Journal of Computer Science and Cybernetics, V.35, N.1 (2019), 57–68
DOI 10.15625/1813-9663/35/1/12935

ON THE PERFORMANCE OF A SIMPLE APPROXIMATION ALGORITHM FOR THE LONGEST PATH PROBLEM

NGUYEN THI PHUONG[1,*], TRAN VINH DUC[1], LE CONG THANH[2]

[1] *Hanoi University of Science and Technology*
[2] *Institute of Mathematics, VAST*
[*] *phuongnt.math@gmail.com*

Abstract. The longest path problem is known to be NP-hard. Moreover, it cannot be approximated within a constant ratio, unless P = NP. The best known polynomial time approximation algorithms for this problem essentially find a path of length that is the logarithm of the optimum.

In this paper we investigate the performance of an approximation algorithm for this problem in almost every case. We show that a simple algorithm, based on depth-first search, finds on almost every undirected graph $G = (V, E)$ a path of length more than $|V| - 3\sqrt{|V| \log |V|}$ and so has performance ratio less than $1 + 4\sqrt{\log |V|/|V|}$.[1]

Mathematics Subject Classification (2010): 68Q17.

Keywords. Path; Hamiltonian path; Approximation algorithm and performance ratio.

1. INTRODUCTION

A well-known problem in graph theory is the longest path problem on graphs (finite, simple, loopless and undirected), write LPath for short, i.e., the problem of finding in a given graph $G = (V, E)$ a sequence $v_0v_1v_2 \dots v_k$ with the largest number of distinct vertices from V such that $v_{i-1}v_i \in E$ for $1 \le i \le k$. This problem is known to be NP-hard [11] and so cannot be solved in polynomial time unless P = NP. From practical requirements, the approximate approach to LPath is an effective solution. However, the problem LPath is also known to be NP-hard even in approximate solutions [16].

For convenience, we recall the concept of the performance of an approximation algorithm. The terminology and notation follow that in [11].

Performance ratios of an approximation algorithm

We are interested in the performance of approximation algorithms in the worst case and also in the almost every case. To formalize this approach, we settle on a general form for our guarantees, in terms of ratios, which was useful for comparison purpose and which seems to express nearness to optimality in a reasonable way. It is well-known that the performance guarantee for an approximation algorithm in the worst case is expressed by the *absolute performance ratio* (see [11]). The analogy to the performance guarantee in the almost every case is called *almost sure performance ratio*, which was defined by Thanh [20].

[1] Here and elsewhere we write the logarithm to the base 2 simply log.

Let Π be an optimization problem with instance set $\mathcal{I}_\Pi$. We use $OPT_\Pi(I)$ to denote the value of an optimal solution for an instance $I \in \mathcal{I}_\Pi$. And let A_Π be an approximation algorithm for Π. We use $A_\Pi(I)$ to denote the value of the feasible solution found by A_Π when applied to I.

If Π is a minimization (respectively, maximization) problem and I is any instance in $\mathcal{I}_\Pi$, then the *performance ratio* $R_{A_\Pi}(I)$ of an approximation algorithm A_Π on an instance I is defined by

$$R_{A_\Pi}(I) = \frac{A_\Pi(I)}{OPT_\Pi(I)} \left(\text{respectively, } \frac{OPT_\Pi(I)}{A_\Pi(I)} \right).$$

The *absolute performance ratio* R_{A_Π} of an approximation algorithm A_Π for a problem Π is given by

$$R_{A_\Pi} = \inf\{r \in \mathbb{R} : R_{A_\Pi}(I) \le r \text{ for any instance } I \in \mathcal{I}_\Pi\},$$

where $\mathbb{R}$ is the set of all real numbers.

Thus the absolute performance ratio is always a rational number greater than or equal 1 and is close to 1 when the feasible solution found by the approximation algorithm on any instance is close to optimal.

In order to compute an almost every case performance, one must first assume some probability distribution on the instances. Because our aim is to consider the almost every case performance of approximation algorithms for optimization problems in graph theory, now we simply define the "almost sure performance ratio" of an approximation algorithm for such an optimization problem Π that each instance has a discrete structure, and the set $\mathcal{I}_\Pi^{(n)}$ of all instances of "size" n is finite and $|\mathcal{I}_\Pi^{(n)}| \longrightarrow \infty$ as $n \longrightarrow \infty$. We turn $\mathcal{I}_\Pi^{(n)}$ into a probability space by taking its elements to be equiprobable. For example, if the instances of Π are labelled graphs, namely $\mathcal{I}_\Pi$ is the set $\mathcal{G}$ of all graphs with labelled vertices, then $\mathcal{I}_\Pi^{(n)}$ can be $\mathcal{G}_n$, the set of all graphs with n labelled vertices. In this case, the uniform probability distribution on the set $\mathcal{G}_n$ is given by probability $1/2^{\binom{n}{2}}$, the probability of a graph of $\mathcal{G}_n$, and each $G = (V, E) \in \mathcal{G}_n$ can be considered as a random graph, in which the edges for E are chosen independently with probability $1/2$.

First, given a property P, we say that the property P holds for *almost every instance* of the problem Π if

$$\Pr[I \in \mathcal{I}_\Pi^{(n)} : P \text{ holds for } I] \longrightarrow 1 \text{ as } n \longrightarrow \infty.$$

We now define the *almost sure performance ratio* $R_{A_\Pi}^{\text{a.s.}}$ of an approximation algorithm A_Π as follows

$$R_{A_\Pi}^{\text{a.s.}} = \inf\{r \in \mathbb{R} : R_{A_\Pi}(I) \le r \text{ for almost every instance } I \in \mathcal{I}_\Pi\}.$$

Thus the assertion "$R_{A_\Pi}^{\text{a.s.}} = r$" is equivalent to the usual statement that *the algorithm* A_Π *on an instance* I *of the problem* Π *has the performance ratio* $R_{A_\Pi}(I) \le r + o(1)$ *with high probability,* i.e.,

$$\Pr[I \in \mathcal{I}_\Pi^{(n)} : R_{A_\Pi}(I) \le r + o(1)] \longrightarrow 1 \text{ as } n \longrightarrow \infty.$$

In particular, the assertion "$R_{A_\Pi}^{\text{a.s.}} = 1$" means that the algorithm A_Π finds on almost every problem instance a feasible solution that is extremely close to optimal.

Essentially, the absolute performance ratio is the performance guarantee of an approximation algorithm in any case and so even in the worst case, while the almost sure performance ratio describes the performance behaviour of the algorithm on almost every problem instance. The performance analysis of approximation algorithms for some fundamental problems in graph theory has been discussed by Thanh [20, 21, 22] and shows that the performance of algorithms in almost every case is generally much better than their worst-case performance.

Related work

A classical problem in graph theory with numerous applications is the most well-known NP-complete Hamiltonian path problem, i.e., the problem of deciding whether a graph contains a Hamiltonian path, that is, a path in which every vertex of the graph appears exactly once. On the existence of a Hamiltonian path, several models of random graphs, in which almost every graph contains Hamiltonian path, have been studied (see [7]); in the general case, it has been shown that almost every graph (in the class $\mathcal{G}$) contains a Hamiltonian path.

The most natural optimization version of the Hamiltonian path problem is the longest path problem (LPath). Even when a graph does not contains a Hamiltonian path, it makes sense in several applications to search for a longest path. However, finding a longest path seems to be more difficult than deciding whether a graph contains a Hamiltonian path. Indeed, unless P = NP, the problem of finding a path of length $n - n^{\epsilon}$ for any $\epsilon < 1$ is NP-hard, where n is the number of vertices of the input graph [16]. Moreover, the problem LPath cannot even be approximated within a constant ratio [16].

Until recently, there are several known approaches to these problems. In individual cases, when the input is restricted to some *small classes of graphs*, there are only a few known polynomial algorithms for the longest path problem, and these were restricted to trees [9], block graphs [24], bipartite permutation graphs [25], ptolemaic graphs [19], and interval graphs [14]. The Hamiltonian path problem can be solved polynomially on proper interval graphs [4], interval graphs [2, 15], and cocomparability graphs [10]. In particular, for the Hamiltonian path problem on the class of graphs having many edges, namely the class of graphs G such that $|E(G)| \geq M_0(n)$ for some $M_0(n)$, where $n = |V(G)|$, Angluin and Valiant [1] gave a polynomial algorithm which finds on almost every graph in the class for $M_0(n) = c_n \log n$ a Hamiltonian path, and this result was improved by Shamir [18] whose algorithm worked for $M_0(n) = (n/2)\{\log n + (4+\varepsilon)\log\log n\}$, $\varepsilon > 0$. Finally Bollobás, Fenner and Frieze [8] constructed an algorithm for $M_0(n) = (n/2)(\log n + \log\log n + c_n)$, $c_n \to \infty$, which is about best possible. However, the algorithms described in [1] and [18] require the input graph to have its adjacency lists given in a random order. Thus these algorithms can be viewed as randomized algorithms that work well on random inputs while the algorithm in [8] is a deterministic algorithm that works well on random inputs.

In general case, when the input is the *class of all graphs*, there is an algorithm for solving the Hamiltonian path problem with polynomial expected running time [8], and even better with linear expected running time [23]. By the approximate approach to the longest path problem, the best known approximation algorithms for this problem essentially find paths of logarithmic length. Specifically, the first approximation algorithms are due to Monien [17] and Bodlaender [6], which find paths of length $\Omega(\log L/\log\log L)$, where L is the length of the longest path in the input graph. Next Alon, Yuster and Zwick [3] presented an algorithm that finds a path of length $\Omega(\log L)$. Later Björklund and Husfeldt [5] gave an algorithm

that finds a path of length $\Omega\big((\log L/\log\log L)^2\big)$. The best approximation algorithm known is of Gabow and Nie [12, 13]. Their algorithm finds a path of length $\exp\big(\Omega(\sqrt{\log L}\,)\big)$.

Our result

In this work, for the problem LPath on general graphs, we consider the performance behaviour of an approximation algorithm in almost every case. We show that a simple algorithm $ODFS_{\mathrm{LPath}}$, based on depth-first search, finds on almost every undirected graph $G=(V,E)$ a path of length more than $|V|-3\sqrt{|V|\log|V|}$ and so has performance ratio $R_{ODFS_{\mathrm{LPath}}}(G)<1+4\sqrt{\log|V|/|V|}$. This implies that the algorithm has almost sure performance ratio $R^{\mathrm{a.s.}}_{ODFS_{\mathrm{LPath}}}=1$. Thus the simple algorithm $ODFS_{\mathrm{LPath}}$ finds on almost every problem instance a feasible solution that is extremely close to optimal.

2. PRELIMINARIES

Let n be a natural number. We shall consider the set $\mathcal{G}_n$ of all graphs with vertex set $V=\{1,2,\ldots,n\}$. Clearly $\mathcal{G}_n$ has $2^{\binom{n}{2}}$ elements (i.e., graphs). For the sake of convenience, we write the set $\mathcal{G}_n$ as follows

$$\mathcal{G}_n=\{G_i\,|\,i=1,2,\ldots,N\},\quad \text{where } N\coloneqq 2^{\binom{n}{2}},$$

and view this set as a probability space in which all graphs have equal probability, namely $1/2^{\binom{n}{2}}$. Then all graph variants occur as random variables on $\mathcal{G}_n$, so we may talk about their expectation and variance.

In this section, in order to present a simple algorithm for the longest path problem and analyse its almost every case behaviour, we will discover features on the vertex degrees of almost every graph, namely low bounds for the minimum vertex degree of the subgraph induced by a subset of vertices of a graph.

Given a graph $G=(V,E)$ with a subset $V'\subseteq V$. For each n', where $n'\coloneqq|V'|$, we use the number $b=\lfloor n'/2-\sqrt{n'\log n'}\rfloor$ as a threshold for the degree of a vertex in the subgraph $G[V']$ of G induced by V'.

For the induced subgraph $G[V']$ of G, let $\eta_k(G[V'])$ denote the number of vertices of degree k for each k, $0\le k<n'$, and let $\eta_{\le b}(G[V'])$ denote the number of vertices of degree not greater than b. As we noted above, the values $\eta_k(G[V'])$ and $\eta_{\le b}(G[V'])$ occur as random variables on $\mathcal{G}_n$. We shall consider the following random variables

$$\eta_{n,n',k}\coloneqq\eta_k(G[V'])\quad\text{ and }\quad\eta_{n,n',\le b}\coloneqq\eta_{\le b}(G[V']).$$

Lemma 1. *For $0<n'\le n$, it holds that*

$$\mathrm{E}\big[\eta_{n,n',\le b}\big]=\frac{n'}{2^{n'-1}}\sum_{k=0}^{b}\binom{n'-1}{k}.$$

Proof. By the definition of the variables $\eta_{n,n',k}$ and $\eta_{n,n',\le b}$, we have

$$\eta_{n,n',\le b}=\sum_{k=0}^{b}\eta_{n,n',k}.$$

Therefore, by linearity of expectation, we obtain

$$\mathrm{E}\big[\eta_{n,n',\le b}\big] = \mathrm{E}\left[\sum_{k=0}^{b} \eta_{n,n',k}\right] = \sum_{k=0}^{b} \mathrm{E}\left[\eta_{n,n',k}\right],$$

where by the definition of expectation

$$\mathrm{E}\left[\eta_{n,n',k}\right] = \frac{1}{N}\sum_{i=1}^{N} \eta_k(G_i),$$

and so

$$\mathrm{E}\big[\eta_{n,n',\le b}\big] = \frac{1}{N}\sum_{k=0}^{b}\sum_{i=1}^{N} \eta_k(G_i). \tag{1}$$

In order to calculate the last sum, for each $0 \le k \le b$, we consider a bipartite graph $B_{N,n'}$ with vertex classes $\mathcal{G}_n$ and V' ($V' = \{v_1, v_2, \ldots, v_{n'}\}$) in which G_i and v_j are adjacent vertices if only if $\deg_{G_i[V']}(v_j) = k$. Now, for each vertex $v_j \in V'$, let $g_k(v_j)$ denote the number of graphs G_i in $\mathcal{G}_n$ such that $\deg_{G_i[V']}(v_j) = k$. Then, by the definition of the bipartite graph $B_{N,n'}$, essentially $\eta_k(G_i) = \deg_{B_{N,n'}}(G_i)$ and $g_k(v_j) = \deg_{B_{N,n'}}(v_j)$, and so

$$\sum_{i=1}^{N} \eta_k(G_i) = \sum_{j=1}^{n'} g_k(v_j).$$

Furthermore, note that the equality $\deg_{G_i[V']}(v_j) = k$ holds if only if in G_i the vertex v_j has exactly k neighbours from V' (the vertex v_j can have other neighbours from $V \setminus V'$). Hence for each $v_j, 1 \le j \le n'$, we find that

$$g_k(v_j) = 2^{\binom{n}{2}-(n'-1)} \cdot \binom{n'-1}{k}.$$

In fact, by the above note, there are $2^{\binom{n}{2}-(n'-1)}$ graphs G_i in $\mathcal{G}_n$ such that $\deg_{G_i[V']}(v_j) = k$ by some way of choosing k neighbours in V' for v_j; there are $\binom{n'-1}{k}$ ways of choosing k neighbours in V' for v_j; and different ways of choosing k neighbours for v_j give different graphs. Finally, for each $0 \le k \le b$, we may calculate that

$$\sum_{i=1}^{N} \eta_k(G_i) = n' 2^{\binom{n}{2}-(n'-1)} \cdot \binom{n'-1}{k} = \frac{n'N}{2^{n'-1}}\binom{n'-1}{k}.$$

This and (1) imply the required equality. ∎

We now determine an upper bound for the above expectation.

Lemma 2. *If $\log^6 n \le n' \le n$ and if n is sufficiently large, then*

$$\mathrm{E}\big[\eta_{n,n',\le b}\big] < \frac{2}{\log^3 n}.$$

Proof. Since $b+1 = \lfloor n'/2 - \sqrt{n'\log n'}\rfloor + 1 < n'/2$ and also $\binom{n'-1}{k} < \binom{n'}{b}$ for each k, $0 \le k \le b$, by Lemma 1 we have

$$\mathrm{E}\big[\eta_{n,n',\le b}\big] = \frac{n'}{2^{n'-1}}\sum_{k=0}^{b}\binom{n'-1}{k} < \frac{n'}{2^{n'-1}}(b+1)\binom{n'-1}{b} < \frac{n'^2}{2^{n'}}\binom{n'}{b}. \tag{2}$$

In order to estimate $\binom{n'}{b}$ we write the number b in the form $b = \frac{n'}{2} - \beta$, where β satisfies $\sqrt{n'\log n'} \le \beta < \sqrt{n'\log n'} + 1$. Thus

$$\binom{n'}{b} = \frac{n'!}{b!(n'-b)!} = \frac{n'!}{\big(\frac{n'}{2}-\beta\big)!\big(\frac{n'}{2}+\beta\big)!}\,.$$

Now by using the following inequalities

$$2\,x^{x+1/2}e^{-x} < x! < 2\sqrt{2}\,x^{x+1/2}e^{-x},$$

which are immediate from Stirling's formula for the factorial, we easily estimate $\binom{n'}{b}$ as follows

$$\begin{aligned}
\binom{n'}{b} &< \frac{{n'}^{n'+1/2}}{\sqrt{2}\big(\frac{n'}{2}-\beta\big)^{n'/2-\beta+1/2}\big(\frac{n'}{2}+\beta\big)^{n'/2+\beta+1/2}} \\
&= \frac{2^{n'+1/2}}{\sqrt{n'}\big(1-\frac{2\beta}{n'}\big)^{n'/2-\beta+1/2}\big(1+\frac{2\beta}{n'}\big)^{n'/2+\beta+1/2}} \\
&< \frac{2^{n'+1}}{\sqrt{n'}\big(1-\frac{2\beta}{n'}\big)^{n'/2-\beta}\big(1+\frac{2\beta}{n'}\big)^{n'/2+\beta}} \\
&= \frac{2^{n'+1}}{\sqrt{n'}\big(1-\frac{2\beta}{n'}\big)^{n'/2-\beta}\big(1+\frac{2\beta}{n'}\big)^{n'/2-\beta}\big(1+\frac{2\beta}{n'}\big)^{2\beta}} \\
&= \frac{2^{n'+1}}{\sqrt{n'}\big(1-\frac{4\beta^2}{n'^2}\big)^{n'/2-\beta}\big(1+\frac{2\beta}{n'}\big)^{2\beta}}.
\end{aligned}$$

Furthermore, applying the inequalities $\big(1-\frac{1}{y}\big)^{y-1} > \frac{1}{e}$ with $y = \frac{n'^2}{4\beta^2}$ and $\big(1+\frac{1}{z}\big)^{z+1} > e$ with $z = \frac{n'}{2\beta}$, respectively, to the terms $\big(1-\frac{4\beta^2}{n'^2}\big)^{n'/2-\beta}$ and $\big(1+\frac{2\beta}{n'}\big)^{2\beta}$, and also since $\sqrt{n'\log n'} \le \beta < \sqrt{n'\log n'}+1$, we have

$$\begin{aligned}
\binom{n'}{b} &< \frac{2^{n'+1}}{\sqrt{n'}e^{2\beta^2/(n'+2\beta)}} < \frac{2^{n'+1}}{\sqrt{n'}e^{2n'\log n'/(n'+2\sqrt{n'\log n'}+2)}} \\
&= \frac{2^{n'+1}}{\sqrt{n'}e^{2\log n'/(1+2\sqrt{(\log n')/n}+2/n')}} \\
&= \frac{2^{n'+1}}{\sqrt{n'}e^{2\ln n'\log e/(1+2\sqrt{(\log n')/n'}+2/n')}} \\
&\le \frac{2^{n'+1}}{\sqrt{n'}e^{2\ln n'}} = \frac{2}{\sqrt{n'}}\cdot\frac{2^{n'}}{n'^2}\,.
\end{aligned}$$

This and (2) imply the required inequality. ∎

Now, by Lemma 2, we may obtain the following essential result.

Theorem 3. *Given a graph $G=(V,E)$ with $V'\subseteq V$ such that $\log^6 n \le n' := |V'| \le n := |V|$. Then the minimum degree of the induced subgraph $G[V']$ of almost every graph G can be bounded from below as follows*

$$\Pr\Big[G\in\mathcal{G}_n\colon \delta(G[V'])> n'/2-\sqrt{n'\log n'}\Big] > 1-\frac{1}{\log^2 n},$$

when n is sufficiently large.

Proof. Applying Markov's inequality $\Pr\big[\xi < t\cdot \mathrm{E}\,[\xi]\big] > 1-1/t$ to the random variable $\eta_{n,n',\le b}$ and by choosing $t=\log^2 n$, we obtain

$$\Pr\Big[\eta_{n,n',\le b} < \log^2 n\cdot \mathrm{E}\big[\eta_{n,n',\le b}\big]\Big] > 1-\frac{1}{\log^2 n}.$$

Now by Lemma 2 we have

$$\Pr\left[\eta_{n,n',\le b} < \frac{2}{\log n}\right] > 1-\frac{1}{\log^2 n}.$$

Since $2/\log n < 1$ when n is sufficiently large, we find that

$$\Pr\big[\eta_{n,n',\le b}=0\big] > 1-\frac{1}{\log^2 n},$$

implying the required assertion of the theorem by the definitions of the variable $\eta_{n,n',\le b}$ and of the minimum degree of a graph. ∎

This result is the basis of the probabilistic analysis of our approximation algorithm for the longest path problem.

3. A SIMPLE ALGORITHM FOR THE LONGEST PATH

In this section we consider a simple approximation algorithm for the longest path problem (LPath). The algorithm is based on the well-known depth-first search (DFS) for traversing a given graph and performs the same search without backtracking. So we can say that this algorithm is an onward depth-first search; therefore, we denote it by $ODFS_{\mathrm{LPath}}$. Thus our algorithm $ODFS_{\mathrm{LPath}}$ starts at an arbitrary vertex and explores as far as possible along the first branch traversed by DFS, i.e., the branch from the starting vertex to the first backtracking one. The principle of the algorithm $ODFS_{\mathrm{LPath}}$ is quite simple: to go onward, from neighbour to neighbour, while this is possible.

This section is devoted to analyze the performance behaviour of the algorithm $ODFS_{\mathrm{LPath}}$. For convenience we describe $ODFS_{\mathrm{LPath}}$ as follows.

$ODFS_{\mathrm{LPath}}$ = "On input graph G:

1. Pick arbitrarily a vertex of G and mark it.

2. Repeat the following until no further marking is possible: pick arbitrarily a vertex among unmarked neighbours of the vertex marked just previously, and mark it.

3. Output the path formed by the sequence of marked vertices."

First note that the algorithm $ODFS_{\mathrm{LPath}}$ indeed runs in polynomial time. The main question is the following: *which rule should we use for picking a vertex at each step to be marked?* For $ODFS_{\mathrm{LPath}}$ to run correctly on every input graph $G = (V, E)$ with the above assumption that its vertices are labelled by natural numbers, $V = \{1, 2, \ldots, n\}$, we can simply stipulate that *at each step usual choice is to pick the vertex with the smallest label.*

Let's analyze the performance behaviour of the algorithm $ODFS_{\mathrm{LPath}}$ in almost every case. We estimate the length of the path found by Algorithm $ODFS_{\mathrm{LPath}}$ on almost every input graph G, namely the value $ODFS_{\mathrm{LPath}}(G)$.

Theorem 4. *The value $ODFS_{\mathrm{LPath}}(G)$ of the solution found by Algorithm $ODFS_{\mathrm{LPath}}$ on almost every input graph G can be bounded from below as follows*

$$\Pr\Big[G \in \mathcal{G}_n \colon ODFS_{\mathrm{LPath}}(G) > n - 3\sqrt{n \log n}\Big] > 1 - \frac{1}{\log n}\,,$$

when n is sufficiently large.

Proof. We consider the behaviour of Algorithm $ODFS_{\mathrm{LPath}}$ on every graph $G = (V, E) \in \mathcal{G}_n$ satisfying the following property concerning the minimum degree of the induced subgraph

$$\delta(G[V']) > n'/2 - \sqrt{n' \log n'}, \tag{D}$$

for each $V' \subseteq V$ such that $\log^6 n \le n' := |V'| \le n := |V|$.

In order to estimate the value $ODFS_{\mathrm{LPath}}(G)$, the computation process of $ODFS_{\mathrm{LPath}}$ on this input graph G is divided into successive stages. In the k^{th} stage, $k = 1, 2, \ldots$, the iteration step 2 is executed starting from the last marked vertex in the previous stage. Let ℓ_k be the number of vertices marked in the k^{th} stage. The number ℓ_k is determined by using property (D) with some subset $V' =: V_k$. Now, by induction on k, we show that

$$\ell_k = \left\lceil \frac{n - 2\sqrt{n \log n}}{2^k} \right\rceil. \tag{3}$$

In the first stage, since the graph G satisfies property (D) with $V' = V_1 = V$, then every vertex of G has degree more than $n/2 - \sqrt{n \log n}$. Therefore, after marking the vertex at step 1, denoted by v_0, the iteration step 2 is repeated at least $\lceil n/2 - \sqrt{n \log n} \rceil$ times and so we have determined the sequence of vertices marked here that $v_{11}, v_{12}, \ldots, v_{1\ell_1}$ with $\ell_1 = \lceil (n - 2\sqrt{n \log n})/2 \rceil$. In fact, property (D) with $V' = V$ ensures that, while the number of marked vertices is still no more than $\lceil n/2 - \sqrt{n \log n} \rceil$, the vertex marked just before that time has at least one unmarked neighbour (even in the worst situation that all marked vertices are its neighbours).

Next, assume formula (3) holds for t. Let $v_{k1}, v_{k2}, \ldots, v_{k\ell_k}$ (denoted by v_{k1}-$v_{k\ell_k}$) be the sequence of vertices marked in the k^{th} stage, $k = 1, 2, \ldots, t$. We will prove (3) for $t + 1$, by using property (D) with $V' = V_{t+1}$ such that

$$V_{t+1} = V \setminus \{v_0, v_{11}\text{-}v_{1\ell_1}, v_{21}\text{-}v_{2\ell_2}, \ldots, v_{t1}\text{-}v_{t\ell_t - 1}\}$$

(V_{t+1} contains $v_{t\ell_t}$, the last vertex marked in the t^{th} stage) and so

$$|V_{t+1}| = n - (1 + \ell_1 + \ell_2 + \cdots + \ell_t - 1) = n - \sum_{k=1}^{t} \ell_k \geq n - \Big(n - 2\sqrt{n \log n}\Big) \cdot \sum_{k=1}^{t} \frac{1}{2^k}$$
$$= n - \Big(n - 2\sqrt{n \log n}\Big)\left(1 - \frac{1}{2^t}\right) = \frac{n - 2\sqrt{n \log n}}{2^t} + 2\sqrt{n \log n}.$$

Note that, since $|V_{t+1}| \geq \log^6 n$ when n is sufficiently large, property (D) with $V' = V_{t+1}$ is satisfied. Therefore

$$\begin{aligned}\delta(G[V_{t+1}]) &> |V_{t+1}|/2 - \sqrt{|V_{t+1}| \log |V_{t+1}|} \\ &\geq \frac{n - 2\sqrt{n \log n}}{2^{t+1}} + \sqrt{n \log n} - \sqrt{|V_{t+1}| \log |V_{t+1}|} \\ &\geq \frac{n - 2\sqrt{n \log n}}{2^{t+1}} \quad (\text{since } |V_{t+1}| \leq n).\end{aligned}$$

Hence, by the same argument as above, the iteration step 2 is executed starting from the vertex $v_{t\ell_t}$ marked last in the t^{th} stage and is repeated at least $\lceil (n - 2\sqrt{n \log n})/2^{t+1} \rceil$ times. Thus we have determined the sequence of vertices marked in the $(t+1)^{\text{th}}$ stage that $v_{t+11}, v_{t+12}, \ldots, v_{t+1\ell_{t+1}}$ with $\ell_{t+1} = \lceil (n - 2\sqrt{n \log n})/2^{t+1} \rceil$. This is the claim.

Furthermore, for each $k = 1, 2, \ldots$, from Theorem 3 it follows that

$$\Pr\Big[G \in \mathcal{G}_n \colon ODFS_{\text{LPath}} \text{ on } G \text{ marks } \ell_k \text{ vertices in the } k^{\text{th}} \text{ stage}\Big] > 1 - \frac{1}{\log^2 n},$$

where $\ell_k = \lceil (n - 2\sqrt{n \log n})/2^k \rceil$. Now, by connecting initial $\lfloor \log n \rfloor$ stages, we have

$$\Pr\Big[G \in \mathcal{G}_n \colon ODFS_{\text{LPath}} \text{ on } G \text{ initially marks } \ell \text{ vertices}\Big] > 1 - \frac{1}{\log n},$$

where $\ell = 1 + \ell_1 + \ell_2 + \cdots + \ell_{\lfloor \log n \rfloor}$. Consequently, by definition $ODFS_{\text{LPath}}(G)$ namely is the length of the path found by Algorithm $ODFS_{\text{LPath}}$ on G, we obtain

$$\Pr\Big[G \in \mathcal{G}_n \colon ODFS_{\text{LPath}}(G) \geq \ell - 1\Big] > 1 - \frac{\lfloor \log n \rfloor}{\log^2 n} \geq 1 - \frac{1}{\log n}. \tag{4}$$

Finally, by formula (3) we have

$$\begin{aligned}\ell - 1 &\geq \Big(n - 2\sqrt{n \log n}\Big) \sum_{k=1}^{\lfloor \log n \rfloor} \frac{1}{2^k} = \Big(n - 2\sqrt{n \log n}\Big)\left(1 - \frac{1}{2^{\lfloor \log n \rfloor}}\right) \\ &> n - 2\sqrt{n \log n} - \Big(n - 2\sqrt{n \log n}\Big) \frac{1}{2^{\log n - 1}} = n - 2\sqrt{n \log n} - 2 + 4\sqrt{\frac{\log n}{n}} \\ &> n - 3\sqrt{n \log n}.\end{aligned}$$

This inequality and (4) imply the proof. ∎

Thus, by definition, Theorem 4 shows that the simple algorithm $ODFS_{\mathrm{LPath}}$ finds on almost every input graph $G = (V, E)$ a path of length $ODFS_{\mathrm{LPath}}(G)$ more than $|V| - 3\sqrt{|V| \log |V|}$.

Now, by this theorem we estimate the performance ratio of Algorithm $ODFS_{\mathrm{LPath}}$ on almost every case.

Theorem 5. *The performance ratio $R_{ODFS_{\mathrm{LPath}}}(G)$ of Algorithm $ODFS_{\mathrm{LPath}}$ on almost every graph instance G of the problem* LPath *can be bounded from above as follows*

$$\Pr\left[G \in \mathcal{G}_n \colon R_{ODFS_{\mathrm{LPath}}}(G) < 1 + 4\sqrt{\frac{\log n}{n}}\right] > 1 - \frac{1}{\log n}\,,$$

when n is sufficiently large.

Proof. It is clear that, for any graph $G \in \mathcal{G}_n$, a longest path has the length $OPT_{\mathrm{LPath}}(G)$ satisfying $OPT_{\mathrm{LP}}(G) \leq n - 1$. Therefore, on every graph G considered in Theorem 4 such that $ODFS_{\mathrm{LPath}}(G) > n - 3\sqrt{n \log n}$, the performance ratio $R_{ODFS_{\mathrm{LPath}}}(G)$ of Algorithm $ODFS_{\mathrm{LPath}}$ satisfies

$$R_{ODFS_{\mathrm{LPath}}}(G) = \frac{OPT_{\mathrm{LPath}}(G)}{ODFS_{\mathrm{LPath}}(G)} < \frac{n-1}{n - 3\sqrt{n \log n}} < 1 + 4\sqrt{\frac{\log n}{n}}\,,$$

when n is sufficiently large. Hence the proof is completed by Theorem 4. ∎

Obviously, by definition, on almost every input $G = (V, E)$ Algorithm $ODFS_{\mathrm{LPath}}$ has performance ratio $R_{ODFS_{\mathrm{LPath}}}(G)$ less than $1 + 4\sqrt{\log |V|/|V|}$ and thus almost sure performance ratio $R^{\mathrm{a.s.}}_{ODFS_{\mathrm{LPath}}} = 1$. However, as any approximation algorithm for the problem LPath, Algorithm $ODFS_{\mathrm{LPath}}$ has absolute performance ratio $R_{ODFS_{\mathrm{LPath}}} = \infty$. Such assertion can be obtained by showing some graph $G \in \mathcal{G}_n$ on which $R_{ODFS_{\mathrm{LPath}}}(G) = r(n)$ (even on which $R_{ODFS_{\mathrm{LPath}}}(G) = n-1$) such that $r(n) \longrightarrow \infty$ as $n \longrightarrow \infty$, and so $R_{ODFS_{\mathrm{LPath}}} = \infty$ by definition. This is easy shown and is left to the reader.

Thus we have the following main consequence.

Corollary 6. *Algorithm $ODFS_{\mathrm{LPath}}$ has the performance ratios satisfying:*

(i) $R_{ODFS_{\mathrm{LPath}}} = \infty$.

(ii) $R_{ODFS_{\mathrm{LPath}}}(G) < 1 + 4\sqrt{\frac{\log |V|}{|V|}}$ *on almost every input graph* $G = (V, E)$.

(iii) $R^{\mathrm{a.s.}}_{ODFS_{\mathrm{LPath}}} = 1$.

Finally we may conclude that the longest path problem LPath is known to be NP-hard, even in approximate solutions. More precisely, from the worst case point of view, the problem LPath cannot be solved by a polynomial approximation algorithm with absolute performance ratio less than ∞, unless P = NP. Moreover, the best known polynomial time algorithms for the problem LPath only find solutions with logarithmic optimal value. However, although this problem is hard, it is only in rare worst cases; from the almost every case point of view, the simple algorithm $ODFS_{\mathrm{LPath}}$ finds on almost every problem instance a solution that is extremely close to optimal.

ACKNOWLEDGMENT

This research is funded by the Hanoi University of Science and Technology (HUST) under the project number T2017-PC-075.

REFERENCES

[1] D. Angluin and L.G. Valiant, "Fast probabilistic algorithm for Hamiltonian circuits and matchings", *Journal of Computer and System Sciences*, vol. 18, no. 2, pp. 155–193, 1979.

[2] S.R. Arikati and C.P. Rangan, "Linnear algorithm for optimal path cover problem on interval graphs", *Inform. Process. Lett.*, vol. 35, no. 3, pp. 149–153, 1990.

[3] N. Alon, R. Yuster, and U. Zwick, "Color-coding", *J. ACM,* vol. 42, pp. 844–856, 1995.

[4] A.A. Bertossi, "Finding Hamiltolian circuits in proper interval graphs", *Inform. Process. Lett.*, vol. 17, pp. 97–101, 1983.

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

[6] H.L. Bodlaender, "On linear time minor tests with depth-first search", *J. Algorithms*, vol. 14, pp. 1–23, 1993.

[7] B. Bollobás, *Random Graphs,* 2nd Edition. Cambridge University Press, 2001.

[8] B. Bollobás, T.I. Fenner and A.M. Frieze, "An algorithm for finding Hamilton paths and cycles in random graphs", *Combinatorica*, vol. 7, pp. 327–341, 1987.

[9] R.W. Bulterman, F.W. van der Sommen, G. Zwaan, T. Verhoeff, A.J.M. van Gasteren, and W.H.J. Feijen, "On computing a longest path in a trees", *Inform. Process. Lett.*, vol. 81, pp. 93–96, 2002.

[10] P. Damaschke, J.S. Deogun, D. Kratsch, and G. Stener, "Finding Hamiltolian paths in cocomparability graphs using the bump number algorithm", *Order*, vol. 8, no. 4, pp. 383–391, 1991.

[11] M.R. Garey and D.S. Johnson, *Computers and Intractability - A Guide to the NP-completeness.* W.H. Freeman, 1979.

[12] H.N. Gabow, "Finding paths and cycles of superpolylogarithmic length", *SIAM Journal on Computing*, vol.36, pp. 1649–1671, 2007.

[13] H.N. Gabow and S. Nie, "Finding long paths, cycles and circuits", *19th Annual International Symp. on Algorithms and Computation, LNCS 5369,* Gold Coast, Australia, December 15-17, 2008. pp. 752–763.

[14] K. Ioannidou, G.B. Mertzios, and S.D. Nikolopoulos, "The longest path problem has a polynomial solution on interval graphs", *Algorithmica,* vol. 61, pp. 320–341, 2011.

[15] J.M. Keil, "Finding Hamiltonian circuits in interval graphs", *Inform. Process. Lett.*, vol. 20, pp. 201–206, 1983.

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

[17] B. Monien, "How to find long paths efficiently", *Annals of Discrete Mathematics*, vol. 25, pp. 239–254, 1985.

[18] E. Shamir, "How many random edges make a graph Hamiltonian?", *Combinatorica*, vol. 3, pp. 123–132, 1983.

[19] Y. Takahara, S. Teramoto, and R. Uehara, "Longest path problems on ptolemaic graphs", *IEICE Trans. Inform. System.*, vol. E91-D, pp. 170–177, 2008.

[20] L.C. Thanh, "On the approximability of Max-Cut", *Vietnam J. Math.*, vol. 34, no. 4, pp. 389–395, 2006.

[21] L.C. Thanh, "Performance analysis of greedy algorithms for Max-IS and Min-Maxl-Match", *Vietnam J. Math.*, vol. 36, no. 3, pp. 327–336, 2008.

[22] L.C. Thanh, "Minimum connected dominating sets in finite graphs", *Vietnam J. Math.*, vol. 38, no. 2, pp. 157–168, 2010.

[23] A. Thomason, "A simple linear expected time algorithm for finding a Hamilton path", *Discrete Math.*, vol. 75, pp. 373–379, 1989.

[24] R. Uehara and Y. Uno, "Efficient algorithms for the longest path problem", *15th Annual International Symp. on Algorithms and Computation, LNCS 3314,* Hong Kong, China, December 20-22, 2004. pp. 871–883.

[25] R. Uehara and G. Valiente, "Linear structure of bipartite permutation graphs and the longest path problem", *Inform. Process. Lett.*, vol. 103, pp. 71–77, 2007.

Received on August 06, 2018
Revised on December 18, 2018