing Principle guarantees the existence of a cycle containing P_1, P_2 and v. The algorithm can find such a cycle using a routine for subgraph homeomorphism. This is done in Step 6.

We find the separation pairs r_0, r_1 using Lemma 2.6. But first we must ensure the lemma's hypothesis holds, i.e., F_0 and F_1 are b-close. This is done using Lemma 2.3(i) (in Step 2).

We now give a high-level statement of the algorithm. Some lower-level details (including how the various numerical quantities are computed) are given in Section 5.

$\mathcal{A}_{p+1}$ has parameters G the graph, v the vertex of degree 2, and ρ the recursion level. Parameter ρ is used to prevent too many levels of recursion (which might violate the time bound). ρ equals 0 in the initial call.

Write

$$b = 2^{\widetilde{a}+1}, \quad g = \left(\frac{\widetilde{a}}{k}\right)^{p+1}.$$

The algorithm generates a number of v-cycles. It maintains

$\overline{C}$ as the longest cycle generated, and eventually returns $\overline{C}$. If a cycle of length $\geq g$ is ever generated, that cycle is immediately returned.

Algorithm $\mathcal{A}_{p+1}(G, v, \rho)$

Step 0. Initialize $\overline{C}$ to any v-cycle. If $\rho \geq a^{p+1}$ then return $\overline{C}$. Otherwise prune G to the biconnected component containing v. Let w_0, w_1 be the two neighbors of v in G. So $W = \{vw_0, vw_1\}$ is a w_0w_1-tricomponent. For every other w_0w_1-tricomponent U, execute all the steps below for the graph G' whose edge set is $U \cup W$. Then return $\overline{C}$.

Step 1. Find a v-cycle C by calling $\mathcal{A}_p(G', v, 0)$.

Step 2. Repeat Step 2.1 until either it makes $|C| \geq g$ or no further enlargement of C is possible. In the former case return C.

Step 2.1. Suppose for some connected component X of $G' - V(C)$, $E[C, X]$ contains independent edges $cx, c'x'$ such that $|C_{\overline{v}}[c, c']| < I$ and some b-round bicomponent of $G[X]$ separates x and x'. Call $\mathcal{A}_p$ to find an xx'-path Q. (Specifically create a new vertex v' along with new edges $v'x, v'x'$, and call $\mathcal{A}_p(H, v', 0)$ for H the graph induced by $V[X] \cup \{v'\}$.) In C replace $C_{\overline{v}}[c, c']$ by c, Q, c'. (It will be shown that this replacement enlarges C.)

In the rest of the algorithm the variable X ranges over all the connected components of $G' - V(C)$ (if any).

Step 3. Execute Step 3.1 for each component X that has a neighbor $c \in C$ with $d_C(c, v) \geq I + 1$ (if X has more than one such neighbor choose c arbitrarily):

Step 3.1. Create 2 new vertices v', c' along with new edges $v'c, v'c'$, and $c'x$ for each vertex $x \in X$ that is a neighbor of $C - c$. Call $\mathcal{A}_{p+1}(H, v', \rho + 1)$ recursively for H the graph induced by $V[X] \cup \{c, c', v'\}$. The cycle returned corresponds to a path c, Q, c'' in G', where $Q \subseteq X$ and $c'' \in C - c$. Update $\overline{C}$ for the v-cycle $c, Q, c'', C_v[c'', c]$ of G.

Step 4. For each component X, check if $E[C, X]$ contains independent edges $cx, c'x'$ with $d_X(x, x') \geq g$. If so let P be an $x'x$-path in X and return the cycle $P, C_v[c, c'], x'$.

Step 5. This step applies Lemma 2.6. Initialize a set of paths $\mathcal{P}$ to $\emptyset$. For each component X not processed in Step 3 (i.e., all neighbors of X are within distance $I + 1$ of v) execute Steps 5.1–5.3:

Step 5.1. Partition $E[C, X]$ into sets F_0 and F_1, where F_0 contains the edges $cx \in E[C, X]$ with $d_C(c, w_0) \leq d_C(c, w_1)$ and F_1 contains the remaining edges.

Step 5.2. Form a collection $\mathcal{T}$ of triplets (r_0, r_1, T), where $r_0, r_1 \in X \cup C$, $T \subseteq E(X) \cup E[C, X]$, r_0, r_1 is a separation pair of G' with T an r_0r_1-tricomponent, and all sets T are pairwise disjoint. It is required that $\mathcal{T}$ contain all triplets given by applying Lemma 2.6 to the segments of C^*. More precisely for any segment of C^* traversing X where Lemma 2.6 (applied to component X and segment C^*) guarantees a separation pair r_0, r_1, $\mathcal{T}$ must include (r_0, r_1, T) where T is the r_0r_1-tricomponent in $X \cup E[C, X]$ containing $C^*[r_0, r_1]$. $\mathcal{T}$ may include other triplets besides these.

Step 5.3. For each $(r_0, r_1, T) \in \mathcal{T}$, call $\mathcal{A}_{p+1}$ to find an r_0r_1-path $P[r_0, r_1]$. Specifically create a new vertex v' along with new edges $v'r_0, v'r_1$, and call $\mathcal{A}_{p+1}(H, v', \rho + 1)$ recursively for H the graph induced by $V[T] \cup \{v'\}$. Add $P[r_0, r_1]$ to $\mathcal{P}$.

Step 6. For every pair of paths in $\mathcal{P}$, search for a cycle containing v and the two paths. If such a cycle is found update $\overline{C}$.

Step 7. Let $P[r_0, r_1]$ be the longest path of $\mathcal{P}$, with corresponding sets X, F_0, F_1. For $i = 0, 1$ if $r_i \in C$ let $c_i = r_i$; otherwise choose edge $c_ix_i \in F_i$, where $c_i \in C$ and if possible c_i is not a neighbor of v. Extend $P[r_0, r_1]$ to a c_0c_1-path $P[c_0, c_1]$ contained in $X \cup \{c_0, c_1\}$. Update $\overline{C}$ for the cycle $P[c_0, c_1], C_v[c_1, c_0]$.

We leave the following implementation details to Section 5: checking for the existence of the separating b-round bicomponent in Step 2.1, forming $\mathcal{T}$ in Step 5.2, finding the desired cycle in Step 6, and computing all the numeric quantities like I, b, etc. The rest of the implementation of the algorithm is clear.

4. LENGTH ANALYSIS

We first establish some inequalities that result from the basic parameters being large. Step 1 of the Main Routine ensures that in Step 2,

$$\ell^* \geq 2^{2^8}, \quad a^* \geq 2^8, \quad k^* \geq 8.$$

Recall that algorithm $\mathcal{A}_p$ is called for all $1 \leq p \leq p^*$, and $\mathcal{A}_p$ uses the value $a = pa^*/p^*$. We claim $\mathcal{A}_p$ always has

$$a \geq 2^4, \quad a \geq 24kp^2. \tag{1}$$

For the first inequality note that $p^* \leq \sqrt{a^*}$. Hence $a \geq a^*/p^* \geq \sqrt{a^*} \geq 2^4$. For the second inequality write $a = pp^*a^*/(p^*)^2$. Since $p^* \leq \sqrt{a^*/24k^*}$ we get $a \geq pp^*a^*/(a^*/24k^*) = 24k^*pp^*$.

We need to verify two inequalities to ensure the algorithm makes sense: $p^* \geq 1$ (since Step 2 calls $\mathcal{A}_{p^*}$) and $b > 2$ (since Step 2.1 looks for b-round components). The first inequality is equivalent to $\sqrt{a^*/24k^*} \geq 1$, i.e., $a^*/k^* \geq 24$. This holds since, remembering $x/\log x$ is an increasing function for $x \geq e$, we have $a^*/k^* = a^*/\log a^* \geq 2^8/8 = 2^5$. The inequality $b > 2$ holds since $b = 2^{\widetilde{a}+1}$, $\widetilde{a} \geq 2^4$.

Define the sequence α_p, $p \geq 1$ by

$$\alpha_p = \frac{1}{24^{p-1}(p!)^2}.$$

We show this implies that in every algorithm $\mathcal{A}_{p+1}$, $p \geq 1$,

$$I \geq 2. \tag{2}$$

By definition $I = \frac{\alpha_p}{2}\left(\frac{\widetilde{a}}{k}\right)^p$. Since $\widetilde{a}$ represents the quantity a in algorithm $\mathcal{A}_p$, we can apply (1) to $\mathcal{A}_p$ to get $\widetilde{a} \geq 24kp^2$. Thus

$$I \geq \frac{1}{2 \cdot 24^{p-1}p^{2p}}\left(\frac{24kp^2}{k}\right)^p = 12 > 2.$$

The main task of this section is to prove the following length guarantee. In the lemma and throughout the proof, a refers to its value in algorithm $\mathcal{A}_{p+1}$ (specifically $a = (p+1)a^*/p^*$).

LEMMA 4.1. *For any $p^* > p \geq 0$, if algorithm $\mathcal{A}_{p+1}$ is called with a graph having a v-cycle of length $\geq 2^a$ then it returns a v-cycle of length $\geq \alpha_{p+1}(a/k)^{p+1}$.*

PROOF. We induct on p. The base case $p = 0$ (i.e., algorithm $\mathcal{A}_1$) has already been verified. So assume $p \geq 1$ and algorithm $\mathcal{A}_p$ fulfills the length guarantee. We prove $\mathcal{A}_{p+1}$ also fulfills the length guarantee.

We begin by showing algorithm $\mathcal{A}_{p+1}$ is well-defined. By this we mean the remark in Step 2.1 is true:

Claim 0: *Every replacement done by Step 2.1 gives a longer cycle C.*

Proof: Applying Lemma 2.3(i) to the vertices x and x' of Step 2.1 shows $D_X(x, x') \geq b/2 = 2^{\widetilde{a}}$. So the inductive assumption applies to $\mathcal{A}_p$ and shows it gives an xx'-path of length $\geq \alpha_p \left(\frac{\widetilde{a}}{k}\right)^p - 2 = 2I - 2$. Hence the length of C increases by $> (2I - 2) + 2 - I = I > 0$. ♠

Let R be the recursion tree of $\mathcal{A}_{p+1}$. As usual identify each node of R with the corresponding invocation of $\mathcal{A}_{p+1}$. (R does not contain nodes for calls to $\mathcal{A}_p$.) For any node τ of R, C_τ denotes the cycle returned by τ.

Claim 1: *Any child σ of any node τ of R has $|C_\sigma| < |C_\tau|$.*

Proof: Suppose σ is called from Step 3.1 of τ. Lemma 2.1(i) implies $|C_\tau| \geq 1 + (|C_\sigma| - 2) + (I + 1) > |C_\sigma|$. Next suppose σ is called from Step 5.3. Without loss of generality we can assume C_σ corresponds to the longest path $P[r_0, r_1]$ in $\mathcal{P}$. Consider the cycle constructed in Step 7. The component X has a neighbor in $C - \{w_0, w_1\}$, by the construction of G' in Step 0. So in Step 7, $|C_v[c_1, c_0]| \geq 3$. Now even if $r_i = c_i$ for $i = 1, 2$ we have $|C_\tau| \geq (|C_\sigma| - 2) + |C_v[c_1, c_0]| > |C_\sigma|$. ♠

Claim 1 implies that if Step 0 ever returns because $\rho \geq a^{p+1}$, the cycle returned by the root of R has (more than) the desired length. So now assume R has height $< a^{p+1}$. We can also assume no invocation of Step 2 or 4 returns a cycle of length $\geq g$. To check this we need only show $g \geq \alpha_{p+1}(a/k)^{p+1}$, equivalently $\left(\frac{p}{p+1}\right)^{p+1} \geq \alpha_{p+1}$. The latter follows from $p \geq 1$, since the left-hand side equals $(1 - 1/(p+1))^{p+1} \geq (1 - 1/2)^2 = 1/4 \geq \alpha_2 \geq \alpha_{p+1}$.

Define the values

$$\delta = a^{2(p+1)}, \quad d = \log \delta = 2(p+1)\log a, \quad \epsilon = 4/a^{p+1}.$$

The core of the argument is the following:

Assertion: *Consider a node τ of R at depth $\geq j$, where j is an integer in $0 \leq j < a^{p+1}$. Let C^* be a longest v-cycle for τ. Assume for some integer i, $1 \leq i \leq \frac{a}{(p+1)d}$,*

$$|C^*| \geq 2^{\widetilde{a} + id - j\epsilon}.$$

Then $|C_\tau| \geq iI$.

Before proving the assertion let us show that it implies the lemma. The root γ of R has depth $j = 0$. We will choose $i = \lfloor \frac{a}{(p+1)d} \rfloor$ in the assertion. To show this value satisfies the assertion's lower bound on $|C^*|$ recall that the lemma assumes $|C^*| \geq 2^a$. Since $a = \widetilde{a} + a/(p+1) \geq \widetilde{a} + id$ this implies the desired lower bound.

Next we show our value of i satisfies

$$i \geq \frac{a}{4(p+1)^2 k}.$$

In proof, the definition of d shows that i is at least the floor of the quantity $a/(p+1)d = a/2(p+1)^2 \log a \geq a/2(p+1)^2 k$. So it suffices to show the last quantity is at least 1 (since $x \geq 1$ implies $\lfloor x \rfloor \geq x/2$). Inequality (1) applied to algorithm $\mathcal{A}_{p+1}$ shows $a \geq 24k(p+1)^2$. Hence the last quantity is $\geq 24k(p+1)^2/2(p+1)^2 k = 12 > 1$ as desired.

Since $(1 + 1/p)^p \leq e < 3$,

$$I = \frac{\alpha_p}{2} \frac{1}{(1+1/p)^p} \left(\frac{a}{k}\right)^p \geq \frac{\alpha_p}{6} \left(\frac{a}{k}\right)^p.$$

Combining the last two displayed inequalities with the assertion gives

$$|C_\gamma| \geq iI \geq \frac{\alpha_p}{24(p+1)^2} \left(\frac{a}{k}\right)^{p+1} = \alpha_{p+1} \left(\frac{a}{k}\right)^{p+1},$$

thus establishing the lemma.

We prove the assertion by another induction, this time on the quantity $i - j$. (The induction is well-founded since $i - j > 1 - a^{p+1}$.) For future reference note $d \geq 2(p+1) \geq 4$ implies

$$d \geq j\epsilon.$$

The next claim provides the base case of the induction.

Claim 2: *For any integers $1 \leq i \leq 2$ and $0 \leq j < a^{p+1}$, $|C_\tau| \geq 2I \geq iI$.*

Proof: Since $id - j\epsilon \geq d - j\epsilon \geq 0$, the assertion's lower bound on $|C^*|$ implies $|C^*| \geq 2^{\widetilde{a}}$. So when Step 1 is executed for the tricomponent containing C^*, $\mathcal{A}_p$ returns a cycle of length $\geq \alpha_p(\widetilde{a}/k)^p = 2I$. Here we have used the lemma for $\mathcal{A}_p$ (true by the inductive assumption on p) and the definition of $\widetilde{a}$. ♠

The inductive step assumes $i \geq 3$. It suffices to show that some execution of Step 3.1, Step 6 or Step 7 constructs a cycle of length $\geq iI$. Fix C as its value after Step 2 has completed. Call a segment of C^* *big* if it has length $\geq 2|C^*|/\delta$. We prove the inductive step by focusing on the big segments. (Claim 5 below shows they exist. Also note $2|C^*|/\delta$ is a large quantity: $|C^*|/\delta \geq 2^{3d-j\epsilon}/2^d = 2^{2d-j\epsilon} \geq 2^d = \delta$.) The next 3 claims exhaust all possibilities for the big segments.

Claim 3: *$|C_\tau| \geq iI$ if some big segment traverses a component X that has a neighbor $c \in C$ with $d_C(c, v) \geq I + 1$.*

Proof: Lemma 2.1(ii) shows that in Step 3.1 graph H has a v'-cycle of length $\geq (2|C^*|/2\delta + 1) + 2 > |C^*|/\delta \geq 2^{\widetilde{a}+(i-1)d-j\epsilon}$. The recursive call to $\mathcal{A}_{p+1}$ has depth $\geq j + 1$. Hence the inductive assertion implies the recursive call finds a cycle of length $\geq (i-1)I$. Lemma 2.1(i) shows the v-cycle constructed has length $\geq ((i-1)I-2)+d_C(c,v)+1 \geq iI$. ♠

Now assume the hypothesis of Claim 3 never holds. We will derive a bound (4) for estimating the performance of Step 5. For the rest of the lemma's proof the notation $C^*[x,y]$ abbreviates $C^*_{\overline{v}}[x,y]$, i.e., all subpaths of C^* that we refer to avoid v.

Consider any component X that contains a big segment, say $a_0, C^*[x_0,x_1], a_1$. From Step 4 we have assumed that $d_X(x_0,x_1) < g$. From Claim 3 we have assumed any edge $cx \in E[C,X]$ has $d_C(c,v) < I+1$. The latter shows any two edges $cx, c'x'$ in the same set F_i (defined in Step 5.1) have $|C_{\overline{v}}[c,c']| < I$. This implies each F_i is b-close, since Step 2 cannot enlarge C any more.

We show a useful intermediate inequality:

$$2 + 2d_X(x_0,x_1)b \leq |C^*|/\delta a^{p+1}. \tag{3}$$

In proof, note $g \leq (a/k)^{p+1} \leq (a/8)^{p+1}$. Hence

$$\begin{aligned} 2 + 2d_X(x_0,x_1)b &\leq 4d_X(x_0,x_1)b \leq 8(a/8)^{p+1}2^{\widetilde{a}} \\ &\leq a^{p+1}2^{\widetilde{a}} = (\delta/a^{p+1})2^{\widetilde{a}} = 2^{\widetilde{a}+d}/a^{p+1}. \end{aligned}$$

Using $i \geq 3$ and $d \geq j\epsilon$ gives $d \leq (i-1)d - j\epsilon$. Hence

$$2^{\widetilde{a}+d} \leq 2^{\widetilde{a}+(i-1)d-j\epsilon} \leq |C^*|/\delta.$$

Combining the last two displayed inequalities gives (3).

We wish to apply Lemma 2.6 (to X and our big segment). We need to verify its remaining hypotheses. For the lemma's lower bound on $|P|$ use (3) to get

$$2|C^*|/\delta - 2 \geq |C^*|/\delta \geq |C^*|/\delta a^{p+1} \geq 2 + 2d_X(x_0,x_1)b.$$

This shows $P = C^*[x_0,x_1]$ satisfies the lemma's length bound. Next the vertex $v \notin Y(E[X,Y])$ gives the lemma's hypothesis on Y.

Finally we need to check that $x_i \in X(F_i)$. We can assume $x_0 \in X(F_0)$ since we are free to rename vertices w_0 and w_1. So we must show $x_1 \in X(F_1)$. The previous displayed inequality implies $D_X(x_0,x_1) \geq 2|C^*|/\delta - 2 \geq d_X(x_0,x_1)b$. Thus $D_X(x_0,x_1)/d_X(x_0,x_1) \geq b$. Now Lemma 2.3($ii$) shows x_0 and x_1 are separated by a b-round bicomponent in X. Since F_0 is b-close, the independent edges a_0x_0 and a_1x_1 show $x_1 \in X(F_1)$.

We conclude that Lemma 2.6 applies to our big segment $a_0, C^*[x_0,x_1], a_1$. It shows r_0, r_1 is a separation pair. Thus Step 5.2 finds this pair and the corresponding tricomponent containing $C^*[r_0,r_1]$.

We claim

$$|C^*[r_0,r_1]| > |C^*[a_0,a_1]| - |C^*|/\delta a^{p+1}. \tag{4}$$

The definition of r_i and Lemma 2.5 show (even if $r_i \notin X$)

$$\begin{aligned} |C^*[r_0,r_1]| &> |C^*[x_0,x_1]| - 2d_X(x_0,x_1)b \\ &\geq |C^*[a_0,a_1]| - 2 - 2d_X(x_0,x_1)b. \end{aligned}$$

Applying (3) gives (4).

To help bound the right-hand side of (4) we use the following estimate:

$$|C^*|(1 - 1/a^{p+1}) \geq 2^{\widetilde{a}+id-(j+1)\epsilon}. \tag{5}$$

To prove this recall $\ln(1-x) \geq -2x$ for $0 < x \leq 1/2$. Hence $\log(1-1/a^{p+1}) = \log e \, \ln(1-1/a^{p+1}) \geq -4/a^{p+1} = -\epsilon$. Combining this with the assertion's assumption $|C^*| \geq 2^{\widetilde{a}+id-j\epsilon}$ gives (5).

We now analyze the two remaining possibilities for the inductive step. Recall the inductive quantity is $i - j$.

Claim 4: *$|C_\tau| \geq iI$ if two big segments exist.*

Proof: (4) and (5) show that each big segment satisfies

$$|C^*[r_0,r_1]| > (2|C^*|/\delta)(1 - 1/a^{p+1}) \geq 2^{\widetilde{a}+(i-1)d-(j+1)\epsilon+1}.$$

The recursive call in Step 5.3 corresponds to a node of depth $\geq j+1$. Hence by induction the recursive call finds an r_0r_1-path P of length $\geq (i-1)I - 2$.

Step 6 eventually considers the pair consisting of these two paths P. The Gluing Principle ensures Step 6 finds a v-cycle containing the paths. The cycle has length $\geq 2((i-1)I-2)+2 = iI + (i-2)I - 2 \geq iI$. The last inequality uses $i \geq 3$ and $I \geq 2$ (from (2)). ♠

Claim 5: *$|C_\tau| \geq iI$ if at most one big segment exists.*

Proof: C^* has $\leq |C|$ segments, and we have assumed (from Step 2) that $|C| \leq g$. So the segments that are not big have total length $\leq (2|C^*|/\delta)(a/k)^{p+1} \leq 2|C^*|/(ak)^{p+1}$. Since the last quantity is $< |C^*|$ the longest segment must be big. Denote this longest segment as $a_0, C^*[x_0,x_1], a_1$. Clearly its length is $\geq |C^*| - 2|C^*|/(ak)^{p+1}$. Using (4), (5), $\delta, k \geq 2$ and $p \geq 1$ gives

$$\begin{aligned} |C^*[r_0,r_1]| &> (|C^*| - 2|C^*|/(ak)^{p+1}) - |C^*|/\delta a^{p+1} \\ &\geq |C^*|(1 - 1/a^{p+1}) \geq 2^{\widetilde{a}+id-(j+1)\epsilon}. \end{aligned}$$

The recursive call in Step 5.3 corresponds to a node of depth $\geq j+1$. Hence by induction the recursive call finds an r_0r_1-path of length $\geq iI - 2$. Step 7 enlarges this path to a v-cycle of length $\geq (iI-2)+2 = iI$ as desired. ♠ ■

Suppose an iteration of Step 2 of the Main Routine has $|C^*| \geq 2^{a^*}$. We show $\mathcal{A}_{p^*}$ returns a cycle of length $\geq e^{p^*}$. For notational simplicity we drop the asterisks and write a, p, k for a^*, p^*, k^*.

Lemma 4.1 shows $\mathcal{A}_p$ returns a cycle of length $\geq \alpha_p(a/k)^p$. The definition of α_p and (1) gives

$$\alpha_p(a/k)^p \geq \frac{1}{24^{p-1}(p!)^2}(24p^2)^p \geq 24\frac{p^{2p}}{(p!)^2}.$$

Any $p \geq 1$ satisfies $p! \leq 3\sqrt{p}(p/e)^p$ [5, p.55]. Hence the rightmost quantity is at least

$$24\frac{p^{2p}}{9p(p/e)^{2p}} \geq \frac{e^{2p}}{p} \geq e^p.$$

This gives the desired inequality.

LEMMA 4.2. *The Main Routine returns a v-cycle of length $\geq \exp(c\sqrt{\log \ell/\log\log \ell})$ for some constant $c > 0$.*

PROOF. We have $p^* \geq \sqrt{a^*/24k^*}/2$ (since $x \geq 1$ implies $\lfloor x \rfloor \geq x/2$). Hence as shown above the returned cycle has length $\geq e^{p^*} \geq e^{c\sqrt{a^*/k^*}}$ for some constant $c > 0$. If Step 1 of the Main Routine does not find a longest cycle we have $\log \ell \geq 2^8$. Hence Step 2 eventually chooses k^* so that $a^* \leq \log \ell \leq 2a^*$. This implies $a^* \geq (\log \ell)/2$, $k^* \leq \log\log \ell$, and $a^*/k^* \geq (\log \ell)/2\log\log \ell$. The lemma follows. ■

5. IMPLEMENTATION AND TIMING ANALYSIS

We give the remaining implementation details and then show the algorithm runs in polynomial time.

Step 2.1

We break this step into a number of "passes", each pass but the last ending with an enlargement of C. Each pass examines the bicomponents B (of each X) that separate two independent edges $cx, c'x'$ selected as in Step 2.1, i.e., $|C_{\overline{v}}[c,c']| < I$. (If there is more than one such pair of edges for B, choose arbitrarily.) Let y (y') be the vertex of $V(B)$ on every minimal path from x (x') to B. So $y \neq y'$. Call $\mathcal{A}_p$ to find a yy'-path Y in B (creating an artificial vertex v' as in Step 2.1). If $|Y| \geq I$, enlarge Y to a path from cx to $c'x'$, and use that cc'-path to enlarge C. Then begin the next pass. Step 2 ends when $|C| \geq g$ or a pass examines every bicomponent B without enlarging C.

The argument for Claim 0 in Lemma 4.1 (applied to y, y') shows that if B is b-round, $|Y| \geq 2I - 2 \geq I$. Hence we enlarge C in this case. It is possible that $|Y| \geq I$ even if B is not b-round. It causes no harm that we enlarge C in this case too. Thus the implementation of Step 2.1 is correct.

For efficiency of the algorithm observe that in each pass, an edge of G is in the graph of ≤ 1 call to $\mathcal{A}_p$.

Step 5.2

This step needs to find the triplets of $\mathcal{T}$. First observe that the definition of $\mathcal{T}$ can be relaxed: $\mathcal{T}$ need only contain the triplets (r_0, r_1, T) that correspond to big segments of C^*, and only when $i \geq 3$. This follows since the proof of Lemma 4.1 only uses Step 5.2 to establish Claims 4 and 5.

We first mention a fact that is not needed for our development but may prevent misconceptions. It is possible that two big segments traverse the same component X. This can occur when one set F_i is a star centered in C and the other set F_{1-i} is not a star. (So C^* enters X on a segment starting in F_{1-i} and ending in F_i, and then leaves X on the next segment starting in F_i and ending in F_{1-i}.) However this is the only possibility: If these conditions are not met then only one big segment can traverse X. This fact follows from Lemma 5.2 below.

Now consider a big segment $a, C^*[x,x'], a'$ with $ax \in F_0$ and $a'x' \in F_1$, traversing a component X. For simplicity abbreviate F_0 to F. Suppose F is not a star. So the separating vertex r for F is given by Lemma 2.5. Let s be the first vertex in $C^*[x,x']$ that weakly separates $X(F)$ from $V(C^*(s,x'])$ in graph $G[X]$. Vertex s can be chosen as the separating vertex r (since s obviously satisfies the length inequality for r in Lemma 2.5). We now show how x' determines s.

Let $\overline{B}$ denote the union of all bicomponents of $G[X]$ that separate two vertices of $X(F)$. We will use this alternate characterization of $\overline{B}$: Let T be a minimal tree of $G[X]$ that spans $X(F)$. Minimality means that every leaf belongs to $X(F)$. Then $\overline{B}$ is the union of all bicomponents that contain an edge of T. This characterization of $\overline{B}$ follows from Lemma 2.2(ii).

For any vertex $v \in X$ let $\overline{v}$ be the vertex of $V(\overline{B})$ that is the end of every minimal path from v to $V(\overline{B})$. This vertex is well-defined. To show this suppose there are two minimal paths from v to $V(\overline{B})$, say P_1 and P_2, with P_i ending in vertex $b_i \in V(\overline{B})$ and $b_1 \neq b_2$. Let Q be a b_1b_2-path in $\overline{B}$. Then $P_1 \cup Q \cup P_2$ contains a cycle that includes edges in $\overline{B}$ and edges not in $\overline{B}$. But this contradicts the fact that $\overline{B}$ is a union of bicomponents.

LEMMA 5.1. *$\overline{x'}$ is the separating vertex r for F.*

PROOF. Write $s = \overline{x'}$. Clearly s belongs to the path $C^*[x,x']$. Observe further that s weakly separates $X(F)$ from $V(C^*(s,x'])$ in $G[X]$. This follows since any z in $C^*(s,x']$ has $\overline{z} = s$.

To complete the proof it suffices to show that no vertex $t \in V(\overline{B}) - s$ separates $X(F)$ from x'. Suppose there were such a t. Vertex s belongs to some bicomponent B that contains an edge uv of T. Let $T - uv$ consist of trees T_u and T_v indexed so that $u \in T_u, v \in T_v$ and $t \notin T_u$. Biconnectivity implies that B contains a us-path Q that avoids t. Now $Q \cup T_u$ contains a path from a leaf ℓ of T to s that avoids t. Since $\ell \in X(F)$ this implies t does not separate $X(F)$ from x'. ∎

To help identify the desired vertex s, for every vertex $b \in V(\overline{B})$ define R_b to be the set of all vertices v with $\overline{v} = b$. Clearly these sets are vertex disjoint and no edge of $G[X]$ leaves $R_b - b$.

The following lemma and its proof returns to using subscripts $i \in \{0,1\}$ to designate the two ends of the segment, e.g., x_i, F_i. We also use these subscripts to refer to notions defined above, e.g., set $\overline{B}_i$ corresponding to F_i, etc.

LEMMA 5.2. *If neither set F_0 nor F_1 is a star then for $i = 0,1$, vertex $s = \overline{x}_i$ is the unique vertex of $V(\overline{B}_{1-i})$ satisfying $X(F_i) \subseteq R_s$.*

PROOF. Define $s = \overline{x}_1$. By symmetry it suffices to show $X(F_1) \subseteq R_s$.

The definition of s implies path $C^*(s, x_1]$ is disjoint from $V(\overline{B}_0)$. The proof of Lemma 2.6 shows the separating vertex for F_1, $\overline{x}_0$, is in this path (i.e., the separating vertex follows s in $C^*[x_0,x_1]$). This $\overline{x}_0 \notin V(\overline{B}_0)$. By symmetry, $s = \overline{x}_1 \notin V(\overline{B}_1)$.

Any vertex $z \in X(F_1)$ has a path Q to x_1 contained in $\overline{B}_1$. Q avoids s, by the last remark. Now $x_1 \in R_s$ implies $z \in R_s$. ∎

We now give the implementation of Step 5.2. Recall the requirements that $\mathcal{T}$ must contain a triplet (r_0, r_1, T) corresponding to our big segment; furthermore all tricomponents T in the triplets of $\mathcal{T}$ must be edge-disjoint. There are 3 cases for Step 5.2:

Case 1: Both sets F_i are stars. Vertices r_i are their centers. Let the corresponding sets T range over all the r_0r_1-tricomponents not containing v.

Case 2: Neither set F_i is a star. Lemma 5.2 allows us to identify both separating vertices r_i. Again T ranges over all the r_0r_1-tricomponents not containing v. (In actuality there is only one.)

Case 3: Exactly one set, say F_1, is a star. r_1 is the star's center. Let s range over every vertex of $V(\overline{B}_0)$ such that $R_s - s$ is adjacent to r_1. Each such pair s, r_1 is a separation pair. The pair determines one or more triconnected components, which partition the edges of $G[R_s]$ plus $E[r_1, R_s]$; since the various sets R_s are vertex disjoint, all the triconnected components for X are edge-disjoint. Lemma 5.1 shows one of these separation pairs corresponds to our big segment. (This argument is not affected by the possibility that X contains two big segments.)

Step 6

Consider the search for a v-cycle containing two paths $P[r_0, r_1]$ and $P[s_0, s_1]$ of $\mathcal{P}$. Choose an internal vertex r' (s') of $P[r_0, r_1]$ ($P[s_0, s_1]$) respectively. Find a cycle A through v, r' and s', if one exists. The pair r_0, r_1 separates r' and v, by construction. Hence A traverses the r_0r_1-tricomponent containing r'. Replace the subpath $A_{r'}[r_0, r_1]$ by $P[r_0, r_1]$. Do the same for $P[s_0, s_1]$. This gives the desired cycle. (In particular the new cycle is simple.)

A can be found in polynomial time using the algorithm of Robertson and Seymour for fixed-vertex subgraph homeomorphism [12]. Instead we use the algorithm of LaPaugh and Rivest that finds a cycle through 3 given vertices [10]. This algorithm uses linear time with no large hidden constants.

Arithmetic

We sketch how all tests involving numeric quantities can be done in polynomial time. The Main Routine Step 2 computes p^*. $\mathcal{A}_{p+1}$ uses a^{p+1} (Step 0), g (Steps 2 and 4) and I (Steps 2.1 and 3). The last 3 quantities are not integers, but it suffices to compute their floor and ceiling, since they are only used in comparison with known integers. Each quantity is a quotient of integers of $O(\log n)$ bits. So the floors and ceiling can be found in polynomial time by exhaustive or binary search. Full details are in the complete paper.

Wrapup

THEOREM 5.3. *The Main Routine returns a v-cycle of length* $\exp(\Omega(\sqrt{\log \ell / \log\log \ell}))$ *in polynomial time.*

PROOF. Throughout this argument G^* denotes the given graph. Suppose the Main Routine calls $\mathcal{A}_{p^*}$. We need only show that $\mathcal{A}_{p^*}$ uses polynomial time.

First observe that every invocation of a routine $\mathcal{A}_p$, $p \le p^*$ is given a graph G that consists of 2 edges incident to v plus edges that are either in G^* or images of such edges. (An image of an edge in G^* is created in Step 3.1: an edge $c'x$ comes from an edge from $C - c$ to x.) This follows by a simple induction.

The pseudocode in Section 3 plus the implementation details of this section show that if we ignore the recursive calls that $\mathcal{A}_{p+1}$ makes to itself or $\mathcal{A}_p$, algorithm $\mathcal{A}_{p+1}$ takes polynomial time. So it suffices to show there are a polynomial number of invocations of all routines $\mathcal{A}_p$, $p \le p^*$. The assertion of the previous paragraph shows each invocation of a routine $\mathcal{A}_p$ uses a graph that contains an edge of G^* (or its image). Hence it suffices to show each edge e of G^* is in the graph of a polynomial number of invocations $\mathcal{A}_p$, $p \le p^*$.

Take any $p \le p^*$. Let R be the recursion tree of $\mathcal{A}_p$. As in the previous section R contains nodes only for calls to $\mathcal{A}_p$. Suppose edge e of G^* belongs to the graph of a node τ of R. Then e belongs to the graph of at most one child of τ. (The recursive calls to $\mathcal{A}_p$ occur in Steps 3.1 and 5.3. A component X is processed in only one of Steps 3 or 5. Step 5.2 ensures that all tricomponents T are pairwise disjoint, so a given edge is involved in only one recursive call in Step 5.3.) We conclude that e occurs in $\le a^p$ nodes τ of R. (Since we are discussing $\mathcal{A}_p$ rather than $\mathcal{A}_{p+1}$, we reduce p in the pseudocode of Section 3 by 1, e.g., Step 0 guarantees that $\rho \le a^p$.)

In each such node τ, e occurs in $\le g \le a^p$ graphs for calls to $\mathcal{A}_{p-1}$ in Steps 1 and 2.1. (This follows since each pass of Step 2.1 except the last enlarges C, and e belongs to ≤ 1 graph per pass.) We conclude that e occurs in at most $a^p \times a^p = a^{2p}$ calls from this invocation of $\mathcal{A}_p$ to $\mathcal{A}_{p-1}$.

Any routine $\mathcal{A}_p$ has $a \le a^*$, $p \le p^*$. So overestimating, edge e is in the initial graph of $\le ((a^*)^{2(p^*)})^i$ recursion trees for routine $\mathcal{A}_{p^*-i}$, for any $0 \le i < p^*$. Since e occurs in $\le (a^*)^{p^*}$ nodes in each tree, this gives a total of $\le (a^*)^{2p^* i} \times (a^*)^{p^*} \le (a^*)^{2p^*(i+1)}$ nodes.

In this last paragraph we drop asterisks and write a, p, k for a^*, p^*, k^*. We have shown that e occurs in a total of $\le pa^{2p^2}$ recursive invocations. Since $p \le a \le \log n$ it suffices to show the quantity a^{2p^2} is polynomial in n. In fact it is sublinear since

$$\log a^{2p^2} = 2p^2 \log a \le 2\tfrac{a}{24k}k < a \le \log n. \qquad \blacksquare$$

6. REFERENCES

[1] N. Alon, R. Yuster and U. Zwick, *Color-coding*, J. ACM 42 (1995), pp. 844–856.

[2] H.L. Bodlaender, *Minor tests with depth-first search*, J. Algorithms, 14 (1993), pp. 1–23.

[3] A. Björklund and T. Husfeldt, *Finding a path of superlogarithmic length*, SIAM J. Comput. 32, 6 (2003), pp. 1395–1402.

[4] A. Björklund, T. Husfeldt and S. Khanna, *Approximating longest directed path*, Electronic Colloq. on Comp. Complexity, Rept. No. 32, 2003.

[5] T.H. Cormen, C.E. Leiserson, R.L. Rivest and C. Stein, *Introduction to Algorithms*, 2nd Ed., McGraw-Hill, NY, 2001.

[6] M.R. Fellows and M.A. Langston, *Nonconstructive tools for proving polynomial-time decidability*, J. ACM 35 (1988), pp. 727–739.

[7] H.N. Gabow and S. Nie, *Finding a long directed cycle*, Proc. 15th Annual ACM-SIAM Symp. Disc. Alg., 2004, pp. 49–58.

[8] J.E. Hopcroft and R.E. Tarjan, *Dividing a graph into triconnected components*, SIAM J. Comput. 2,3 (1973), pp. 135–158.

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

[10] A.S. LaPaugh and R.L. Rivest, *The subgraph homeomorphism problem*, J. Comput. Sys. Sci. 20 (1980), pp. 133–149.

[11] B. Monien, *How to find long paths efficiently*, Annals Disc. Math., 25 (1985), pp. 239–254.

[12] N. Robertson and P.D. Seymour, *Graph Minors. XIII: The disjoint paths problem*, J. Comb. Th. B 63 (1995), pp. 65–110.

[13] R.E. Tarjan, *Depth-first search and linear graph algorithms*, SIAM J. Comput., 1 (1972), pp. 146–160.

[14] D.B. West, *Introduction to Graph Theory*, 2nd Ed., Prentice Hall (2001).