SIAM J. COMPUT.
Vol. 32, No. 6, pp. 1395–1402

FINDING A PATH OF SUPERLOGARITHMIC LENGTH*

ANDREAS BJÖRKLUND† AND THORE HUSFELDT†

Abstract. We consider the problem of finding a long, simple path in an undirected graph. We present a polynomial-time algorithm that finds a path of length $\Omega\big((\log L/\log\log L)^2\big)$, where L denotes the length of the longest simple path in the graph. This establishes the performance ratio $O\big(n(\log\log n/\log n)^2\big)$ for the longest path problem, where n denotes the number of vertices in the graph.

Key words. approximation algorithms, graph algorithms, longest path

AMS subject classifications. 68R10, 68W25

DOI. 10.1137/S0097539702416761

1. Introduction.

Given an unweighted, undirected graph $G = (V, E)$ with $n = |V|$, the *longest path* problem is to find the longest sequence of distinct vertices $v_1 \cdots v_k$ such that $v_i v_{i+1} \in E$.

This is a classical optimization problem; the Hamiltonian path problem is a special case of it and appears in Karp's original list of NP-complete problems [8]. While today the approximability of most of these problems is well understood, the longest path problem has remained elusive, and notoriously so [5]: In spite of a considerable body of research the gap between upper and lower bounds is very wide.

Previous work. The first approximation algorithms for longest path are due to Monien [9] and Bodlaender [3], both finding a path of length $\Omega(\log L/\log\log L)$ in a graph with longest path length L.

A natural and harder variant of the problem is to find a path of length $\log n$ if it exists. Papadimitriou and Yannakakis [10] conjectured that this could be done in polynomial time, which was confirmed by Alon, Yuster, and Zwick [1], introducing the important method of *color-coding*. If the longest path has length $O(\log n)$, then their algorithm finds it (or, rather, it finds *a* longest path); else it finds a path of length $\Omega(\log n)$. Especially, the algorithm finds an $\Omega(\log L)$-path and thus has the performance ratio $O\big(n/\log n\big)$, which is the best ratio for the longest path problem known prior to the present paper.

Motivated by the weakness of these bounds for general graphs the problem has received additional study for restricted classes of graphs. In Hamiltonian graphs the algorithm of Vishwanathan [11] finds a path of length $O\big((\log n/\log\log n)^2\big)$. In *sparse* Hamiltonian graphs, Feder, Motwani, and Subi [5] find even longer paths. Moreover, they prove the following remarkable result: If a graph with vertices of degree at most 3 has a cycle of length r, then one can find in polynomial time a cycle of length at least r^c, where $c = \frac{1}{2}\log_3 2$.

The hardness results for this problem are mainly due to Karger, Motwani, and Ramkumar [7]: The longest path problem does not belong to APX unless P = NP, and it cannot be approximated within $2^{\log^{1-\epsilon} n}$ unless NP $\subseteq$ DTIME$\big(2^{O(\log^{1/\epsilon} n)}\big)$ for

*Received by the editors October 29, 2002; accepted for publication (in revised form) June 20, 2003; published electronically September 9, 2003. A preliminary version of this work was announced at the 29th International Colloquium on Automata, Languages, and Programming (ICALP) 2002.
http://www.siam.org/journals/sicomp/32-6/41676.html
†Department of Computer Science, Lund University, P.O. Box 118, SE–221 00 Lund, Sweden (thore@cs.lu.se).

any $\epsilon > 0$. More recently, it was shown that for *directed* graphs, the problem admits stronger lower bounds [2].

This paper. We present a polynomial-time algorithm that finds a path of length $\Omega\big((\log L/\log\log L)^2\big)$ in a graph with longest path length L. Since $L < n = |V|$, this corresponds to a performance ratio of order

$$O\left(\frac{n(\log\log n)^2}{\log^2 n}\right). \tag{1.1}$$

The main idea of our algorithm is a new graph decomposition which forms the basis of a recursive procedure. We find a cycle C of length $\log n/\log\log n$, using the algorithm from [3], remove C, and continue recursively in the resulting connected components. This decomposes the graph into a number of disjoint cycles of sufficient length which can be assembled into a long path.

The performance ratio (1.1) was obtained earlier by Vishwanathan [11] but only for Hamiltonian graphs.

For bounded degree graphs, we can improve the ratio to $O\big(n\log\log n/\log^2 n\big)$. For 3-connected graphs, we establish the performance ratio (1.1) for the *longest cycle* problem, a variant of the problem that also requires $v_1v_k \in E$.

2. Paths and cycles. In what follows, we consider a connected graph $G = (V, E)$ with $n = |V|$ vertices and $e = |E|$ edges. We write $G[W]$ for the graph induced by the vertex set W.

The *length* of a path and a cycle is its number of edges. The length of a cycle C is denoted $l(C)$. A k-cycle is a cycle of length k, and a k^+-cycle is a cycle of length k or larger. A k-path and k^+-path are defined similarly. For vertices x and y, an xy-path is a (simple) path from x to y. If P is a path containing u and v, we write $P[u, v]$ for the subpath from u to v. We let $L_G(v)$ denote the length of the longest path from a vertex v in the graph G. The *path length* of G is $\max_{v\in V} L_G(v)$.

We need the following result: Theorem 5.3(i) of [3].

THEOREM 1 (Bodlaender). *Given a graph, two of its vertices s, t, and an integer k, one can find a k^+-path from s to t (if it exists) in time $O\big((2k)!2^{2k}n + e\big)$.*

COROLLARY 1. *Given a graph, one of its vertices s, and an integer k, one can find a k^+-cycle through s (if it exists) in time $O\big(((2k)!2^{2k}n + e)n\big)$.*

Proof. For all neighbors t of s, apply the theorem on the graph with the edge st removed. □

We also need the following easy lemma.

LEMMA 1. *If a connected graph contains a path of length r, then every vertex is an endpoint of a path of length at least $\frac{1}{2}r$.*

Proof. Given vertices $u, v \in V$, let $d(u, v)$ denote the length of the shortest path between u and v.

Let $P = p_0 \cdots p_r$ be a path, and let v be a vertex. Find i minimizing $d(p_i, v)$. By minimality there is a path Q from v to p_i that contains no other vertices from P. Now either $QP[p_i, p_r]$ or $QP[p_i, p_0]$ has length at least $\frac{1}{2}r$. □

2.1. Decomposition into cycles. The next lemma is central to our construction and describes the graph decomposition that underlies our recursive algorithm. It formalizes the following observation: Assume that a vertex v originates a long path P and v lies on a cycle C. Then the removal of C decomposes G into connected components, one of which must contain a large part of P.

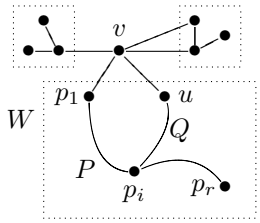

FIG. 1. *Statement* 1 *of Lemma* 2. *The path* $P = vp_1 \cdots p_r$ *continues in the component* W. *We assume that* v *does not lie on a large cycle. This means that an arbitrary path* Q *from* v*'s neighbor* u *must intersect* P *"early"; i.e.,* $QP[p_i, p_r]$ *is long.*

Pretending for a moment that our algorithm knew which component this is, we could continue the decomposition in it, recursively removing cycles until P is exhausted. In the end, we would have produced a long string of connected cycles. Especially, this string contains a path (using at least half the vertices of each cycle) that will be longer than the length of each individual cycle. The gist of this is that if we can find long *cycles* in graphs (like with Bodlaender's algorithm), then with our decomposition we can find even longer *paths*.

The lemma needs to distinguish between two cases, depending on whether or not v lies on a large cycle.

LEMMA 2. *Assume that a connected graph* G *contains a simple path* P *of length* $L_G(v) > 1$ *originating in vertex* v. *Then there exists a connected component* $G[W]$ *of* $G[V - v]$ *such that the following holds:*

1. *If* $G[W + v]$ *contains no* k^+*-cycle through* v, *then every neighbor* $u \in W$ *of* v *is the endpoint of a path of length*

$$L_{G[W]}(u) \geq L_G(v) - k.$$

2. *If* C *is a cycle in* $G[W + v]$ *through* v *of length* $l(C) < L_{G[W+v]}(v)$, *then there exists a connected component* H *of* $G[W - C]$ *that contains a neighbor* u *of* $C - v$ *in* $G[W + v]$. *Moreover, every such neighbor* u *is the endpoint of a path in* H *of length*

$$L_H(u) \geq \frac{L_G(v)}{2l(C)} - 1.$$

Proof. Let $r = L_G(v)$ and $P = p_0 \cdots p_r$, where $p_0 = v$. Note that $P[p_1, p_r]$ lies entirely in one of the components $G[W]$ of $G[V - v]$.

First consider statement 1; see Figure 1. Let $u \in W$ be a neighbor of v. Since $G[W]$ is connected, there exists a path Q from u to some vertex of P. Consider such a path. The first vertex p_i of P encountered on Q must have $i < k$, since otherwise the three paths vu, $Q[u, p_i]$, and $P[p_0, p_i]$ form a k^+-cycle. Thus the path $Q[u, p_i]P[p_i, p_r]$ has length at least $r - k + 1 > r - k$.

We proceed to statement 2; see Figure 2. Consider any cycle C in $G[W + v]$ through v. We will show that depending on how often P intersects C, there exists a long subpath in one of the components of $G[W - C]$. The length of this subpath is inversely proportional to the number of intersections, which could be no more than the length of C.

Case 1. First assume that $P \cap C = v$ so that one component H of $G[W - C]$ contains all of P except v. Let N be the set of neighbors of $C - v$ in H. First note

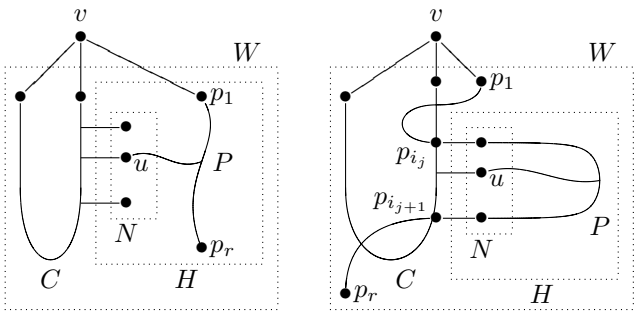

FIG. 2. *Statement* 2 *of Lemma* 2. *Here we assume that* v *does lie on a large cycle* C. *In Case* 1 *(left) the path* $P = vp_1 \cdots p_r$ *does not intersect* C *after it leaves* v. *Thus* $P[p_1, p_r]$ *lies entirely in a component* H *of* $W - C$. *Any neighbor* $u \in N$ *of* C *in this component must be the head of a long path using at least half of* $P[p_1, p_r]$. *In Case* 2 *(right) the path* P *intersects* C *in several places. Consider the largest section of* P *that lies entirely in a component* H *of* $W - C$, *here shown as a "loop" starting after* p_{i_j} *and ending before* $p_{i_{j+1}}$. *Any neighbor* $u \in N$ *of* C *in this component must be the head of a long path using at least half of the "loop."*

that N is nonempty, since $G[W]$ is connected. Furthermore, the path length of H is at least $r - 1$, so Lemma 1 gives $L_H(u) \geq (r-1)/2$ for every $u \in N$.

Case 2. Assume instead that $|P \cap C| = s > 1$. Enumerate the vertices on P from 0 to r and let $i_1, \ldots, i_s$ denote the indices of vertices in $P \cap C$, in particular $i_1 = 0$. Let $i_{s+1} = r$. An averaging argument shows that there exists j such that $i_{j+1} - i_j \geq r/s$. Consequently, there exists a connected component H of $G[W - C]$ containing a simple path of length $r/s - 2$. At least one of the i_jth or i_{j+1}th vertices of P must belong to $C - v$, so the set of neighbors N of $C - v$ in H must be nonempty. As before, Lemma 1 ensures $L_H(u) \geq r/2s - 1$ for every $u \in N$, which establishes the bound after noting that $s \leq l(C)$. ∎

3. Result and algorithm. The construction in this section and its analysis establishes the following theorem, accounting for the worst-case performance ratio of (1.1) as claimed in the introduction.

THEOREM 2. *If a graph contains a simple path of length* L, *then we can find a simple path of length*

$$\Omega\left(\left(\frac{\log L}{\log\log L}\right)^2\right)$$

in polynomial time.

We first give a brief overview of the algorithm; the next two sections will provide the details.

Assume for simplicity that the input graph is connected; if not, then we can iterate the algorithm over each connected component of the input graph and return the longest path found.

Pick any vertex v. Lemma 1 ensures that v is the head of a path of length at least $r > L/2$. In the next sections we will pretend that we know the value

$$k = \left\lceil \frac{2\log r}{\log\log r} \right\rceil;$$

but this is not a restriction since we can (in polynomial time) run the algorithm for every value of $k = 6, \ldots, \lceil 2\log n / \log\log n \rceil$ and return the longest path found.

Given v and k we will construct a tree $T_k(G, v)$ as detailed in section 3.1; this tree will describe a recursive decomposition of the input graph G into paths and cycles. Finally, we find a long (weighted) path in $T_k(G, v)$. This path will describe a path in G which will have the desired length as shown in section 3.2.

In summary, assuming a connected input graph, the algorithm proceeds as follows:

1. Pick any vertex $v \in G$.
2. For every $k = 6, \ldots, \lceil 2 \log n / \log \log n \rceil$ perform the following two steps and return the longest path found:
3. Construct the tree $T_k(G, v)$ as detailed in section 3.1.
4. Find a longest weighted path in $T_k(G, v)$ and return the path in G described by it, as detailed in section 3.2.

Steps 3 and 4 take polynomial time (see below), so the entire algorithm takes polynomial time.

3.1. Construction of the cycle decomposition tree. Given a vertex v in G, our algorithm constructs a node-weighted tree $T_k = T_k(G, v)$, rooted at v, called the *cycle decomposition tree*. Every node of T_k is either a *singleton* or a *cycle* node: A singleton node corresponds to a single vertex $u \in G$ and is denoted $\langle u \rangle$, while a cycle node corresponds to a cycle C with a specified vertex $u \in C$ and is denoted $\langle C, u \rangle$. Every singleton node has unit weight, and every cycle node $\langle C, u \rangle$ has weight $\frac{1}{2} l(C)$.

The tree $T_k(G, v)$ is constructed as follows. Initially, T_k contains a singleton node $\langle v \rangle$, and a call is made to the following procedure with arguments G and v:

1. [Iterate over components:] For every maximal connected component $G[W]$ of $G[V - v]$, execute step 2.
2. [Find cycle:] Search for a k^+-cycle through v in $G[W + v]$ using Theorem 1. If such a cycle C is found, then execute step 3; otherwise, execute step 5.
3. [Insert cycle node:] Insert the cycle node $\langle C, v \rangle$ and the tree edge $\langle v \rangle \langle C, v \rangle$. For every connected component H of $G[W - C]$ execute step 4.
4. [Recurse:] Choose an arbitrary neighbor $u \in H$ of $C - v$, and insert the singleton node $\langle u \rangle$ and the tree edge $\langle u \rangle \langle C, v \rangle$. Then, recursively execute step 1 to compute $T_k(H, u)$.
5. [Insert singleton node and recurse:] Pick an arbitrary neighbor $u \in G[W + v]$ of v, insert the node $\langle u \rangle$ and the tree edge $\langle v \rangle \langle u \rangle$, and recursively execute step 1 to compute $T_k\big(G[W], u\big)$.

Note that each recursive step constructs a tree that is connected to other trees by a single edge, so T_k is indeed a tree. Also note that the ancestor of every cycle node must be a singleton node. The root of T_k is $\langle v \rangle$.

To see that the running time of this procedure is polynomial, first note that step 2 is polynomial because of the corollary to Theorem 1. The number of recursive steps is linear, since every step inserts a node into T_k, which is clearly of linear size after the procedure.

3.2. Paths in the cycle decomposition tree. Our algorithm proceeds by finding a path of greatest weight in T_k. This can be done in linear time by depth first search. The path found in T_k represents a path in G if we interpret paths through cycle nodes as follows. Consider a path in T_k through a cycle node $\langle C, u \rangle$. Both neighbors are singleton nodes, so we consider the subpath $\langle u \rangle \langle C, u \rangle \langle v \rangle$. By construction, v is connected to some vertex $w \in C$ with $w \neq u$. One of the two paths from u to w in C must have length at least half the length of C; call it P. We will interpret the path $\langle u \rangle \langle C, u \rangle \langle v \rangle$ in T_k as a path uPv in G. If a path ends in a cycle node $\langle C, u \rangle$, we may associate it with a path of length $l(C) - 1$ by moving along C from u in any of its two

directions. Thus a path of weight m in T_k from the root to a leaf identifies a path of length at least m in G.

We need to show that T_k for some small k has a path of sufficient length.[1]

LEMMA 3. *If G contains a path of length $r \geq 2^8$ starting in v, then $T_k = T_k(G, v)$ for*

$$k = \left\lceil \frac{2 \log r}{\log \log r} \right\rceil$$

contains a weighted path of length at least $\frac{1}{8}k^2 - \frac{1}{4}k - 1$.

Proof. We follow the construction of T_k in section 3.1.

We need some additional notation. For a node $x = \langle w \rangle$ or $x = \langle C, w \rangle$ in T_k we let $L(x)$ denote the length of the longest path from w in the component $G[X]$ corresponding to the subtree rooted at x. More precisely, for every successor y of x (including $y = x$), the set X contains the corresponding vertices w' (if $y = \langle w' \rangle$ is a singleton node) or C' (if $y = \langle w', C' \rangle$ is a cycle node).

Furthermore, let $\mathbf{S}(n)$ denote the singleton node children of a node n, and let $\mathbf{C}(n)$ denote its cycle node children. Consider any singleton node $\langle v \rangle$.

Lemma 2 asserts that

$$L(v) \leq \max \left\{ \max_{w \in \mathbf{S}\langle v \rangle} L(w) + k, \max_{\substack{\langle C, v \rangle \in \mathbf{C}\langle v \rangle \\ w \in \mathbf{S}\langle C, v \rangle}} \big(2L(w) + 2\big) l(C) \right\}. \tag{3.1}$$

Define $n(v) = w$ if $\langle w \rangle$ maximizes the right-hand side of the inequality (3.1), and consider a path $Q = \langle x_0 \rangle \cdots \langle x_t \rangle$ from $\langle v \rangle = \langle x_0 \rangle$ described by these heavy nodes. To be precise, we have either $n(x_i) = x_{i+1}$ or $n(x_i) = x_{i+2}$; in the latter case, the predecessor of $\langle x_{i+2} \rangle$ is a cycle node.

We will argue that the gaps in the sequence

$$L(x_0) \geq L(x_1) \geq \cdots \geq L(x_t)$$

cannot be too large due to the inequality above. This, combined with the fact that $L(x_t)$ must be small (otherwise, we are done), implies that Q contains a lot of cycle nodes or even more singleton nodes.

Let s denote the number of cycle nodes on Q. Since every cycle node has weight at least $\frac{1}{2}k$ the total weight of Q is at least $\frac{1}{2}sk + (t - s) = s(\frac{1}{2}k - 1) + t$.

Consider a singleton node that is followed by a cycle node. There are s such nodes; we will call them *cycle parents*. Assume $\langle x_j \rangle$ is the first cycle parent node. Thus, according to the first part of Lemma 2, its predecessors $\langle x_0 \rangle, \ldots, \langle x_j \rangle$ satisfy the relation $L(x_{i+1}) \geq L(x_i) - k$, so

$$L(x_j) \geq r - jk \geq r - \tfrac{1}{8}k^3 \geq \tfrac{7}{8}r,$$

since $j \leq t \leq \frac{1}{8}k^2$ (otherwise, we are finished) and $r \geq k^3$.

From the second part of Lemma 2 we have

$$L(x_{j+2}) \geq \frac{7r}{16 l(C)} - 1 \geq \frac{r}{k^2},$$

[1] All logarithms are to the base 2, and the constants involved have been chosen aiming for simplicity of the proof rather than optimality.

where we have used $l(C) \leq \frac{1}{4}k^2$ (otherwise, we are finished) and $r \geq \frac{4}{3}k^2$.

This analysis may be repeated for the subsequent cycle parents as long as their remaining length after each cycle node passage is at least k^3. Note that Q must pass through as many as $s' \geq \lceil \frac{1}{4}k - 1 \rceil$ cycle nodes before

$$\frac{r}{k^{2s'}} < k^3,$$

at which point the remaining path may be shorter than k^3. Thus we either have visited $s \geq s'$ cycle nodes, amounting to a weighted path Q of length at least

$$s(\tfrac{1}{2}k + 1) \geq \tfrac{1}{8}k^2 - \tfrac{1}{4}k - 1$$

(remembering that any two consecutive cycle nodes must have a singleton node in between), or there are at most $s < s'$ cycle nodes on Q. In that case there is a tail of singleton nodes starting with some $L(x) \geq k^3$. Since $L(x_j) \leq L(x_{j+1}) + k$ for the nodes on the tail, the length of the tail (and thus the weight of Q) is at least k^2. □

It remains to check that the path found by our algorithm satisfies the stated approximation bound: For the right k, the preceding lemma guarantees a weighted path in $T_k(G, v)$, and hence a path in G, of length

$$\frac{k^2}{8} - \frac{k}{4} - 1 = \Omega\left(\left(\frac{\log r}{\log\log r}\right)^2\right) = \Omega\left(\left(\frac{\log L}{\log\log L}\right)^2\right)$$

because $r \geq \frac{1}{2}L$ by Lemma 1. This finishes the proof of Theorem 2.

4. Extensions.

4.1. Bounded degree graphs. As in [11], the class of graphs with their maximum degree bounded by a constant admits a relative $\log\log n$-improvement over the performance ratio shown in this paper. All paths of length $\log n$ can be enumerated in polynomial time for these graphs. Consequently, we can replace the algorithm from Theorem 1 by an algorithm that efficiently finds cycles of logarithmic length or larger through any given vertex if they exist.

PROPOSITION 1. *If a constant degree graph contains a simple path of length L, then we can find a simple path of length*

$$\Omega\left(\frac{\log^2 L}{\log\log L}\right)$$

in polynomial time.

This gives the performance ratio $O\big(n \log\log n / \log^2 n\big)$ for the longest path problem in constant degree graphs.

4.2. 3-connected graphs. Bondy and Locke [4] have shown that every 3-connected graph with path length L must contain a cycle of length at least $2L/5$. Moreover, their construction is easily seen to be algorithmic and efficient. This implies the following result on the longest cycle problem.

PROPOSITION 2. *If a 3-connected graph contains a simple cycle of length L, then we can find a simple cycle of length*

$$\Omega\left(\left(\frac{\log L}{\log\log L}\right)^2\right)$$

in polynomial time.

This gives the performance ratio $O\big(n(\log\log n/\log n)^2\big)$ for the longest cycle problem in 3-connected graphs. Note that for 3-connected cubic graphs, [5] shows a considerably better bound.

Acknowledgments. We thank Andrzej Lingas for bringing [11] to our attention and Gerth Stølting Brodal for commenting on a previous version of this paper.

Note added in proof. Recently, Gabow and Nie [6] have improved the bound in Corollary 1 to $O(e) + 2^{O(k)} n \log n$. As a consequence, the bounds in Theorem 2 and Proposition 2 are improved to $\Omega(\log^2 L/\log\log L)$, and the performance ratio for longest path becomes $O\big(|V|\log\log|V|/\log^2|V|\big)$.

REFERENCES

[1] N. Alon, R. Yuster, and U. Zwick, *Color-coding*, J. ACM, 42 (1995), pp. 844–856.
[2] A. Björklund, T. Husfeldt, and S. Khanna, *Approximating Longest Directed Path*, Technical report TR03-032, Electronic Colloquium on Computational Complexity, Vol. 10, 2003.
[3] H. L. Bodlaender, *On linear time minor tests with depth-first search*, J. Algorithms, 14 (1993), pp. 1–23.
[4] J. A. Bondy and S. C. Locke, *Relative length of paths and cycles in 3-connected graphs*, Discrete Math., 33 (1981), pp. 111–122.
[5] T. Feder, R. Motwani, and C. Subi, *Approximating the longest cycle problem in sparse graphs*, SIAM J. Comput., 31 (2002), pp. 1596–1607.
[6] H. N. Gabow and S. Nie, *Finding a Long Directed Cycle*, CU Technical report CU-CS-961-03, University of Colorado, Boulder, CO, 2003.
[7] D. Karger, R. Motwani, and G. D. S. Ramkumar, *On approximating the longest path in a graph*, Algorithmica, 18 (1997), pp. 82–98.
[8] R. M. Karp, *Reducibility among combinatorial problems*, in Complexity of Computer Computations, Plenum Press, New York, 1972, pp. 85–103.
[9] B. Monien, *How to find long paths efficiently*, Ann. Discrete Math., 25 (1985), pp. 239–254.
[10] C. H. Papadimitriou and M. Yannakakis, *On limited nondeterminism and the complexity of the V–C dimension*, J. Comput. System Sci., 53 (1996), pp. 161–170.
[11] S. Vishwanathan, *An approximation algorithm for finding a long path in Hamiltonian graphs*, in Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms, San Francisco, CA, 2000, pp. 680–685.