On Approximating the Longest Path in a Graph

David Karger [*] *Rajeev Motwani* [†] *G.D.S. Ramkumar* [‡]

Department of Computer Science
Stanford University
Stanford, CA 94305

Abstract

We consider the problem of approximating the longest path in undirected graphs. In an attempt to pin down the best achievable performance ratio of an approximation algorithm for this problem, we present both positive and negative results. First, a simple greedy algorithm is shown to find long paths in dense graphs. We then consider the problem of finding paths in graphs that are guaranteed to have extremely long paths. We devise an algorithm that finds paths of a logarithmic length in Hamiltonian graphs. This algorithm works for a much larger class of graphs (weakly Hamiltonian), where the result is the best possible. Since the hard case appears to be that of sparse graphs, we also consider sparse random graphs. Here we show that a relatively long path can be obtained, thereby partially answering an open problem of Broder, Frieze and Shamir.

To explain the difficulty of obtaining better approximations, we prove some strong hardness results. We show that, for any $\epsilon < 1$, the problem of finding a path of length $n - n^{\epsilon}$ in an n-vertex Hamiltonian graph is **NP**-hard. We then show that no polynomial-time algorithm can find a constant factor approximation to the longest path problem unless $\mathbf{P} = \mathbf{NP}$. We conjecture that the result can be strengthened to say that for some constant $\delta > 0$, finding an approximation of ratio n^{δ} is also **NP**-hard. As evidence towards this conjecture, we show that if any polynomial-time algorithm can approximate the longest path to a ratio of $2^{O(\log^{1-\epsilon} n)}$, for any $\epsilon > 0$, then **NP** has a quasi-polynomial deterministic time simulation. The hardness results apply even to the special case where the input consists of bounded degree graphs. These hardness results have found application towards establishing limits on the approximability of the Moderate-Mean Cycle problem.

[*] Supported an NSF Graduate Fellowship, NSF Grant CCR-9010517, and grants from Mitsubishi Corporation and OTL.

[†] Supported by an Alfred P. Sloan Research Fellowship, an IBM Faculty Development Award, grants from Mitsubishi and OTL, NSF Grant CCR-9010517, and NSF Young Investigator Award CCR-9357849, with matching funds from IBM, Schlumberger Foundation, Shell Foundation, and Xerox Corporation.

[‡] Supported by a grant from Toshiba Corporation.

1. Introduction

The area of approximation algorithms for **NP**-hard optimization problems has received a lot of attention over the past two decades [16, 22]. Although some notable positive results have been obtained, such as the fully polynomial approximation scheme for bin packing [14, 19], it has now become apparent that even the approximate solution of a large class of **NP**-hard optimization problems remains outside the bounds of feasibility. For example, a sequence of results [13, 8, 2, 1] established the intractability of approximating the largest clique in a graph, culminating in the result of Arora, Lund, Motwani, Sudan and Szegedy [1] that for some constant $\delta > 0$, there does not exist any polynomial-time algorithm that will approximate the maximum clique within a ratio of n^δ unless $\mathbf{P} = \mathbf{NP}$. Arora et al. [1] also established that unless $\mathbf{P} = \mathbf{NP}$, there do not exist polynomial-time approximation schemes (PTAS) for optimization problems that are **MAX SNP**-hard. The class **SNP** is a strict version of **NP** and was defined by Papadimitriou and Yannakakis [23] based on a syntactic characterization of **NP** due to Fagin [12]. They also provided a notion of approximation-preserving reductions for problems in this class and, under this reduction, identified a large number of approximation problems that are **MAX SNP**-hard, and are therefore unlikely to have any PTAS. These problems include such widely studied problems as MAX 3SAT, vertex cover, metric TSP and Steiner trees. Recently, Lund and Yannakakis [20] settled another important open problem by showing that the chromatic number of a graph is as hard to approximate as the clique number.

In this context, a major outstanding open problem is that of determining the approximability of the **longest path** in an unweighted **undirected** graph. The optimization version of this problem is **NP**-hard since it includes the Hamiltonian path problem as a special case. Therefore, it is natural to look for polynomial-time algorithms with a small performance ratio, where the performance ratio is defined as the ratio of the longest path in the input graph to the length of the path produced by the algorithm. Our results attempt to pin down the best possible performance ratio achievable by polynomial-time approximation algorithms for longest paths. We provide some approximation algorithms for this problem, but unfortunately the performance ratio of these algorithms is as weak as in the case of the best-known approximation algorithms for clique [7] and chromatic number [4, 18]. We explain the difficulty of obtaining better performance guarantees for longest path approximations by providing hardness results. These results come fairly close to establishing our conjecture that the situation for longest paths is essentially as bad as for the above two problems, i.e. if there exists an approximation algorithm that has a performance ratio of n^δ, for some constant $\delta > 0$, then $\mathbf{P} = \mathbf{NP}$.

In Section 2, we present several polynomial-time approximation algorithms for longest paths. A simple greedy algorithm is presented and it is shown that it finds long paths in dense graphs. At this point, this is the best algorithm known for arbitrary dense graphs. In the case of cliques and chromatic number, the extreme hardness of the problem led to the study of special inputs where the optimum was guaranteed to take on an extreme value; for example, the approximate coloring of 3-colorable graphs was studied by Blum [4] and Karger, Motwani and Sudan [18], and the approximation of cliques in graphs containing a linear-sized clique is studied by Boppana and Halldorsson [7]. We therefore formulate the problem of finding long paths in Hamiltonian graphs. For the purposes of this paper, there is no essential difference between the cases where the input graph has Hamiltonian paths or Hamiltonian cycles, and we concentrate on the latter case. Our second algorithm finds paths of a logarithmic length in Hamiltonian graphs. In fact, we show that this algorithm will find such paths in a much larger class of graphs, viz. **weakly Hamiltonian** graphs, or even **1-tough** graphs. Some variants of this algorithm are also analyzed. This result is the best possible in the sense that we can demonstrate the existence of such graphs where the longest path is of logarithmic length.

The hard case appears to be that of finding better approximations for sparse Hamiltonian graphs. In Section 3, we consider sparse random Hamiltonian graphs and show that it is possible to find paths of length $\Omega(\sqrt{n}/\log n)$. Surprisingly, this algorithm works in any graph obtained by adding **any number** of random edges to a Hamiltonian cycle. This result partially answers an open question posed by Broder, Frieze and Shamir [6]. They had considered the problem of finding Hamiltonian cycles in graphs obtained by adding a relatively large number of random edges to a Hamiltonian cycle.

In Section 4, we provide hardness results for the problem of approximating the longest path. We first consider the problem of finding long paths in Hamiltonian graphs and show that for any $\epsilon < 1$, it is impossible to find paths of length $n - n^\epsilon$ in an n-vertex Hamiltonian graph unless $\mathbf{P} = \mathbf{NP}$. The problem of finding long paths is easier for Hamiltonian graphs than for arbitrary inputs. Therefore, it is not surprising that we can prove much stronger negative results in general input graphs. We first prove a self-improvability result for the longest path problem. Combining this with the recent results on the intractability of approximation problems that are **MAX SNP**-hard, we obtain that no polynomial-time algorithm can find a constant factor approximation for the longest path problem unless $\mathbf{P} = \mathbf{NP}$. We conjecture that the result can be strengthened to say that for some constant $\delta > 0$, finding an approximation of ratio n^δ is also **NP**-hard. As evidence towards this conjecture, we show that if any polynomial-time algorithm can approximate the longest path to a ratio of $2^{\mathrm{O}(\log^{1-\epsilon} n)}$, for any $\epsilon > 0$, then **NP** has a quasi-polynomial deterministic time simulation. The hardness results apply even to the special case where the input consists of **bounded degree** graphs.

These hardness results have found application to the Moderate-Mean Cycle problem. Subramanian [25] established that the Moderate-Mean Cycle problem is **NP**-complete by reducing the Hamiltonian cycle problem to it. The reduction also establishes that finding an approximate solution to Moderate-Mean Cycle is at least as hard as finding good approximations to Hamiltonian paths, and so our hardness results carry over to the new problem too.

Before describing our results in greater detail we review some related work. Monien [21] presented an $\mathrm{O}(k!\,nm)$ time algorithm that finds paths of length k in a Hamiltonian graph with n vertices and m edges. Our results are an improvement on this since in polynomial-time Monien's algorithm can only find paths of length $\mathrm{O}(\log n/\log\log n)$. Furer and Raghavachari [15] present approximation algorithms for the minimum-degree spanning tree problem that delivered absolute performance guarantees (within an additive factor of 1). From this we can derive a polynomial-time algorithm for finding logarithmic length paths in Hamiltonian graphs, matching our result for that case. However, note that our result even in that case is more general in that it applies to a wider class of graphs, viz. the weakly Hamiltonian graphs. No hardness results for longest paths were known earlier, although a seemingly related problem has been studied by Berman and Schnitger [8]. They show that the hardness conjecture we stated is true for the problem of approximating the longest **induced** path in an undirected graph. Note that the induced path problem is strictly harder and their hardness result does not carry over to the problem under consideration here. Bellare [3] considers a generalization of the longest paths problem called the *longest color-respecting path* problem. This involves graphs with 2-colored edges and labeled vertices, and a feasible path must have the property that at each vertex its label specifies whether the incident edges of the path are of the same color or not. He obtains essentially the same hardness results through different techniques. Our results are strictly stronger since there are no color constraints on the paths.

1.1. Weak Hamiltonicity

Chvatal [9, 10, 11] considered several necessary conditions for Hamiltonicity obtained via an integer linear programming formulation of the problem. The graphs satisfying these conditions are known as *weakly Hamiltonian graphs.* We omit the formal definition and instead provide three properties of weakly Hamiltonian graphs – these are referred to as the "1-2-3" properties. Given a graph $G(V, E)$, and a set $U \subseteq V$, we will denote the vertex induced subgraph of G by $G[U]$.

Definition 1: [1-Toughness] *A graph $G = (V, E)$ is said to be 1-tough if for any set $U \subset V$, the induced subgraph $G[V - U]$ has at most $|U|$ connected components.*

In other words, by removing any k vertices from the graph, the graph cannot be decomposed into more than k connected components.

Definition 2: [2-Factors] *A graph $G = (V, E)$ is said to have a 2-factor if there exists $E^{'} \subset E$ such that in the graph $G^{'} = (V, E^{'})$, each vertex $v \in V$ has degree 2.*

That is, a 2-factor is a subgraph of G made up of cycles that cover all the vertices.

Definition 3: [3-Cyclability] *A graph $G = (V, E)$ is said to be 3-cyclable if for every three vertices $u, v, w \in V$, there exists a cycle in the graph G containing u, v and w.*

Theorem 1: [Chvatal] *Any weakly Hamiltonian graph $G = (V, E)$ is* 1-tough*, has a* 2-factor *and is* 3-cyclable

It is not very hard to see that every Hamiltonian graph G satisfies these three properties. Removing any k vertices from a cycle decomposes it into a collection of k paths; hence, a Hamiltonian graph can be decomposed into at most k connected components upon the removal of k vertices, implying 1-toughness. A Hamiltonian cycle is a 2-factor of the graph G. Finally, 3-cyclability follows from the fact that every set of three vertices lies on the Hamiltonian cycle.

Theorem 2: *Every Hamiltonian graph is weakly Hamiltonian.*

2. Algorithms for Finding Long Paths

We now give algorithms for finding long paths in 1-tough graphs. Unless otherwise specified, for any graph $G(V, E)$ we will assume that $|V| = n$ and $|E| = m$. We will measure the length of a path in terms of the number of vertices in it. All logarithms are to base 2. First we describe a simple algorithm that finds long paths in dense graphs. Then, we show how to find a path of length $\Theta(\log n)$ in any one tough graph.

Theorem 3: *Any graph with m edges and n vertices contains a path of length $d = m/n$.*

Proof: Consider first a graph with minimum degree d. Clearly, any maximal path in this graph has length at least d. Thus, a greedy construction will find a path of this length. Now, consider a graph with n vertices and m edges, and let $d = m/n$. Repeatedly choose a vertex of degree smaller than d, and remove this vertex and all incident edges from the graph. This process terminates when the residual graph has minimum degree at least d. Clearly, the residual graph cannot be empty since at most $(d-1)n < m$ edges can be removed from the graph in the process of deleting small degree vertices. But this graph has minimum degree d, so a greedy construction works.
□

Theorem 4: *Let G be a 1-tough graph with n vertices. Then the depth-first search tree of G has depth $\Omega(\log n)$.*

Proof: Take any depth-first search tree, and consider the root-leaf path defined as follows: at each step, move into the largest subtree. Suppose this path has length l and that the vertices on this path have degrees $d_1, \ldots, d_l$ in the depth-first tree. Then since a depth-first tree has no cross edges, removing the l vertices on this path necessarily disconnects each of the children of these l vertices from all the others, yielding exactly $\sum(d_i - 1)$ connected components. By the 1-toughness property, we know that $\sum(d_i - 1) \leq l$, i.e. $\sum d_i \leq 2l$. On the other hand, since we take the largest branch at each step, it is necessarily the case that the subtree rooted at d_j has size at least $n(\frac{1}{d_1}\frac{1}{d_2}\cdots\frac{1}{d_{j-1}})$. Thus, $\Pi d_i \geq n$. For any fixed l, the product Πd_i is maximized (under the constraint $\sum d_i \leq 2l$) by making all the d_i equal, namely $d_i = 2$. That is, the root-leaf path length l is minimum (and is $\Theta(\log n)$) when each of the d_i is 2. Therefore, $l \geq \log n$.
□

Corollary 1: *In a Hamiltonian graph on m edges, a path of length k can be found in $O(m + k^2!)$ time.*

Proof: By the same arguments as the previous theorem, at any level of the depth-first tree, the number of nodes at that level is at most equal to the number of nodes in all the previous levels (thus, the tree that is found is very close to being a binary tree in an "average" sense). If the depth of the tree exceeds k, then we are done. Otherwise, find some subtree containing between k and k^2 vertices. Such a subtree must exist since, by arguments similar to those above, every tree node has at most k children). The path from the root to this subtree, which contains at most k vertices, cuts the subtree off from the rest of the graph, so the Hamiltonian path can enter and then exit the subtree at most k times. It follows that this subtree contains a path of length k, which can be found by exhaustive search.

□

2.1. Tough Graphs with Short Paths

We now prove an upper bound of $O(\log n)$ on the longest path in 1-tough graphs. In fact, we prove that there exist graphs satisfying the 1-2-3 properties that have a longest path of length $O(\log n)$. We define a graph $G_k(V_k, E_k)$ for each non-negative integer k(refer to Fig. 1). The graph consists of $2^k - 1$ *normal* vertices and 2 *super* vertices. The normal vertices are arranged in a complete binary tree. The tree is augmented by an edge between each pair of siblings. We will distinguish between the *tree* edges and the *sibling* edges. The two super vertices are called s_1 and s_2, and they are adjacent to every other vertex in the graph, including each other. The size of V_k is $n = 2^k + 1$.

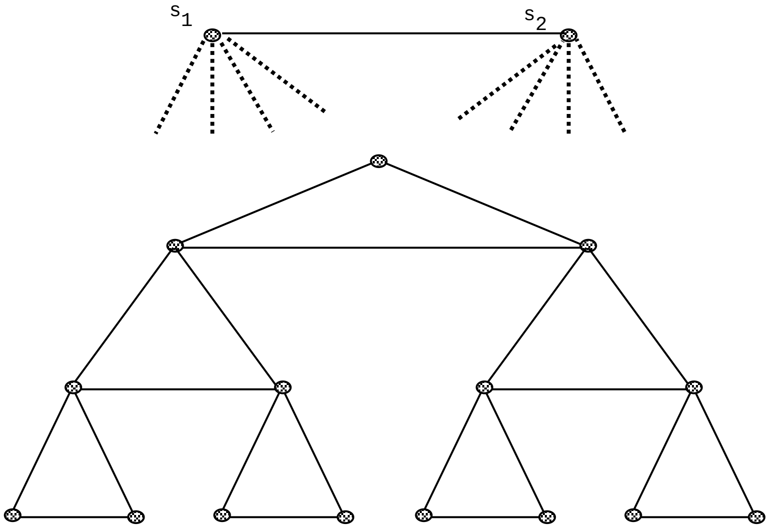

Figure 1: *A weakly Hamiltonian graph with no long paths.*

The following lemmas show that G_k has the 1-2-3 properties.

Lemma 1: *The graph G_k is 1-tough.*

Proof: Suppose that the graph G_k did not contain the super vertices. We then show that the removal of any set W of normal vertices yields an induced subgraph $G_k[V_k - W]$ with at most $|W| + 1$ connected components. This is proved by induction on $|W|$. It is necessary to prove a stronger inductive claim, which is as follows: by removing the set W of vertices, the graph is decomposed into at most $|W| + 1$ components *and* each component is a binary tree (not necessarily complete) with additional edges between each siblings.

The claim is trivially true in the case $|W| = 0$. Let us assume that the claim is true for $|W| = r$. Now suppose that we delete the set $W \cup \{w\}$, where $w \notin W$ is a normal vertex. There are two cases, depending on whether the vertex w is a leaf or a non-leaf vertex of the component of $G_k[V_k - W]$ containing it. If it is a leaf, then the number of components is unchanged by the additional deletion. But if it is a not a leaf vertex, then it is easy to see that the component in which w lies is split into two components. Also, it remains true that there is an edge between each pair of siblings. This establishes the inductive claim.

Now consider the super vertices also. Suppose we delete a set $W \subseteq V_k$ from G_k. Clearly, the residual graph remains connected if W does not contain both s_1 and s_2. Let us restrict our attention to the case where $s_1 \in W$ and $s_2 \in W$, and let $W^{'} = W - \{s_1, s_2\}$. By the preceding claim, the removal of W from G_k can yield at most $|W^{'}| + 1$ components. This implies that the number of components is $|W| - 1$, establishing the 1-toughness of G_k.
□

Lemma 2: *The graph G_k has a 2-factor.*

Proof: First notice that the set of normal vertices in any two successive levels of the tree are easily decomposable into a disjoint collection of triangles.

Consider first the case where the number of levels (i.e. k) is even. We can pair up the levels of the graph and obtain a set of disjoint triangles involving all the vertices except the two supervertices. The two super vertices can now be combined with any one triangle to yield a K_5, i.e. the complete graph on 5 vertices. Since K_5 contains a 5-cycle, we obtain a collection of disjoint cycles that includes each vertex. This is a 2-factor. When the number of levels is odd, the two super vertices form a triangle with the root and the remaining levels are partitioned into triangles as before, yielding a 2-factor.
□

Lemma 3: *The graph G_k is 3-cyclable.*

Proof: Let v_1, v_2, v_3 be any three vertices. Assume for now that none of the three is a super vertex. Clearly, there is a path between v_1 and v_2 using only the tree edges. We can append the two super vertices to the two end-points of this path, and use the edge from s_1 to v_3 and from v_3 to s_2 to complete a cycle containing all three vertices. Suppose now that one of the three vertices, without loss of generality v_3, is a super vertex. Then there exists a path from v_1 to v_2 not including v_3. Since v_3 is a super vertex, it is adjacent to both v_1 and v_2, and those edges complete the cycle. If two of the vertices are super vertices, then we obtain a triangle containing all three vertices.
□

The following lemma helps establish that the longest path of G_k is of length $\mathrm{O}(\log n)$,

Lemma 4: *The longest path in the graph G_k not containing the super vertices is of length $\Theta(\log n)$*

Proof: Consider the graph obtained by deleting the two super vertices; it is a complete binary tree with all the sibling edges thrown in. For convenience, we will measure the length of a path in terms of the number of edges in it. We prove the lemma by induction on the number of levels in the binary tree. The inductive hypothesis is that the length of the longest path is at most $4k$ and the length of the longest path ending at the root is at most $2k$. The base case, for $k = 1$, is trivial.

Now consider the graph with k levels, and note that $k = \log n$. The longest path in this graph consists of the longest path in the left subtree of the root ending at its root, a connecting path of length two through the root, and the longest path in the right subtree of the root ending at its root. Hence, the length of the longest path in a graph of k levels can be at most two more than the sum of the lengths of the longest path in left and right subtrees (both with $k-1$ levels) ending at their roots. By the inductive hypothesis, this means the longest path is no more than $4k$. Also, the length of the longest path ending at the root is at most 2 greater than the length of the longest path ending at the root in either the left or the right subtree. Again, by the inductive hypothesis, the longest path ending at the root is at most $2k$.

It is not hard to see that these upper bounds are tight within small additive constants.
□

The length of the longest path in G_k can be greater than the length of the longest path not including the super vertices by at most a factor of 3. This is because any path containing the two super vertices consists of three paths not containing the super vertices, stitched together by the super vertices. We have established a bound on the longest path in the entire graph G_k.

Theorem 5: *The graph G_k satisfies the 1-2-3 properties, and the longest path in this graph is of length $\Theta(\log n)$.*

3. Finding Long Paths in Sparse Random Graphs

We now turn to the issue of random Hamiltonian graphs. It should be noted that the amplification described later in the paper becomes ineffectual when we consider finding paths of length n^ϵ. It is therefore an interesting coincidence that on random Hamiltonian graphs, a path of length $\Omega(\sqrt{n}/\log n)$ can in fact be found. The following analysis applies to numerous distributions on graphs—in fact, to any random graph obtained by adding any number of edges to a Hamiltonian cycle, such that each non-Hamiltonian edge has an equal probability of occurring. For example, the random edges could form a $G_{n,p}$ graph, or a random regular graph [5]. For the case of random Hamiltonian graphs with average degree much larger than 3, Broder et al [6] give an algorithm that actually finds the Hamiltonian cycle in polynomial-time. However, they leave open the question of what can be done for sparse graphs, e.g. graphs of degree 3.

Theorem 6: *Let G be a random 3-regular Hamiltonian graph on n vertices. There is a polynomial-time algorithm that finds a path of length $\Omega(\sqrt{n}/\log n)$ in G, with high probability of success.*

A random 3-regular Hamiltonian graph can be viewed as a Hamiltonian cycle with a random perfect matching added. We will therefore refer to the edges on the Hamiltonian path as *Hamiltonian edges*, and will refer to the other edges as *matching edges*. We will call the neighbor of a vertex along a matching edge its *match*. The key property of the random edges used in this analysis is that for any vertex, its match is chosen uniformly at random from among the other vertices. Finally, let the *Hamiltonian distance* between two vertices be the length of the shortest path between them made up entirely of Hamiltonian edges.

The algorithm proceeds as a series of trials. In each trial, we start at a randomly selected vertex of the graph and perform a random walk, with the modification that the last edge traversed in not considered as a possibility for the next step, so that we do not immediately back up along the walk. Mark each vertex as it is visited. The trial ends when we visit a marked vertex, and succeeds if we visit $k = \Omega(\sqrt{n}/\log n)$ vertices before encountering a vertex that is already marked (during this or any previous trial). We claim that with high probability, we will find a path of length k within $\mathrm{O}(\log n)$ trials.

Consider the t^{th} trial, $t = \mathrm{O}(\log n)$. At the end of this trial at most kt vertices can have been marked. Analyze the walk during this trial as a series of epochs. An epoch ends when the walk traverses a matching edge. Since at each step of the walk a matching edge is traversed with probability 1/2, epochs have length $\mathrm{O}(\log n)$ with high probability. Call a vertex *safe* if its Hamiltonian distance from any marked vertex exceeds $\Omega(\log n)$. Call an epoch *safe* if it begins at a safe vertex. It follows that with high probability no marked vertex is visited during a safe epoch. Thus, the path continues to extend safely so long as safe epochs occur.

Consider the end of a safe epoch. This happens when a matching edge is traversed. Since the endpoints of a matching edge are random, this means that the starting vertex of the next epoch is chosen uniformly at random from among the unvisited vertices. Since at most kt vertices are marked, there are $\mathrm{O}(kt \log n)$ unsafe vertices; thus, the next epoch is unsafe with probability $\mathrm{O}(kt \log n/n) = \mathrm{O}(\log n/\sqrt{n})$. It follows that with constant probability $\Omega(\sqrt{n}/\log n)$ safe epochs occur during the trial. Clearly the length of the path that is found is equal to at least the number of epochs, so we get the desired path length. Furthermore, since a trial ends when we visit an unsafe vertex, we can treat the trials as independent. Thus, with high probability one of the $\mathrm{O}(\log n)$ trials succeeds.

In fact, the algorithm itself can be made deterministic. All that is required is that an epoch (in the sense described above) ends in $\mathrm{O}(\log n)$ steps. To ensure this we use a form of “universal non-traversal sequence.” Consider assigning labels to the edges of the graph in the standard traversal sequence manner. It is simple to interpret a sequence over $\{0, 1\}$ so that when one arrives at a given vertex via some edge, the next symbol in the sequence determines which of the other two edges should be traversed to leave that vertex. We can now modify traversal sequences in the following way to produce walks that do not “back up.” Given a traversal sequence drawn from $\{1, 2\}^*$, interpret it as follows: at each step, examine the label on the edge the walk arrived from. If i is the traversal symbol and j the label of the edge just traversed, then next traverse the edge labeled $(i + j) \bmod 3$.

Now consider the particular modified traversal sequence that causes a traversal of the Hamiltonian path. This is a sequence of length n, and it therefore follows that some sequence of length $\log n + 1$ does not occur as a substring of the traversal sequence (since there are $2n$ possible sequences of which only n occur in the Hamiltonian traversal). If we try all $2n$ such sequences, we will eventually stumble upon one satisfying the property. If we construct our walk using this sequence (repeated over and over again) then it is clear that we will traverse a matching edge at least once every $2 \log n$ steps of our walk, and then the analysis given above is applicable.

The random walk algorithm can also be generalized to any other random graph distribution, so long as the endpoint of a randomly chosen non-Hamiltonian edge out of v is uniformly distributed among the vertices of G. The argument goes through unchanged, except in the case that some of the vertices of G have degree two (*i.e,* only Hamiltonian edges. If such vertices are forbidden in the distribution, then we have the desired result immediately.

The degree two situation can be rectified if for any input graph we first "shortcut" any degree two vertex v. replacing (u, v) and (v, w) by (u, w). Any path we construct in the shortcut graph yields a longer path in the original graph. If the original graph had the uniform endpoint distribution property, then so does the new graph. If the number of vertices in the shortcut graph is less than $n^{2/3}$, then there must have been a path of degree two vertices of length $n^{1/3}$ in the original graph. If not, then we find a path of length at least $n^{1/3}$ in the shortcut graph, yielding a path of at least this length in the original graph.

4. Hardness Results

It is important to keep in mind that it may be possible to achieve better performance guarantees on graphs that are known to be Hamiltonian than on arbitrary input graphs. This is analogous to the chromatic number problem where significant improvements can be obtained if the input graph is known to be 3-colorable [4]. We first consider the "easier" problem and prove that, for any constant $\epsilon < 1$, no polynomial-time algorithm can find a path of length $n - n^\epsilon$ in an n-vertex Hamiltonian graph, unless $\mathbf{P} = \mathbf{NP}$. Next we demonstrate a *self improvability* result for approximating longest paths. The self-improvability result is used to show that finding constant factor approximations to the longest path problem is **NP**-hard. We also provide evidence that it is unlikely to be the case that there exists any $o(n^\delta)$ ratio approximation algorithm for this problem. Our results extend to showing the hardness of the longest path problem even in the case of *bounded degree* graphs.

4.1. Hardness Result for Hamiltonian Graphs

The following theorem easily generalizes to the approximation of longest paths in arbitrary (non-Hamiltonian) graphs. But the subsequent results are much stronger for that problem.

Theorem 7: *For any $\epsilon < 1$, the problem of finding a path of length $n - n^\epsilon$ in a Hamiltonian graph is* **NP***-complete.*

We present only a sketch of the proof. Let $G = (V, E)$ be a Hamiltonian graph on n vertices, and define $K = (n+2)^{\epsilon/(1-\epsilon)}$. We define an *auxiliary graph* $G' = (V', E')$. Informally, G' consists of K *copies* of G "glued" together in a cycle by K "special" vertices. The K copies of G are arranged in a cycle and any two successive copies of G are separated by a special vertex that is connected to all the vertices of both the copies. In more formal terms, $V' = (V \cup \{v_s\}) \times \{1, 2, \ldots, K\}$, where v_s is a special vertex. Any two vertices (v_1, k_1) and (v_2, k_2) in V' share an edge in E' if and only if one of the following conditions hold.

- $k_1 = k_2$ and $(v_1, v_2) \in E$.
- $v_1 = v_s$ and $k_2 = (k_1 + 1) \pmod{K}$.
- $v_1 = v_s$ and $k_2 = (k_1 - 1) \pmod{K}$.

Let A be an algorithm that is guaranteed to find a path of length at least $n - n^{\epsilon}$ in a Hamiltonian graph on n vertices. Consider what happens when we run algorithm A on G'. We claim that A finds a Hamiltonian path in at least one of the copies of G in G'. Since there are only a polynomial number of copies of G in G', this gives us a polynomial-time algorithm to find a Hamiltonian path in G.

To see this claim, we first observe that any simple path in G' is a disjoint collection of simple paths in copies of G "stitched" together by special vertices. In this collection, there can be at most one simple path per copy of G (except the first or last visited copies) since there is exactly one special vertex to enter a copy of G and one special vertex to leave the copy, and these vertices cannot be visited twice in a simple path. There can be at most two simple non-disjoint paths from either the first or the last visited copies of G (but not both). For instance, the path P may start within the first copy of G and use a special vertex to exit and then "turn back" into a second disjoint path within the same copy before moving out through the second special vertex into the second copy of G and so on.

Now, since $|V'| = (n+1)K$, by running A on G' we obtain a path P of length l, where

$$l > (n+1)K - ((n+1)K)^{\epsilon} > (n+1)K - (n+2)^{(1+\epsilon/(1-\epsilon))\epsilon} + 1 = (n+1)K - K + 1.$$

In other words, P includes all but at most $K - 2$ vertices of G'. Since there are K copies of G, we can conclude that P includes *all* the vertices of at least two copies of G. All the vertices of at least one of these copies of G must occur contiguously within P. Hence the path P includes a Hamiltonian path from that copy of G.

4.2. Self-Improvability and Hardness Results for Longest Paths

We now show that if the longest path can be approximated to some constant k in polynomial-time, then we can approximate it to *any* constant k', also in polynomial-time. To this end, we define the following novel definition of a graph product.

Definition 4: *For a graph $G(V, E)$, its* **edge square** *graph $G^2(V^2, E^2)$ is obtained as follows. Replace each edge $e = (u, v)$ in G by a copy of G, call it G_e, and connect both u and v to each vertex in G_e. The vertices u and v are referred to as the* **contact** *vertices of G_e.*

If the graph G has n vertices, then G^2 will have at most n^3 vertices. Observe that the edge square of a Hamiltonian graph need not be Hamiltonian. It is precisely for this reason that the subsequent results apply only to the more general problem of approximating the longest path in arbitrary graphs, as opposed to the possible easier problem where the input is guaranteed to be Hamiltonian. The following lemma relates the length of the longest paths in G and G^2.

Lemma 5: *Let $G = (V, E)$ be any graph. If the longest simple path in G has length l then the longest simple path in G^2 has length at least l^2. Moreover, given a path of length m in G^2, we can obtain in polynomial-time a path of length $\sqrt{m} - 1$ in G.*

Proof: First, we exhibit a path P^2 of length l^2 in G^2. Suppose that a longest path in G is the path P with end-points x and y. The path P^2 traverses the edge copies G_e in exactly the same order as the edges in P. Moreover, in each G_e being traversed, it traverses exactly the path corresponding to P. It is easy to see that the resulting path is of length $l(l+2)$.

Consider any path Q of length m in G^2. We show that it is possible to derive a path of length at least $\sqrt{m}-1$ in G. The path Q can enter or leave a given copy of G in G^2 only via its contact vertices. Therefore, all the vertices of any copy of G occur contiguously in Q, except possibly for the first and the last copy visited by Q. In other words, the path Q starts off in some copy G_e, and then visits a sequence of copies that are all distinct, except that it could finish off back inside the copy G_e. The sequence of vertices visited inside any particular edge copy forms a simple path in G, except that in G_e there could be two disjoint simple paths. We claim that one of the following holds.

1. One of the simple paths inside the edge copies visited by Q is of length at least $\sqrt{m}-1$.
2. The number r of distinct copies of G traversed by Q is at least $\sqrt{m}-1$.

The proof is by contradiction. Let r be the number of distinct copies of G visited, and let s be the length of the longest path traversed in any edge copy. Observe that the total number of copies visited could be $r+1$, since the first and the last copy visited may be identical. In terms of r and s, the length of Q must be at most $(r+1)s+r$. Assume both the above conditions are violated, i.e. $r<\sqrt{m}-1$ and $s<\sqrt{m}-1$. Then, we obtain that

$$\begin{aligned} m &\leq rs++s+r \\ &< (m-2\sqrt{m}+1)+(\sqrt{m}-1)+(\sqrt{m}-1) \\ &= m-1 \end{aligned}$$

This gives a contradiction.

Given the path Q, it is fairly easy to construct a path of length s in G, and also a path of length $r-1$. This gives the desired result.
□

We apply this lemma to obtain the following theorem.

Theorem 8: *If the longest path problem has a polynomial-time algorithm that achieves a constant factor approximation, then it has a PTAS.*

Proof: Let A_k be a polynomial-time approximation algorithm with a performance ratio of $k>1$, i.e. it obtains a path of length l/k in a graph with longest path length l. Let p be the smallest integer exceeding

$$\log\frac{2\log k}{\log(1+\epsilon)}$$

Given an input graph G with longest path length l, suppose we run the algorithm A_k on the graph G^{2^p} obtained by squaring G repeatedly p times; this yields a path of length at least l^{2^p}/k in G^{2^p}. Using Lemma 5, a simple calculation shows that we obtain in G a path of length at least

$$\left(\frac{l^{2^p}}{k}\right)^{1/2^p}-p \geq \frac{l}{\sqrt{1+\epsilon}}-p \geq \frac{l}{1+\epsilon}$$

provided $l \geq 2p(1+\epsilon)/\epsilon$. Observe that for small l, we can compute the optimal solution by brute-force in polynomial time. Moreover, the running time of the resulting algorithm is polynomial for fixed ϵ, since the graph G^{2^p} has at most n^{3^p} vertices. This gives the desired PTAS.
□

We now show the non-existence of a PTAS for longest paths. The proof is based on the recent results of Arora et al. [1].

Theorem 9: *There is no PTAS for the longest path problem, unless* $\mathbf{P} = \mathbf{NP}$.

Proof: We first claim that it suffices to demonstrate the hardness result when the longest path problem is restricted to instances containing a Hamiltonian *cycle*. Clearly, if it is **NP**-hard to find a constant factor approximation to the longest path in graphs with a Hamiltonian cycle, then this is also the case for the more general problem of approximating the longest path in arbitrary graphs.

Let us denote by TSP(1,2) the problem of finding an optimal traveling salesman tour in a complete graph where all edge lengths are either 1 or 2. The approximation version of this problem has been shown to be **MAX SNP**-hard by Papadimitriou and Yannakakis [24]. Their reduction is from a version of the MAX 3SAT problem which is also **MAX SNP**-complete. We claim that the following statement can be obtained as a consequence of their result. Suppose that for every $\delta > 0$ there is a polynomial-time algorithm that on any instance of TSP(1,2) with optimum value n returns a tour of cost at most $(1+\delta)n$, then MAX 3SAT has a PTAS. Observe that having an optimum solution of size n corresponds to having a Hamiltonian cycle using only the edges of length 1.

Suppose now that we have a PTAS for the longest path problem in Hamiltonian graphs. In particular, this implies that we can find paths of length at least $(1-\delta)n$, for any fixed $\delta > 0$, in graphs that possess a Hamiltonian path. Clearly, this works equally well with graphs that possess a Hamiltonian cycle. Given any instance of TSP(1,2) with optimum value n, the edges of weight 1 form a Hamiltonian graph. Using the PTAS for longest paths, we can construct a path of length $(1-\delta)n$ using only the edges of weight 1. This can be converted into a tour of weight at most $(1+\delta)n$ in the instance of TSP(1,2) by extending the path with the edges of length 2. Thus, the PTAS for the longest path problem can be used to obtain a PTAS for such instances of TSP(1,2).

But the results of Arora et al. [1] show that if any **MAX SNP**-hard problem has a PTAS, then $\mathbf{P} = \mathbf{NP}$. By the reduction of Papadimitriou and Yannakakis, we may conclude that MAX 3SAT has a PTAS, and therefore $\mathbf{P} = \mathbf{NP}$.
□

The self-improvability result implies the **NP**-hardness of constant factor approximations to longest paths. Unfortunately, this result does not hold for the case of Hamiltonian graphs since the edge product of a graph with itself does not preserve Hamiltonicity. The next corollary follows by combining Theorems 8 and 9.

Corollary 2: *There does not exist a constant factor approximation algorithm for the longest path problem, unless* $\mathbf{P} = \mathbf{NP}$.

Since finding constant factor approximations to the longest path problem is hard, the next step would be to try to find weaker approximations. The following theorem provides evidence that finding such weaker approximations may also be very difficult. The proof uses the edge product to amplify the gap in the longest path lengths.

Theorem 10: *If there is a poly-time algorithm for the longest path problem with a performance ratio of* $2^{\mathrm{O}(\sqrt{\log n})}$*, then* $\mathbf{NP} \subseteq \mathbf{DTIME}\left(2^{\mathrm{O}(\log^5 n)}\right)$.

Proof: Suppose that a polynomial-time algorithm A can approximate the longest path to within the specified ratio. Let $G = (V, E)$ be any instance of the longest path problem with n vertices and longest path length l. Choose p to be the smallest integer such that $N = n^{3^p} = 2^{\log^5 n}$. Notice that $3^p = \log^4 n$, and that $2^p \geq \log^{2.5} n$.

We use the edge squaring operation defined earlier to produce the graph G^2. We apply this operation repeatedly p times, to obtain the graph G^{2^p} with a longest path length of at least l^{2^p}. The algorithm A finds a path of length at least $l^{2^p}/f(N)$, where $f(N) = 2^{\mathrm{O}(\sqrt{\log N})}$. Using a repeated application of Lemma 5, we can then obtain a path of length at least $\frac{l}{g(n)} - p$ in G, such that

$$g(n) = f(N)^{1/2^p} = 2^{\mathrm{O}\left(\frac{\sqrt{\log N}}{2^p}\right)} = 2^{\mathrm{O}\left(\frac{\log^{2.5} n}{2^p}\right)} = \mathrm{O}(1)$$

Since $p = \mathrm{O}(\log\log n)$, we obtain a constant factor approximation for $l = \Omega(\log n/\log\log n)$; otherwise, Monien's algorithm can be used to find the exact solution. This implies that the longest path in G can be approximated within a constant factor in time poly$(N) = 2^{\mathrm{O}(\log^{1/\epsilon} n)}$. But we already know that finding a constant factor approximation to the longest path is **NP**-complete. Thus, we obtain the desired simulation of **NP**.
□

The previous theorem can be strengthened if one can define an edge squaring operation on graphs that only squares the number of vertices, instead of cubing it as currently defined. We achieve the same effect by considering a restricted class of graphs, namely graphs of bounded degree. Since the number of edges in such graphs is linear, the number of vertices in the edge squared graph is quadratic instead of cubic. All the above hardness results can also be extended to the case where the input graphs are of bounded degree.

We briefly explain the reason here. The TSP(1,2) reduction of Papadimitriou and Yannakakis also applies to the case where the edges of length 1 induce a bounded degree graph. Recall from Lemma 5 that in the graph G^2, the only vertices that had their degrees increased were the contact vertices u and v, In the bounded degree case, we need to modify the construction of the graph G^2 so that the maximum degree of the vertices remains the same during the squaring operation, while maintaining the longest-path properties. To achieve this, we consider the problem of approximating the longest path between a specified pair of vertices, say s and t. A hardness result for this problem can be easily extended to the more general longest path problem. Now, instead of using a "fan-out" from vertices u and v to $G_{(u,v)}$, we connect vertices u and v only to vertices s and t of $G_{(u,v)}$, respectively. It can be shown that regardless of the number of squaring operations, the maximum degree in the graph increases by at most 1. The remaining argument is identical to the case of general graphs. We obtain the following theorem by using the new edge squaring operation.

Theorem 11: *For any $\epsilon > 0$, if there is a polynomial-time algorithm for approximating the longest path to a ratio of $2^{\mathrm{O}(\log^{1-\epsilon} n)}$ then* $\mathbf{NP} \subseteq \mathbf{DTIME}\left(2^{\mathrm{O}(\log^{1/\epsilon} n)}\right)$*. The theorem holds even for the special case of bounded degree graphs.*

Proof: Suppose that a polynomial-time algorithm A can approximate the longest path to within the specified ratio. Let $G = (V, E)$ be any instance of the longest path problem with n vertices and longest path length l. Choose p to be the smallest integer such that

$$N = n^{2^p} = 2^{\log^{1/\epsilon} n}$$

We use the squaring operation defined in Lemma 5 that produces the graph G^2. We apply this operation repeatedly p times, to obtain the graph G^{2^p} with a longest path length of at least l^{2^p}. The algorithm A finds a path of length at least $l^{2^p}/f(N)$, where $f(N) = 2^{\mathrm{O}(\log^{1-\epsilon} N)}$. Using a repeated application of Lemma 5, we can then obtain a path of length at least $\frac{l}{g(n)} - p$ in G, such that

$$\begin{aligned} g(n) &= f(N)^{1/2^p} \\ &= 2^{\mathrm{O}\left(\frac{\log^{1-\epsilon} N}{2^p}\right)} \\ &= 2^{\mathrm{O}\left(\frac{\log^{1-\epsilon} n}{2^{\epsilon p}}\right)} \\ &= 2^{\mathrm{O}(1)} = \mathrm{O}(1) \end{aligned}$$

Since $p = \mathrm{O}(\log\log n)$, we obtain a constant factor approximation for $l = \Omega(\log n/\log\log n)$; otherwise, Monien's algorithm can be used to find the exact solution. This implies that the longest path in G can be approximated within a constant factor in time $\mathrm{poly}(N) = 2^{\mathrm{O}(\log^{1/\epsilon} n)}$. But we already know that finding a constant factor approximation to the longest path is **NP**-complete. Thus, we obtain the desired simulation of **NP**.
□

The argument used in the proof of this theorem can be easily extended to show the following.

Theorem 12: *For any $\delta > 0$, if there is a polynomial-time algorithm for approximating the longest path to a ratio of $2^{O\left(\frac{\log n}{\log\log n}\right)}$ then* $\mathbf{NP} \subseteq \mathbf{DTIME}\left(2^{O(n^\delta)}\right)$.

Proof: Let $G = (V, E)$ be any graph. Let l be the length of its longest path. We use the squaring operation defined in Lemma 5 that produces the graph G^2. We apply this operation repeatedly k times, to obtain the graph G^{2^k}. It is easy to verify that the longest path of G^{2^k} is of length l^{2^k}.

We proved that finding a constant factor approximation to the longest path is **NP**-complete. In particular, consider the problem of finding a path of length $l/2^{1/\delta}$ in G. By Lemma 5, it is equivalent to the problem of finding a path of length $(l/2^{1/\delta})^{2^k}$ in the graph G^{2^k}. Choose k such that $n^{2^k} = 2^{n^\delta}$.

Let A be a polynomial-time algorithm that approximates longest path to a ratio of $2^{\log n/\log\log n}$. We claim that A finds a path of the required length $(l/2^{1/\delta})^{2^k}$ in the graph G^{2^k} corresponding to the above choice of k.

To see this, let

$$N = n^{2^k} = 2^{n^\delta},$$

and let G_N be G^{2^k}. The longest path in G_N is of length l^{2^k}. Hence, A finds a path of length l^{2^k}/K, where,

$$\begin{aligned} K &= 2^{\log N/\log\log N} \\ &= 2^{n^\delta/(\delta \log n)} \\ &= n^{2^k/(\delta \log n)} \\ &= 2^{2^k/\delta} \end{aligned}$$

Therefore, A finds a path of length

$$l^{2^k}/2^{2^k/\delta} = (l/2^{1/\delta})^{2^k}$$

in G_N. Hence, by running A on G_N we can find a constant ratio approximation to Hamiltonian path in time polynomial in N. Since the constant ratio problem is **NP**-complete, it follows that $\mathbf{NP} \subseteq \mathbf{DTIME}\left(2^{n^\delta}\right)$.
□

Acknowledgements

We would like to express our gratitude to Sanjeev Arora, Yossi Azar, Sanjeev Khanna, Madhu Sudan and Mihalis Yannakakis for valuable comments. We also thank Umesh Vazirani for useful discussions during the early stages of this work.

References

[1] S. Arora, C. Lund, R. Motwani, M. Sudan and M. Szegedy, *Proof Verification and Hardness of Approximation Problems*, Proc. 33rd Annual IEEE Symp. on Foundations of Computer Science, 1992, pp. 14–23.

[2] S. Arora and S. Safra, *Approximating Clique is NP-complete*, Proc. 33rd Annual IEEE Symp. on Foundations of Computer Science, 1992, pp. 2–13.

[3] M. Bellare, *Interactive Proofs and Approximation*, IBM Research Report RC 17969, 1992.

[4] A. Blum, *Some tools for approximate 3-coloring*, Proc. 31st IEEE Symp. on Foundations of Computer Science, 1990, pp. 554–562.

[5] B. Bollobas, *Random Graphs.* Academic Press, 1985.

[6] A. Broder, A.M. Frieze and E. Shamir, *Finding hidden Hamiltonian cycles*, Proc. 23rd ACM Symp. on Theory of Computing, 1991, pp. 182–189.

[7] R. B. Boppana and M. M. Halldorsson, *Approximating maximum independent sets by excluding subgraphs*, Proc. of the 2nd Scandanavian Workshop on Algorithmic Theory, Lecture Notes in Computer Science, No. 447, Springer-Verlag, 1990, pp. 13–25.

[8] P. Berman and G. Schnitger, *On the Complexity of Approximating the Independent Set Problem*, Information and Computation, 96 (1992), pp. 77–94.

[9] V. Chvatal, *Tough graphs and Hamiltonian circuits*, Discrete Mathematics, 5 (1973), pp. 215–228.

[10] V. Chvatal, *Edmonds Polytopes and Weakly Hamiltonian Graphs*, Mathematical Programming, 5 (1973), pp. 29–40.

[11] V. Chvatal, *Hamiltonian cycles*, In *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization* (ed: E.L. Lawler et al.), 1985, pp. 402–430.

[12] R. Fagin, *Generalized First-Order Spectra and Polynomial-time Recognizable Sets*, In *Complexity of Computer Computations* (ed: Richard Karp), AMS, 1974.

[13] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy, *Approximating clique is almost NP-complete*, Proc. 32nd IEEE Symp. on Foundations of Computer Science, 1991, pp. 2–12.

[14] W. Fernandez de la Vega and G.S. Lueker, *Bin Packing can be solved within* $1+\epsilon$ *in Linear Time*, Combinatorica, 1 (1981), pp. 349–355.

[15] M. Furer and B. Raghavachari, *Approximating the Minimum Degree Spanning Tree to within One from the Optimal Degree*, Proc. 3rd Annual ACM-SIAM Symp. on Discrete Algorithms, 1992, pp. 317–324.

[16] Michael R. Garey and David S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, 1979.

[17] David S. Johnson, *The Tale of the Second Prover*, The NP-Completeness Column: An Ongoing Guide. Journal of Algorithms, 13 (1992), pp. 502–524.

[18] David R. Karger, Rajeev Motwani, and Madhu Sudan. Approximate graph coloring by semidefinite programming. In *Proceedings of the* 35^{th} *Annual Symposium on the Foundations of Computer Science*, pages 2–13. To appear in JACM.

[19] N. Karmakar and R.M. Karp, *An Efficient Approximation Scheme For The One-Dimensional Bin Packing Problem*, Proc. 23rd IEEE Symp. on Foundations of Computer Science, 1982, pp. 312–320.

[20] C. Lund and M. Yannakakis, *On the Hardness of Approximating Minimization Problems*, Proc. 25th ACM Symp. on Theory of Computing, 1993 (to appear).

[21] B. Monien, *How to find long paths efficiently*, Annals of Discrete Mathematics, 25 (1984), pp. 239–254.

[22] R. Motwani, *Lecture Notes on Approximation Algorithms*, Technical Report No. STAN-CS-92-1435, Department of Computer Science, Stanford University, 1992.

[23] C. H. Papadimitriou and M. Yannakakis, *Optimization, Approximation, and Complexity Classes*, Proc. 20th ACM Symp. on Theory of Computing, 1988, pp. 229–234.

[24] C. H. Papadimitriou and M. Yannakakis, *The traveling salesman problem with distances one and two,* Mathematics of Operations Research, to appear.

[25] A. Subramanian. *Personal communication*, 1994.