Electronic Colloquium on Computational Complexity, Report No. 32 (2003)

Approximating Longest Directed Path

Andreas Björklund* Thore Husfeldt* Sanjeev Khanna†

April 16, 2003

Abstract

We investigate the hardness of approximating the longest path and the longest cycle in directed graphs on n vertices. We show that neither of these two problems can be polynomial time approximated within $n^{1-\epsilon}$ for any $\epsilon > 0$ unless P = NP. In particular, the result holds for digraphs of constant bounded outdegree that contain a Hamiltonian cycle. Assuming the stronger complexity conjecture that Satisfiability cannot be solved in subexponential time, we show that there is no polynomial time algorithm that always finds a path of length $\Omega(\log^{2+\epsilon} n)$, or a cycle of length $\Omega(\log^{1+\epsilon} n)$, for any constant $\epsilon > 0$ in these graphs. In contrast we show that there is a polynomial time algorithm always finding a path of length $\Omega(\log^2 n/\log\log n)$ in these graphs. This separates the approximation hardness of Longest Path and Longest Cycle in this class of graphs. Furthermore, we present a polynomial time algorithm that finds paths of length $\Omega(n)$ in most digraphs of constant bounded outdegree.

1 Introduction

Given an unweighted graph or digraph $G = (V, A)$ with $n = |V|$, the *Longest Path* problem is to find the longest sequence of distinct vertices $v_1 \cdots v_k$ such that $v_i v_{i+1} \in A$. This problem is notorious for the difficulty of understanding its approximation hardness [6]. The present paper presents a number of upper and lower bounds for the directed case.

Hardness result. The best known algorithm [1] finds a path or directed path of length $\Theta(\log n)$ if it exists. In the present paper we establish a strong inapproximability bound for the *directed* version of the problem. Specifically, Theorem 1 states that in directed graphs the length of the longest path cannot be polynomial time approximated within an approximation ratio of $n^{1-\epsilon}$ for any $\epsilon > 0$ unless P = NP.

* Department of Computer Science, Lund University, Box 118, 221 00 Lund, Sweden. Email: thore@cs.lth.se.

† Dept. of CIS, University of Pennsylvania, Philadelphia, PA 19104. Email: sanjeev@cis.upenn.edu. Supported in part by an Alfred P. Sloan Research Fellowship and by an NSF Career Award CCR-0093117.

ISSN 1433-8092

We can claim a stronger bound if we make a stronger assumption called the *Exponential Time Hypothesis* (ETH), namely that Satisfiability has no subexponential time algorithms [9]: Our Theorem 2 states that for any constant $\epsilon > 0$, if there is a polynomial time algorithm that finds a dipath of length $\log^{2+\epsilon} n$ then there is an deterministic algorithm for 3-Sat with n variables with running time $2^{o(n)}$. This is relevant to the remaining open question in [1]: "Is there a polynomial time algorithm for deciding if a given graph $G = (V, E)$ contains a path of length, say, $\log^2 n$?" Even though this question remains open, Alon, Yuster, and Zwick's choice of time bound was not as capricious as their wording may suggest: any stronger algorithm than $\log^2 n$ for Longest Path would violate the Exponential Time Hypothesis.

Algorithm for Hamiltonian Digraphs. Our lower bound holds even if the input digraph is known to be Hamiltonian, which addresses the question to what extent knowledge of the presence of a long path helps in the search for one. On the other hand, in Theorem 3 we show how to efficiently find paths of length $\Omega(\log^2 n/\log\log n)$ in Hamiltonian digraphs of constant bounded outdegree; this is close to our own $\Omega(\log^{2+\epsilon} n)$ lower bound and matches the current best result for the undirected case [4, 16]. The best previous upper bound [1] was $O(\log n)$.

Algorithm for Most Sparse Digraphs. Even though our lower bounds suggest that an efficient approximation algorithm with good worst-case performance is unlikely to exist, this does not mean that all instances of the problem are difficult. Indeed, we show with Theorem 4 that a very simple algorithm finds paths of length $\Omega(n)$ in most sparse digraphs on n vertices (it works for all bounded-outdegree expanders). Previously, [11] presented a deterministic polynomial time algorithm that find paths of length $\Omega(\sqrt{n}/\log n)$, but works only for a more restricted class of graphs: most Hamiltonian cubic undirected graphs. Another related result [5] shows how to find a hidden Hamiltonian circuit fast in most undirected graphs, even restricted to graphs whose maximum degree is bounded by a large enough constant.

Longest Path versus Longest Cycle. For the related longest *cycle* problem, where we also require $v_k v_1 \in A$, Theorem 2 shows that the longest cycle cannot be approximated within $O(\log^{1+\epsilon} n)$. This lower bound, together with the longest path guarantee $\Omega(\log^2 n/\log\log n)$ from Theorem 3, separates the complexities of the Longest Path and Longest Cycle problems, at least for the directed, bounded outdegree, Hamiltonian case, and assuming ETH.

Related work. Among the canonical NP-hard problems, the *undirected* version of this problem has been identified as the one that is least understood [6]. The best known guarantee is $O\big((\log n/\log\log n)^2\big)$ [4], slightly better than for the directed case, and a recent result establishes a $O\big(n^{\alpha}\big)$ bound for graphs of maximum degree three [6]. However, it remains fair to say that also in undirected graphs, Longest Path does not seem to admit good approximation algorithms. Indeed, it has been conjectured [11] that the length of a longest path in undirected graphs cannot be approximated within n^α for some $\alpha > 0$ unless P = NP, a somewhat weaker bound than the one we prove for digraphs, but this is far from being proved: the quoted reference shows that the Longest Path is not in APX, and that no polynomial time algorithm can approximate the length of the longest path within $2^{\log^{1-\epsilon} n}$ for any $\epsilon > 0$ unless $\mathrm{NP} \subseteq \mathrm{DTIME}\big(2^{\log^{O(1/\epsilon)} n}\big)$. It is believed to be much harder to find paths and cycles in directed graphs than in undirected, and our result may be seen to give some credit to this position. Our lower bound is stronger than the best current result for the undirected case, and the proof is elementary.

The central question of closing the gap between the upper and the lower bound for *undirected* graphs remains open. Our lower bound uses a reduction to the *k Vertex Disjoint Paths* problem in digraphs. Thus there is no direct way to translate our argument to the undirected case, because the problem is known to be polynomially solvable for undirected graphs [14].

Some of the results in the present paper have been announced by the authors in [12] and [3].

2 Preliminaries

We write uv for the arc (u, v). The vertex set V is sometimes identified with $\{1, 2, \ldots, n\}$. For a subset $W \subseteq V$ of the vertices of a graph G, we denote by $G[W]$ the graph induced by W.

2.1 Two Vertex Disjoint Paths

Our proof starts in a reduction from a problem known to be NP-complete for over twenty years. In the *k Vertex Disjoint Paths* problem we are given a digraph G of order $n > 2k$, and we are asked whether there exists a set of k vertex disjoint paths in G such that the ith path connects vertex $2i - 1$ to vertex $2i$, for $i = 1, \ldots k$. This problem is NP-complete [7] even when $k = 2$. We need to modify this result slightly to see that it is valid even if we restrict the 'yes'-instances to be partitionable into two disjoint paths. To be precise, we define the *Two Vertex Disjoint Paths* problem (2VDP): given a digraph G of order $n \geq 4$, decide whether there exists a pair of vertex disjoint paths, one from 1 to 2 and one from 3 to 4. We study the *restricted* version of this problem (R2VDP), where the 'yes'-instances are guaranteed to contain two such paths that together exhaust all vertices of G. In other words, the graph G with the additional arcs 23 and 41 contains a Hamiltonian cycle through these arcs.

Proposition 1 *Restricted Two Vertex Disjoint Paths is NP-complete.*

The proof is an extension of the construction in [7] and can be found in Sec. 7. It replaces a reduction from 3-Sat by a reduction from Monotone 1-in-3-Sat, and uses a more intricate clause gadget to guarantee the existence of two paths that cover all vertices. The modification is necessary to prove the lower bound for Longest Path even for Hamiltonian instances.

3 Long Paths Find Vertex Disjoint Paths

We will use instances of R2VDP to build graphs in which long paths must reveal a solution to the original problem. Given an instance $G = (V, A)$ of R2VDP, define $T_d[G]$ as a graph made up out of $m = 2^d - 1$ copies $G_1 \cdots G_m$ of G arranged in a balanced binary tree structure. For all $i < 2^{d-1}$, we say that the copies G_{2i} and G_{2i+1} are the left and right *child* of the copy G_i. The copy G_1 is the *root* of the tree, and G_i for $i \geq 2^{d-1}$ are the *leaves* of the tree. The copies of G in $T_d[G]$ are connected by additional arcs as follows. For every copy G_i having children, three arcs are added (cf. Fig. 1):

- One arc from 2 in G_i to 1 in G_{2i}.
- One arc from 4 in G_{2i} to 1 in G_{2i+1}.
- One arc from 4 in G_{2i+1} to 3 in G_i.

Moreover, in every leaf copy G_i ($i \geq 2^{d-1}$) we add the arc 23, and in the root G_1 we add the arc 41.

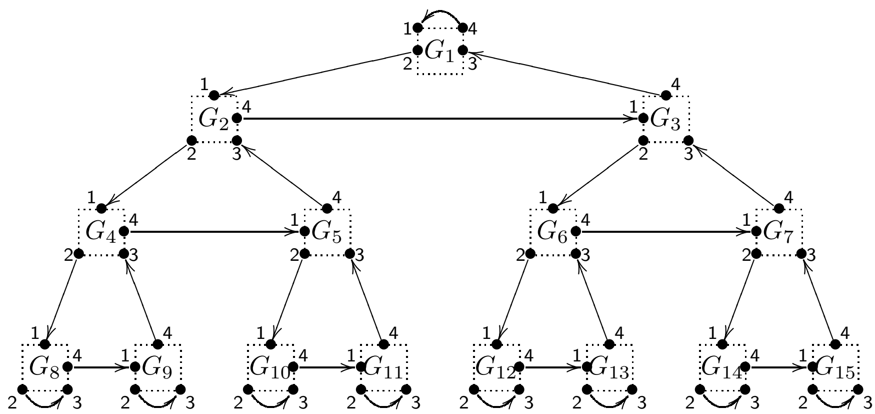

Figure 1: $T_4[G]$.

Lemma 1 *Given an instance $G = (V, A)$ of R2VDP on $n = |V|$ vertices, and any integers $m = 2^d - 1 > 3$, consider $T_d[G]$ with $N = mn$ vertices. Then*

- *If G has a solution then $T_d[G]$ contains a path of length $N - 1$.*
- *Given any path of length larger than $(4d - 5)n$ in $T_d[G]$, we can in time polynomial in N construct a solution to G.*

Proof. For the first part of the lemma, consider a solution for G consisting of two disjoint paths P and Q connecting 1 to 2 and 3 to 4, respectively, such that $P + 23 + Q + 41$ is a Hamiltonian cycle in G. The copies of P and Q in all G_is together with the added arcs constitute a Hamiltonian cycle in $T_d[G]$ of length mn and thus a path of the claimed length.

For the second part, first consider an internal copy G_i and observe that if a path traverses all of the four arcs connecting G_i to the rest of the structure then this path constitutes a solution to R2VDP for G. Thus we can restrict our attention to paths in $T_d[G]$ that avoid at least one the four external arcs of each internal G_i; we call such paths *avoiding*.

Given $T_d[G]$ define $e_d[G]$ as the length of the longest avoiding path in $T_d[G]$ ending in vertex 4 of its root copy, and $s_d[G]$ as the length of the longest avoiding path starting in vertex 1 of the root copy. Consider a path P ending in vertex 4 of the root copy, for $d > 1$. At most n vertices of P are in G_1. The path P has entered G_1 via vertex 3 from G_3's vertex 4. There are two possibilities. Either the first part of P is entirely in the subtree rooted at G_3, in which case P has length at most $n + e_{d-1}[G]$. Or it entered G_3 via 1 from the subtree rooted at G_2, in which case it may pass through at most n vertices in G_3, amounting to length at most $2n + e_{d-1}[G]$. (Especially, P cannot leave via G_3's vertex 2, because then it wouldn't be avoiding). A symmetric argument for $s_d[G]$ for $d > 1$ shows an equivalent relation. Thus we have that

$$\begin{aligned} e_1[G] &\leq n, \quad e_{d+1}[G] \leq 2n + e_d[G], \\ s_1[G] &\leq n, \quad s_{d+1}[G] \leq 2n + s_d[G]. \end{aligned}$$

Furthermore, note that a longest avoiding path in $T_d[G]$ connects a path amounting to $e_{d-1}[G]$ in the right subtree, through a bridge consisting of as many vertices as possible in the root, with a path amounting to $s_{d-1}[G]$ in the left subtree. Consequently, a typical longest avoiding path starts in a leaf copy of the right subtree, travels to its sister copy, goes up a level and over to the sister of that copy, continues straight up in this zigzag manner to the root copy, and down in the same fashion on the other side. Formally, the length of a longest avoiding path in $T_d[G]$ for $d > 1$ is bounded from above by $e_{d-1}[G] + n + s_{d-1}[G] \leq (4d - 5)n$. ∎

Theorem 1 *There can be no deterministic, polynomial time approximation algorithm for Longest Path or Longest Cycle in a Hamiltonian directed graph on n vertices with performance ratio $n^{1-\epsilon}$ for any fixed $\epsilon > 0$, unless P = NP.*

Proof. First consider the path case. Given an instance $G = (V, A)$ of R2VDP with $n = |V|$, fix $k = 1/\epsilon$ and construct $T_d[G]$ for the smallest integers $m = 2^d - 1 \geq (4dn)^k$. Note that the graph $T_d[G]$ has order $N = n^{O(k)}$. Assume there is a deterministic algorithm finding a long path of length l_{apx} in time polynomial in N, and let l_{opt} denote the length of a longest path. Return 'yes' if and only if $l_{\text{apx}} > (4d - 5)n$. To see that this works note that if G is a 'yes'-instance and if indeed $l_{\text{opt}}/l_{\text{apx}} \leq N^{1-\epsilon}$ then $l_{\text{apx}} > (4d - 5)n$, so Lem. 1 gives a solution to G.

If on the other hand G is a 'no'-instance then the longest path must be *avoiding* as defined in the proof of Lem. 1, so its length is at most $(4d-5)n$. Thus we can solve the R2VDP problem in polynomial time, which by Prop. 1 requires P = NP.

For the cycle case, we may use a simpler construction. Simply connect m copies $G_1, \cdots, G_m$ of G on a string, by adding arcs from vertex 2 in G_i to vertex 1 in G_{i+1}, and arcs from vertex 4 in G_i to vertex 3 in G_{i-1}. Finally, add the arc 41 in G_1 and the arc 23 in G_m. The resulting graph has a cycle of length mn whenever G is a 'yes'-instance, but any cycle of size at least $2n+1$ must reveal a solution to G. ∎

4 Subexponential Algorithms for Satisfiability

In this section we show that a superlogarithmic dipath algorithm implies subexponential time algorithms for Satisfiability.

We need the well-known reduction from Monotone 1-in-3-Sat to 3-Sat. It can be verified that the number of variables in the construction (see also [13, Exerc. 9.5.3]) is not too large:

Lemma 2 ([15]) *Given a 3-Sat instance φ with n variables and m clauses we can construct an instance of Monotone 1-in-3-Sat with $O(m)$ clauses and variables that is satisfiable if and only if φ is.*

The next lemma is a variant of Theorem 1.

Lemma 3 *There is a deterministic algorithm for Monotone 1-in-3-Sat on n variables running in time $2^{O(n^{1/(1+\epsilon)})}$, if there is*

1. *a polynomial time deterministic approximation algorithm A_{LP} for Longest Path in N-node Hamiltonian digraphs with guarantee $\log^{2+\epsilon} N$, or*

2. *a polynomial time deterministic approximation algorithm A_{LC} for Longest Cycle in N-node Hamiltonian digraphs with guarantee $\log^{1+\epsilon} N$.*

Proof. We need to verify that our constructions obey the necessary size bounds. The R2VDP instance build from the instance to Monotone 1-in-3-Sat described in Sec. 7 has size $n' = O(n)$.

For the path case, set $d = (4n')^{1/(1+\epsilon)}$ and construct $T_d[G]$ as in Sec. 3, which will have $N = (2^d - 1)n'$ nodes. Run the algorithm A_{LP} on $T_d[G]$. Observe that $(4d-5)n' < \log^{2+\epsilon}((2^d-1)n')$, so Lem. 1 tells us how to use A_{LP} to solve the R2VDP instance, and hence the 1-in-3-Sat instance.

The cycle case follows in a similar fashion from the construction in the proof of Theorem 1. ∎

Theorem 2 *There is a deterministic algorithm for 3-Sat on n variables running in time $2^{o(n)}$ if there is*

1. *a polynomial time deterministic approximation algorithm for Longest Path in N-node Hamiltonian digraphs with guarantee* $\log^{2+\epsilon} N$, *or*

2. *a polynomial time deterministic approximation algorithm for Longest Cycle in N-node Hamiltonian digraphs with guarantee* $\log^{1+\epsilon} N$.

Proof. The previous two lemmas give an algorithm that runs in time $2^{o(m)}$; which implies a $2^{o(n)}$-algorithm by the Sparsification Lemma of [10]. ∎

5 Finding Long Paths in Hamiltonian Digraphs

Vishwanathan [16] presents a polynomial time algorithm that finds a path of length $\Omega(\log^2 n/\log\log n)$ in *undirected* Hamiltonian graphs with constant bounded degree. We show in this section that, after a minor extension to the argument, the algorithm and its analysis apply to the directed case as well.

Theorem 3 *There is a polynomial time algorithm always finding a path of length* $\Omega(\log^2 n/\log\log n)$ *in any Hamiltonian digraph of constant bounded outdegree on n vertices.*

To prove the theorem, we need some additional notation. Let $G = (V, A)$ be a digraph. We say that a vertex $v \in V$ *spans* the subgraph $G_v = G[V_v]$ where $V_v \subseteq V$ is the set of vertices reachable from v in G. Consider the algorithm below. It takes a digraph $G = (V, A)$ on $n = |V|$ vertices and a specified vertex $v \in V$ as input, and returns a long path starting in v.

1. Enumerate all paths in G starting in v of length $\log n$, if none return the longest found.

2. For each such path $P = (v, \cdots, w)$, let V_w be the set of vertices reachable from w in $G[V - P + \{w\}]$.

3. Compute a depth first search tree rooted at w in $G[V_w]$.

4. If the deepest path in the tree is longer than $\log^2 n$, return this path.

5. Otherwise, select the enumerated path P whose end vertex w spans as large a subgraph as possible after removal of $P - \{w\}$ from the vertex set, i.e the path maximising $|V_w|$.

6. Search recursively for a long path R starting from w in $G[V_w]$, and return $(P - \{w\}) + R$.

First note that the algorithm indeed runs in polynomial time. The enumeration of all paths of length $\log n$ takes no more than polynomial time since the outdegree is bounded by a constant k, and thus there cannot be more than $k^{\log n}$ paths. Computing a depth first search tree is also a polynomial time task, and it is seen to be performed a polynomial number of times, since the recursion does not branch at all.

To prove that the length of the resulting path is indeed $\Omega(\log^2 n/\log\log n)$, we need to show that at each recursive call of the algorithm, there is still a long enough path starting at the current root vertex.

Lemma 4 *Let $G = (V, A)$ be a Hamiltonian digraph. Let $S \subseteq V, v \in V \setminus S$. Suppose that on removal of the vertices of S, v spans the subgraph $G_v = (V_v, A_v)$ of size t. If each vertex $w \in V_v$ is reachable from v on a path of length less than d, then there is a path of length $t/d|S|$ in G_v starting in v.*

Proof. Consider a Hamiltonian cycle C in G. The removal of S cuts C into at most $|S|$ paths $P_1 \cdots P_{|S|}$. Since each vertex in V lies on C, the subgraph G_v must contain at least $t/|S|$ vertices W from one of the paths, say P_j. In fact, G_v must contain a path of length $t/|S|$, since the vertex in W first encountered along P_j implies the presence in G_v of all the subsequent vertices on P_j, and these are at least $|W|$. Denote one such path by $P = p_0 \cdots p_{|W|-1}$, and let $R = r_0 \cdots r_{l-1}$ be a path from $r_0 = v$ to $r_{l-1} = p_0$, of length $l \leq d$. Set $s = |P \cap R|$ and enumerate the vertices on P from 0 to $|W| - 1$ and let $i_1 \cdots i_s$ denote the indices of vertices in $P \cap R$, in particular $i_1 = 0$. Let $i_{s+1} = |W|$. An averaging argument shows that there exists j, such that $i_{j+1} - i_j \geq |W|/s$. Let q be the index for which $r_q = p_{i_j}$. The path along R from r_0 to r_q and continuing along P from p_{i_j+1} to $p_{i_{j+1}-1}$ has the claimed length. ∎

Observe that the algorithm removes no more than $\log n$ vertices from the graph at each recursive call. Thus, at call i we have removed at most $i \log n$ vertices from the original graph; the very same vertices constituting the beginning of our long path. Lemma 4 tells us that we still are in a position were it is possible to extend the path, as long as we can argue that the current end vertex of the path we are building spans large enough a subgraph. Note that whenever we stand at a vertex v starting a long path P of length $> \log n$ in step 1 of the algorithm, the path consisting of the first $\log n$ vertices of P is one of the paths of length $\log n$ being enumerated. This is our guarantee that the subgraph investigated at the next recursive call is not all that smaller than the graph considered during the previous one. It must consist of at least $|P| - \log n$ vertices. Of course, we cannot be sure that exactly this path is chosen at step 5, but this is of no concern, since it is sufficient for our purposes to assure that there are still enough vertices reachable.

Formally, let V_i denote the vertex set of the subgraph considered at the recursive call i. In the beginning, we know that regardless of the choice of start vertex v, we span the whole graph and thus $V_0 = V$, and furthermore, that a path of length n starts in v. Combining the preceding discussion with Lem. 4, we establish the following inequality for the only non-trivial case that no path of length $\log^2 n$ is ever found during step 4 of the algorithm:

$$|V_{i+1}| > \frac{|V_i|}{i \log^3 n} - \log n$$

It is readily verified that $|V_i| > 0$ for all $i < c \log n / \log\log n$ for some constant c, which completes the proof of Theorem. 3.

6 Finding Long Paths in Most Sparse Digraphs

In this section we show that in a sparse *expander* graph, a relatively long path is easily found. These ideas are inspired by the analysis of the low expected time for finding a Hamiltonian cycle in a random graph [2, 8].

A digraph $G = (V, A)$ on n vertices is a c-expander if $|\delta U| \geq c(1 - \frac{|U|}{n})|U|$ for every subset $U \subset V$ where $\delta U = \{\, v \notin U \mid \exists u \in U : uv \in A \,\}$. A standard probabilistic argument shows that with high probability a random digraphs with outdegree k ($k > 2$), are c_k-expanders for some constant c_k, for large enough $n > n_k$. In other words, the result holds for most bounded outdegree graphs.

We propose the following algorithm for finding a long path $p_0 \cdots p_l$ in a sparse expander.

1. Pick an arbitrary start vertex p_0, and set $i = 0$.
2. Let $G_i = (V_i, A_i)$ be the subgraph spanned by p_i in $G[V \setminus (\bigcup_{j=0}^{i-1} p_j)]$.
3. If G_i consists only of p_i, exit.
4. For each neighbour v of p_i in G_i, evaluate the size of the subgraph spanned by v in $G_i[V_i \setminus p_i]$.
5. Choose the neighbour who has the largest spanned subgraph as p_{i+1}.
6. Set $i = i + 1$ and goto 2.

Theorem 4 *The algorithm finds a path of length $\frac{c}{2(k+1)}n$ in every c-expander digraph $G = (V, A)$ with maximum outdegree k.*

Proof. Consider step i. Enumerate the neighbours of p_i in G_i as $r_1 \cdots r_{k'}$. Let $V_i[r_j]$ be the vertices reachable from r_j in $G_i[V_i - \{p_i\}]$. Now observe that the $V_i[r_j]$ either are very small or really large for small i, since the set of vertices outside $V_i[r_j]$ in G which are directly connected by an arc from a vertex in $V_i[r_j]$ must lie on the prefix path $p_0 \cdots p_i$ by definition, and there must be a lot of them because of the expander criterion. Specifically, when i is small, there must be a j for which $V_i[r_j]$ is large, since $k' \leq k$ and $\bigcup V_i[r_j] = V_i - \{p_i\}$. Observe that V_{i+1} is the largest $V_i[r_j]$, to obtain

$$|V_{i+1}| \geq n - \frac{2(i+1)}{c}$$

whenever at least one $V_i[r_j]$ is too large to be a small subgraph, i.e. as long as

$$\frac{c(|V_i| - 1)}{2k} \geq i + 1,$$

where we for the sake of simplicity have used the expansion factor $c/2$ which holds for all set sizes. Observing that $V_0 = n$, we may solve for the smallest i, when the inequality above fails to hold. This will not happen unless $i \geq \frac{c}{2(k+1)}n$, as promised. ∎

7 Proof of Proposition 1

We review the construction in [7], in which the *switch* gadget from Fig. 2 plays a central role. Its key property is captured in the following statement.

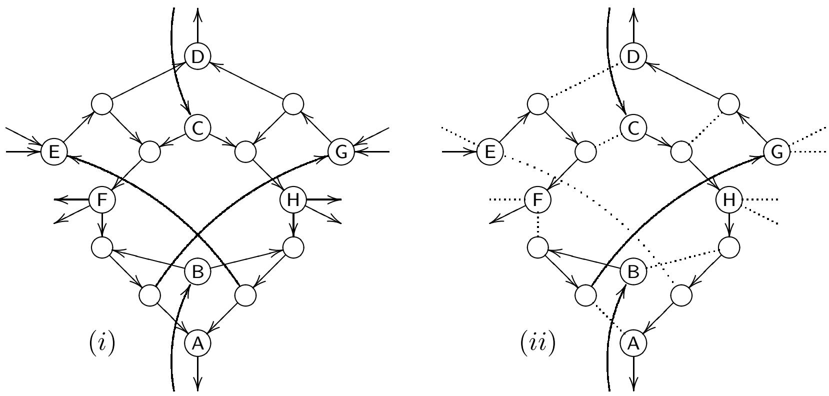

Figure 2: (i) A switch. Only the labelled vertices are connected to the rest of the graph, as indicated by the arrows. (ii) Three vertex-disjoint paths through a switch.

Lemma 5 ([7]) *Consider the subgraph in Fig. 2. Suppose that are two vertex disjoint paths passing through the subgraph—one leaving at A and the other entering at B. Then the path leaving A must have entered at C and the path entering at B must leave at D. Furthermore, there is exactly one additional path through the subgraph and it connects either E to F or G to H, depending on the actual routing of the path leaving at A.*

Also, if one of these additional paths is present, all vertices are traversed.

To prove Prop. 1 we reduce from Monotone 1-in-3-Satisfiability, rather than 3-Satisfiability as used in [7]. An instance of 1-in-3-Sat is a Boolean expression in conjunctive normal form in which every clause has three literals. The question is if there is a truth assignment such that in every clause, exactly one literal is true. It is known that even when all literals are positive (*Monotone* 1-in-3-Sat) the problem is NP-complete [15].

Given such an instance φ with clauses $t_1, \ldots, t_m$ on variables $x_1, \ldots, x_n$ we construct and instance G_φ of R2VDP as follows.

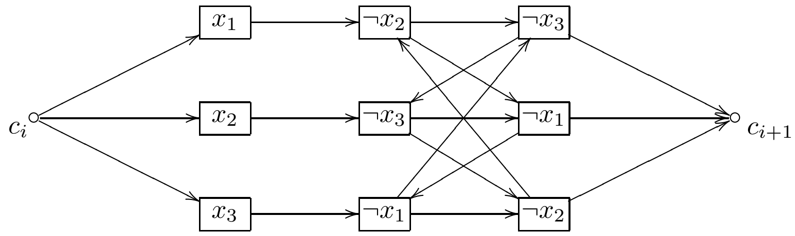

Figure 3: A clause gadget consisting of 9 switches. Every incoming arc to a switch ends in the switch's vertex E, and every outgoing arc leaves the switch's vertex F.

Clause gadgets. Every clause t_i is represented by a gadget consisting of a vertex c_i and nine switches, three for every literal in t_i. Consider the clause $t_i = (x_1 \vee x_2 \vee x_3)$. The vertices c_i, c_{i+1} and the E and F vertices in the nine switches are connected as shown in Fig. 3. Thus all clause gadgets are connected on a string ending in a dummy vertex c_{m+1}.

The clause gadget has the following desirable properties: Call a path from c_i to c_{i+1} *valid* if it is consistent with a truth assignment to $\{x_1, x_2, x_3\}$ in the sense that if it passes through a switch labelled with a literal (like $\neg x_2$) then it cannot pass through its negation (like x_2). The following claims are easily verified:

Lemma 6 *Consider the construction in Fig. 3.*

1. *Every valid path from c_i to c_{i+1} corresponds to a truth assignment to $\{x_1, x_2, x_3\}$ that sets exactly one variable to true.*

2. *If there is a truth assignment to $\{x_1, x_2, x_3\}$ that sets exactly one variable to true then there is a valid path from c_i to c_{i+1} corresponding to the assignment. Moreover, there is such a valid path passing through all five switches whose labels are consistent with the assignment.*

Variable gadgets. Every variable x_i is represented by a vertex v_i. (Again, vertex v_{n+1} is a dummy vertex.) All switches in the clause gadgets representing the positive literal of the variable v_i are connected in series (the ordering of the switches on this string is not important): the vertex H in a switch is connected to vertex G of the next switch with the same label. Furthermore, there is an arc from v_i to vertex G in the first switch on its literal path, and an arc from vertex H in the last switch on the path to vertex v_{i+1}.

Likewise, all switches labelled with negated literals of this variable are connected. Thus there are two strings of switches leaving v_i: one contains all the positive literals, and one contains all the negated literals. Both end in v_{i+1}.

Also, *all* the switches are arranged on a path and connected by added arcs from vertex A in a switch to vertex C in the next one, and arcs back from vertex D in a switch to vertex B of the preceding switch. The ordering of the switches on this switch path is not important.

Finally, there is an arc from v_{n+1} to c_1 and an arc from vertex D in the first switch on the switch path to v_1.

To finish the construction of an instance of R2VDP it remains to identify the first four vertices. Vertex 1 is vertex B of the last switch on the switch path, vertex 2 is c_{m+1}, vertex 3 is vertex C of the first switch on the switch path, and vertex 4 is vertex A of the last switch on the switch path.

Lemma 7 *G_φ has two vertex disjoint paths from 1 to 2 and from 3 to 4 if and only if φ has a solution. Moreover, if G_φ contains such paths then it contains two such paths that together exhaust all its vertices.*

Proof. Assume φ can be satisfied so that exactly one variable in every clause is true. Walk through G_φ starting in vertex 1. This path is forced to traverse all switches until it reaches v_1. In general, assume that we reached v_i. To continue to v_{i+1} traverse the G–H paths of the string of negative literal switches if x_i is true; otherwise take the string of positive literal switches. Note that this forces us to avoid the E–F paths in these switches later.

Arriving at v_{n+1} continue to c_1. To travel from c_i to c_{i+1} we are forced to traverse the clause gadget of Fig. 3. Note that the truth assignment has set exactly one of the variables to true, blocking the E–F path in the two switches labelled by its negative literal. Likewise, two of the variables are false, blocking the (two) switches labelled by their positive literal. The remaining five switches are labelled by the positive literal of the true variable or negative literals of the falsified variables. The valid path ensured by Lem. 6 passes through exactly these five switches.

Finally, the path arrives at $v_{m+1} = 2$. The path travelling from 3 to 4 is now unique. Observe that the two paths exhaust all the vertices and thus form a Hamiltonian cycle if we add 23 and 41.

Conversely, assume there are two paths from 1 to 2 and from 3 to 4. The subpaths connecting v_i to v_{i+1} ensure that all literal switches are consistent in the sense that if the E–F path in a switch labelled x_i is blocked then it is blocked in *all* such switches, and not blocked in any switch labelled $\neg x_i$. This forces the subpaths from c_i to c_{i+1} to be valid. Lem. 6 ensures that the corresponding truth assignment is satisfying and sets exactly one variable in each clause. ∎

Acknowledgement

The third author would like to express his thanks to Chandra Chekuri for many useful discussions on this problem.

References

[1] N. Alon, R. Yuster, and U. Zwick. Color-coding. *Journal of the ACM*, 42(4):844–856, 1995.

[2] D. Angluin and L. G. Valiant. Fast probabilistic algorithms for Hamiltonian circuits and matchings. *Journal of Computer and Systems Sciences*, 18(2):155–193, 1979. Announced at STOC '77.

[3] A. Björklund and T. Husfeldt. Finding long paths in directed graphs is hard. Manuscript, 2001.

[4] A. Björklund and T. Husfeldt. Finding a path of superlogarithmic length. In *Proc. 29th ICALP*, volume 2380 of *LNCS*, pages 985–992. Springer Verlag, Berlin, 2002.

[5] A. Z. Broder, A. M. Frieze, and E. Shamir. Finding hidden Hamiltonian cycles. In *Proc. 23ndSTOC*, pages 182–189, 1991.

[6] T. Feder, R. Motwani, and C. S. Subi. Finding long paths and cycles in sparse Hamiltonian graphs. In *Proc. 32th STOC*, pages 524–529. ACM, 2000.

[7] S. Fortune, J. Hopcroft, and J. Wyllie. The directed subgraph homeomorphism problem. *Theoretical Computer Science*, 10:111–121, 1980.

[8] Y. Gurevich and S. Shelah. Expected computation time for Hamiltonian path problem. *SIAM Journal of Computing*, 16(3):486–502, 1987.

[9] R. Impagliazzo and R. Paturi. On the complexity of k-SAT. *Journal of Computer and Systems Sciences*, 62(2):367–375, 2001. Announced at CoCo '99.

[10] R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity? In *Proc. 39th FOCS*, pages 653–663, 1998.

[11] D. Karger, R. Motwani, and G.D.S. Ramkumar. On approximating the longest path in a graph. *Algorithmica*, 18(1):82–98, 1997.

[12] Sanjeev Khanna. Longest directed path is $n^{1-\epsilon}$-hard. Manuscript, 1999.

[13] C. H. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.

[14] N. Robertson and P. D. Seymour. Graph minors XIII: The disjoints paths problem. *J. Combinatorial Theory Ser. B*, 35, 1983.

[15] T. J. Schaefer. The complexity of satisfiability problems. In *Proc. 10th STOC*, pages 216–226, 1978.

[16] S. Vishwanathan. An approximation algorithm for finding a long path in Hamiltonian graphs. In *Proc. 11th SODA*, pages 680–685, 2000.

ECCC ISSN 1433-8092
http://www.eccc.uni-trier.de/eccc
ftp://ftp.eccc.uni-trier.de/pub/eccc
ftpmail@ftp.eccc.uni-trier.de, subject 'help eccc'