Detecting whether a graph has a fixed-point-free automorphisms is in polynomial Time

Yasunori Ohto

Abstract

The problem of determining whether a graph has a fixed-point-free automorphism is NP-complete. We demonstrate that the problem can be solved efficiently within polynomial time. First, we obtain the automorphisms of an input graph G by using a spectral method. Next, we prove the Theorem used to detect whether there is a fixed-point free automorphism in G. Next, we construct an algorithm to detect whether G has a fixed-point-free automorphism using this result. The computational complexity of detecting whether a graph has a fixed-point-free automorphism is $\mathcal{O}(n^5)$. If fixed-point-free automorphism exist, the computational complexity of obtaining a fixed-point-free automorphism is $\mathcal{O}(n^6)$. Then, the complexity classes P and NP are the same.

Index Terms

fixed-point-free automorphism, graph spectrum, polynomial time computation, NP-complete problem.

I. Introduction

The P versus NP problem [1], [2] is one of the major problems in theoretical computer science. An answer to this problem would determine whether problems that can be verified in polynomial time can also be solved in polynomial time. Attempts have been made to prove that P is not equal to NP. However, it has been shown that this cannot be proven or is difficult to prove using the methods of relativizing proofs [3], natural proofs [4], and algebrizing proofs [5]. On the other hand, many attempts have been made to show the lower bound of the computational complexity of NP problems [6], [7], mainly by pruning unnecessary branches [8]. Some attempts have been made to obtain the average computational complexity as polynomial time [9]. It has also been shown that by restricting the inputs, the computational complexity can be achieved polynomial time [10]. However, it is still unclear whether P and NP are equal. In contrast, we will show a lower bound on the computational complexity of an NP-complete problem without limitation by introducing a spectral method to handle multiple states at once.

If a problem is NP and all other NP problems are polynomial-time reducible to it, the problem is NP-complete [11]. If one of the NP-complete problems can be solved in polynomial time, the complexity classes P and NP are the same. The problem of determining whether a given graph has a fixed-point-free automorphism is NP-complete [12]. In this paper, we show that it is solvable in polynomial time. After obtaining all automorphisms containing a transposition of two vertices, G has a fixed-free-point automorphism if, and only if, these automorphisms transpose all vertices. Then, we show that this algorithm can be solved in polynomial time. Note that we compare the two eigenvalue sets without real number calculations by using Frobenius normal form [13], [14]. The computational complexity of detecting whether a graph has a fixed-point-free automorphism is $\mathcal{O}(n^5)$. If fixed-point-free automorphism exists, the computational complexity of obtaining a fixed-point-free automorphism is $\mathcal{O}(n^6)$. Since one of the NP-complete problems is solvable in polynomial time, the complexity classes P and NP are the same.

This paper is organized as follows. Section II provides the proofs used to determine whether a given graph has a fixed-point-free automorphism. Section III presents an algorithm to solve this problem. Section IV presents a discussion of this result. Finally, Section V presents a conclusion regarding the result of this paper.

II. Proof

In this section, we provide a proof for detecting whether a given graph has a fixed-point-free automorphism in polynomial time.

First, we show that we can obtain all automorphisms by composition using automorphisms of order two. Next, we show that we can obtain the automorphisms containing transposing two vertices by comparing the eigenvalue set of the adjacency matrix of a vertex-weighted graph by weighting it to a vertex. Next, we show that there is a fixed-point-free automorphism exists if and only if, for any vertex v_a there is a vertex $v_b \neq v_a$ such that automorphisms that contain transposing two vertices v_a and v_b.

A. Fixed-point-free automorphism

We define the fixed-point-free automorphism as follow.

Definition II.1. Suppose that a graph $G = (E, V)$ has an automorphism. Let ψ be the automorphic transformation. A fixed-point-free automorphism is an automorphism such that $\psi(v) \neq v$ at all vertices $v \in V$.

B. Preparation

We define the following functions, which will be used in the proofs and the methods. Suppose S is a vertex-weighted graph. Let $V_{w0}(S)$ be the set of vertices of S with weight 0. Let $Sg(S, v, w)$ be the vertex-weighted graph in which the weight $w \in \mathbb{N}$ is given to vertex v of S. Denote the adjacency matrix of S by $A(S)$. Let $Ev(S)$ be the set (with multiplicities) of eigenvalues of $A(S)$.

C. Composition using automorphisms of order two

Let $Aut(G)$ be the automorphism group of G. This subsection shows that we can obtain all automorphisms $\psi \in Aut(G)$ by composition of automorphisms of order two.

Corollary II.1. *There are certain automorphisms $\psi_1, \psi_2, \ldots, \psi_m \in Aut(G = (V, E))$ of order two, and we can explain $\psi = \psi_m \psi_{m-1} \cdots \psi_1$.*

Proof. Permuting vertices of ψ which consists of transposition (automorphism of order two) and cycling (automorphism of order above two). Let $\sigma_1 \cdots \sigma_r \in V$. Suppose that a composition of automorphisms has a cycle $\psi_c = (\sigma_1 \cdots \sigma_r)$ of order r. There is a bijection $\sigma_i \to \sigma_{i+1}$ with $0 < i < r$ and a bijection $\sigma_r \to \sigma_1$ exists. Thus, there exist two transpositions. One is the automorphism $\psi_{1,2}$ containing the transposition of σ_1 and σ_2. And the other is the automorphism $\psi_{2,r}$ containing the transposition of σ_2 and σ_r. So, we obtain ψ_c by applying $\psi_{2,r}$ following $\psi_{1,2}$. Thus, we can reduce all cycling to the composition of transpositions. $\square$

Example II.2. *Suppose a cycle $\psi_{c5} = (\sigma_1, \cdots, \sigma_5)$ of order 5. The automorphism contains the transposition $\psi_{1,2} = ((\sigma_1, \sigma_2), (\sigma_3, \sigma_5))$ and $\psi_{2,5} = ((\sigma_2, \sigma_5), (\sigma_3, \sigma_4))$. Thus, we obtain $\psi_{c5} = (((\sigma_1, \sigma_2), (\sigma_3, \sigma_5)), ((\sigma_2, \sigma_5), (\sigma_3, \sigma_4)))$.*

Example II.3. *Suppose a cycle $\psi_{c6}(\sigma_1, \cdots, \sigma_6)$ of order 6. The automorphism contains the transposition $\psi_{1,2} = ((\sigma_1, \sigma_2), (\sigma_3, \sigma_6), (\sigma_4, \sigma_5))$ and $\psi_{2,6} = ((\sigma_2, \sigma_6), (\sigma_3, \sigma_5))$. Thus, we obtain $\psi_{c6} = (((\sigma_1, \sigma_2), (\sigma_3, \sigma_6), (\sigma_4, \sigma_5)), ((\sigma_2, \sigma_6), (\sigma_3, \sigma_5)))$.*

Figure 2 shows an example of obtaining a cyclic automorphism by compositions of automorphisms of order two.

D. Obtaining the automorphisms that contain transposing two vertices

This subsection shows that we can obtain the automorphisms containing transposing two vertices by comparing the eigenvalue set of the adjacency matrix of a vertex-weighted graph by weighting it to a vertex.

An automorphism of order two contains transposing two vertices. So, we remove fixed points by compositions of automorphisms that contain transposing two vertices. Thus, we use Theorem II.4 and Corollary II.5 [15] to obtain the automorphisms that contain transposing two vertices of an input graph G using eigenvalue sets.

Theorem II.4. *Let $S_{v_i} = Sg(S, v_i, w)$ and $S_{v_j} = Sg(S, v_j, w)$ with $v_i, v_j \in V_{w0}(S)$, $v_i \neq v_j$ and $w > 0$. When $Ev(S_{v_i}) = Ev(S_{v_j})$, S_{v_i} and S_{v_j} are isomorphic.*

The following proof is reproduced from the reference [15].

Proof. We show that if $Ev(S_{v_i}) = Ev(S_{v_j})$, then S_{v_i} and S_{v_j} are not cospectral but isomorphic.

Let $A(S_{v_i})$ and $A(S_{v_j})$ be A_{v_i} and A_{v_j}, respectively. When there exists a permutation matrix P such that $A_{v_i} = P^t A_{v_j} P$, S_{v_i} and S_{v_j} are isomorphic. Denote the eigenfunctions of A_{v_i} and A_{v_j} by f_{v_i} and f_{v_j}, respectively. When f_{v_i} and f_{v_j} are the same, the eigenvalue sets of A_{v_i} and A_{v_j} are the same. Therefore, we will prove that such a nontrivial permutation matrix exists when $f_{v_i} - f_{v_j} = 0$.

Without loss of generality, we may assume $i = 1$ and $j = 2$. We show the characteristic polynomials f_{v_1} and f_{v_2} as below.

$$\begin{aligned} f_{v_1} &= |A_{v_1} - \lambda I| \\ &= \begin{vmatrix} w-\lambda & a_{1,2} & a_{1,3} & a_{1,4} & \cdots & a_{1,n} \\ a_{2,1} & -\lambda & a_{2,3} & a_{2,4} & \cdots & a_{2,n} \\ a_{3,1} & a_{3,2} & w_3-\lambda & a_{3,4} & \cdots & a_{3,n} \\ a_{4,1} & a_{4,2} & a_{4,3} & \ddots & & \vdots \\ \vdots & \vdots & \vdots & & \ddots & \vdots \\ a_{n,1} & a_{n,2} & a_{n,3} & \cdots & \cdots & w_n-\lambda \end{vmatrix}, \\ f_{v_2} &= |A_{v_2} - \lambda I| \\ &= \begin{vmatrix} -\lambda & a_{1,2} & a_{1,3} & a_{1,4} & \cdots & a_{1,n} \\ a_{2,1} & w-\lambda & a_{2,3} & a_{2,4} & \cdots & a_{2,n} \\ a_{3,1} & a_{3,2} & w_3-\lambda & a_{3,4} & \cdots & a_{3,n} \\ a_{4,1} & a_{4,2} & a_{4,3} & \ddots & & \vdots \\ \vdots & \vdots & \vdots & & \ddots & \vdots \\ a_{n,1} & a_{n,2} & a_{n,3} & \cdots & \cdots & w_n-\lambda \end{vmatrix}. \end{aligned}$$

The weights of the vertices are w, $w_3, \ldots, w_n$, all of which are integers. Then,

$$\begin{aligned} f_{v_1} - f_{v_2} &= w \begin{vmatrix} 0 & a_{2,3} & a_{2,4} & \cdots & a_{2,n} \\ a_{3,2} & w_3-\lambda & a_{3,4} & \cdots & a_{3,n} \\ a_{4,2} & a_{4,3} & \ddots & & a_{3,n} \\ \vdots & \vdots & & \ddots & \vdots \\ a_{n,2} & a_{n,3} & \cdots & \cdots & w_n-\lambda \end{vmatrix} - w \begin{vmatrix} 0 & a_{1,3} & a_{1,4} & \cdots & a_{1,n} \\ a_{3,1} & w_3-\lambda & a_{3,4} & \cdots & a_{3,n} \\ a_{4,1} & a_{4,3} & \ddots & & a_{3,n} \\ \vdots & \vdots & & \ddots & \vdots \\ a_{n,1} & a_{n,3} & \cdots & \cdots & w_n-\lambda \end{vmatrix} \\ &= 0. \end{aligned} \tag{1}$$

If $n = 2$, f_{v_1} and f_{v_2} are the same. Hence, in this case, S_{v_1} and S_{v_2} are isomorphic.

We treat the case of $n = 3$ as follows. Equation 1 becomes

$$\begin{aligned} f_{v_1} - f_{v_2} &= w \begin{vmatrix} 0 & a_{2,3} \\ a_{3,2} & w_3 - \lambda \end{vmatrix} - w \begin{vmatrix} 0 & a_{1,3} \\ a_{3,1} & w_3 - \lambda \end{vmatrix} \\ &= w(a_{2,3}a_{3,2} - a_{1,3}a_{3,1}) \\ &= 0. \end{aligned}$$

So, when $a_{2,3} = a_{1,3}$, f_{v_1} and f_{v_2} are the same. For this case, then, S_{v_1} and S_{v_2} are isomorphic.

Let $n > 3$. Suppose the matrix A' is as follows.

$$A' = \begin{pmatrix} w_3 & a_{3,4} & \cdots & a_{3,n} \\ a_{4,3} & \ddots & & a_{3,n} \\ \vdots & & \ddots & \vdots \\ a_{n,3} & \cdots & \cdots & w_n \end{pmatrix}.$$

Let vertex $u_1 = (a_{1,3}, a_{1,4}, \ldots, a_{1,n})^t$ and $u_2 = (a_{2,3}, a_{2,4}, \ldots, a_{2,n})^t$. Then, Equation 1 becomes as follows.

$$\begin{aligned} f_{v_1} - f_{v_2} &= w \begin{vmatrix} 0 & u_2^t \\ u_2 & A' - \lambda I \end{vmatrix} - w \begin{vmatrix} 0 & u_1^t \\ u_1 & A' - \lambda I \end{vmatrix} \\ &= 0. \end{aligned}$$

In order for f_{v_1} and f_{v_2} to be the same, it is necessary that $f_{v_1} - f_{v_2} = 0$ for all λ. So, we assume $|A' - \lambda I| \neq 0$. Then,

$$\begin{aligned} f_{v_1} - f_{v_2} &= & w|A' - \lambda I||0 - u_2^t(A' - \lambda I)^{-1}u_2| \\ &- & w|A' - \lambda I||0 - u_1^t(A' - \lambda I)^{-1}u_1| \\ &= & w|A' - \lambda I|(u_2 - u_1)^t(A' - \lambda I)^{-1}(u_2 - u1) \\ &= & 0. \end{aligned}$$

When $u_1 = u_2$, f_{v_1} and f_{v_2} are the same. In this case, then, S_{v_1} and S_{v_2} are isomorphic.

Let $u_2 \neq u_1$. When $(u_2 - u_1)^t(A' - \lambda I)^{-1}(u_2 - u_1) = 0$, $u_2 - u_1$ and $(A' - \lambda I)^{-1}(u_2 - u_1)$ are orthogonal. So,

$$\begin{aligned} (u_2 - u_1)^t(A' - \lambda I)(u_2 - u_1) &= u_2^t A' u_2 - u_1^t A' u_1 - u_2^t \lambda I u_2 + u_1^t \lambda I u_1 \\ &= 0. \end{aligned}$$

In order for f_{v_1} and f_{v_2} to be the same, it is necessary that $f_{v_1} - f_{v_2} = 0$ for all λ. So, the number of elements with value 1 in u_2 and u_1 is the same.

Since $u_2 - u_1$ and $(A' - \lambda I)(u_2 - u_1)$ are orthogonal,

$$\begin{aligned} (u_2 - u_1)^t A'(u_2 - u_1) &= (u_2 - u_1)^t P'^t A' P'(u_2 - u_1) \\ &= (u_1 - u_2)^t P'^t A' P'(u_1 - u_2) \\ &= 0 \end{aligned}$$

with P' a liner operator. When A_1 and A_2 have the same eigenvalue set, there exists a set of nontrivial permutation matrices $\{P' | P'^t A' P' = A' \wedge (u_2 - u_1) = P'(u_1 - u_2))\}$. So, S_{v_1} and S_{v_2} are isomorphic. □

Corollary II.5. *Let $S_{v_i} = Sg(S, v_i, w)$ and $S_{v_j} = Sg(S, v_j, w)$ with $v_i, v_j \in V_{w0}(S)$, $v_i \neq v_j$ and $w > 0$. If $Ev(S_{v_i}) \neq Ev(Sv_j)$, then S_{v_i} and S_{v_j} are not isomorphic.*

The following proof is reproduced from the reference [15].

Proof. By applying a permutation matrix P, $P^t A(S_{v_j})P$ is not equal to $A(S_{v_i})$. So, there is no bijection between S_{v_i} and S_{v_j}. Therefore, S_{v_i} and S_{v_j} are not isomorphic. □

Thus, since Theorem II.4 and Corollary II.5, when $Ev(S_{v_i}) = Ev(S_{v_j})$ if, and only if, there is an automorphism in G that contains the transposition of v_i and v_j. Function 2 obtains all automorphisms in G. And figure 1 shows an example of obtaining all automorphisms in G that contains the transposition of two vertices.

E. Detecting whether there is a fixed-point-free automorphism

This subsection shows that there is a fixed-point-free automorphism exists if and only if, for any vertex v_a there is a vertex $v_b \neq v_a$ such that automorphisms that contain transposing two vertices v_a and v_b.

Lemma II.6 and Theorem II.7 prove that it is possible to detect whether there is a fixed-point-free automorphism in G.

1) Composition of automorphisms does not increase the fixed points: We prove that the composition of automorphisms does not increase the fixed points.

Lemma II.6. *Suppose that a graph $G = (E, V)$ has nontrivial automorphisms $\psi_a, \psi_b \in Aut(G)$, where $\psi_a \neq \psi_b$. Let ψ_a have fixed points $V_{fixed,\psi_a} = \{v|\psi_a(v) = v, v \in V\}$. Suppose, ψ_b has the vertex transposition $\psi_b(v_a) = v_b$ and $\psi_b(v_b) = v_a$, $v_a \in V_{fixed,\psi_a}$. When we apply ψ_b following ψ_a, the set of fixed points becomes $V_{fixed,\psi_a} \cap V_{fixed,\psi_b}$.*

Proof. Suppose $\psi_a : V_{a,s} \mapsto V_{a,d}$ with $(V_{a,s} \cup V_{a,d}) \oplus V_{fixed,\psi_a} = V$. When we apply ψ_b following ψ_a, we obtain $\psi_b \circ \psi_a$ such that $V_{a,s} \cup V_{fixed,\psi_a} \mapsto V_{a,s} \cup V_{fixed,\psi_a}$ and $V_{a,d} \cup V_{fixed,\psi_a} \mapsto V_{a,d} \cup V_{fixed,\psi_a}$, so the vertices belonging to $V_{a,s}$ and $V_{a,d}$ are not returned to the original points by ψ_b. The automorphic transformation ψ_b maps at least one vertex v to another. Then, v becomes not a fixed point. □

Figure 2 shows an example of fixed points being reduced by using composition of automorphisms.

2) Detecting whether there is a fixed-point-free automorphism: We prove how to detect whether there is a fixed-point-free automorphism.

Theorem II.7. *We obtain the vertex sets $V_\lambda \subset V$ with the same $\lambda_v = Ev(Sg(S, v, w))$, $v \in V$, $w > 0$. $V_{\lambda_{v_i}} > 1$ for all vertex sets of λ_{v_i} if, and only if, G has a fixed-point-free automorphism.*

Proof. From Lemma II.6, applying the composition of automorphic transformations to G does not increase the size of the set of fixed points. Suppose that the set of fixed points V_{fixed} exists after applying the composition of the automorphism $\psi_1 \cdots \psi_i$ to G. We can reduce the size of V_{fixed} by applying an automorphism ψ_{i+1} that contains the transposition of $v \in V_{fixed}$ and another vertex.

When $V_{\lambda_{v_i}} > 1$ for all vertex sets, there exists ψ such that $\psi(v) \neq v$ at every vertex v. On the other hand, suppose there is a set of vertices such that $|V_{\lambda_{v_j}}| = 1$. There is no ψ such that $\psi(v) \neq v$ at $v \in V_{\lambda_{v_j}}$. Then, v becomes a fixed point. □

Function 3 detects whether a graph G has a fixed-point-free automorphism in h. And figure 1 shows an example of detecting a fixed point for the graph $G = (V, E)$.

III. Algorithm

This section presents a polynomial-time algorithm to determine whether a graph has a fixed-point-free automorphism and shows to be able to obtain a fixed-point-free automorphism in polynomial time if it exists. We assume that the number of vertices of the graph is n.

First, we show that how to compare the sets of eigenvalues without real number calculations. Next, we show how to obtain a fixed-point-free automorphism.

Algorithm 1 Obtaining a fixed-point-free automorphism in G if it exists.

function OBTAIN_A_FIXED-POINT-FREE_AUTOMORPHISM_IF_IT_IS_EXISTS($G = (V, E)$)
    $h \leftarrow$ OBTAIN_AUTOMORPHISMS(G)
    if IS_FIXED-POINT-FREE_AUTOMORPHISM_EXISTS(h) **then**
        return OBTAIN_A_FIXED_POINT_FREE_AUTOMORPHISM(G,h)
    else
        return null
    end if
end function

Algorithm 2 Obtaining all automorphisms in $G = (V, E)$.

function OBTAIN_AUTOMORPHISMS($G = (V, E)$)
    $S \leftarrow G$ with all vertex weights are 0
    $w \leftarrow 2|V|$
    clear hash h
    for each $v \in V$ **do**
        $\lambda \leftarrow Ev(Sg(S, v, w))$
        if $h(\lambda) = \emptyset$ **then**
            $h(\lambda) \leftarrow \{v\}$
        else
            $h(\lambda) \leftarrow h(\lambda) \cup \{v\}$
        end if
    end for
    return h
end function

A. Comparing the sets of eigenvalues

This subsection shows that how to compare the sets of eigenvalues without real number calculations.

Since the elements of an adjacency matrix of a vertex-weighted graph are all integers, the coefficients of the eigenequation of this matrix are all integers. We use the set of coefficients of the eigenequation of the adjacency matrix of a vertex-weighted graph instead of its set of eigenvalues. We calculate the Frobenius normal form to obtain the set of coefficients without real number calculations. Then, we compare the coefficients to determine whether the sets of eigenvalues are the same. The amount of computation required to convert an adjacency matrix into the Frobenius normal form is $\mathcal{O}(n^4)$.

B. Obtaining a fixed-point-free automorphism

This subsection shows how to obtain a fixed-point-free automorphism.

1) Flow of the algorithm: Function 1 obtains all fixed-point-free automorphism in G. First, we obtain the automorphisms in G by using Function 2. Next, we check whether a fixed-point-free automorphism exists by using Function 3. If it exists, we obtain a fixed point free automorphism by using Function 4.

2) Obtaining all automorphisms: Function 2 obtains all automorphisms in G. By adding a weight $w > 0$ to a vertex, we obtain a set of vertices V_λ with the same eigenvalue set. Thus, we obtain automorphisms of G from Theorem II.4 and Corollary II.5. The computational complexity of this function is $\mathcal{O}(n^5)$.

3) Detecting whether there is a fixed-point-free automorphism: Function 3 detects whether a graph G has a fixed-point-free automorphism in h. We check if the size of the vertex set V_λ is 1 or above to determine whether there is a fixed-point-free automorphism based on Theorem II.7. The computational complexity of this function is $\mathcal{O}(n)$.

Example III.1. *Figure 1 shows an example of detecting a fixed point for the graph $G = (V, E)$. Let the vertex weighted graph $S = G$. We identify the sets of vertices $V_\lambda \subset V$ that share the same eigenvalue $\lambda_v = Ev(Sg(S, v, w))$, where v is a vertex in the graph and w is a positive weight. Then, we obtain*

Algorithm 3 Detecting whether a graph G has a fixed-point-free automorphism in h.

function IS_FIXED-POINT-FREE_AUTOMORPHISM_EXISTS(h)
    for each $T \in h$ **do**
        if $|T| = 1$ **then**
            return FALSE
        end if
    end for
    return TRUE
end function

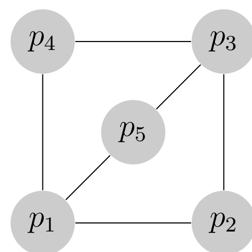

Fig. 1. An example of detecting a fixed point.

$V_{\lambda_1} = \{p_1, p_3\}$, $V_{\lambda_2} = \{p_2, p_4\}$ and $V_{\lambda_3} = \{p_5\}$. Thus, since $|V_{\lambda_3}| = 1$, G has no fixed-point-free automorphism.

Since $V_{\lambda_1} = \{p_1, p_3\}$, there exists an automorphic transformation ψ_1 that contains the transposition of vertices p_1 and p_3. When we apply ψ_1, vertices p_2, p_4 and p_5 become fixed points. Now, since $V_{\lambda_2} = \{p_2, p_4\}$, there exists an automorphic transformation ψ_2 that contains the transposition of vertices p_2 and p_4. Thus, applying ψ_2 after ψ_1 leaves p_5 as a fixed point. Since $V_{\lambda_3} = \{p_5\}$, there is no automorphic transformation that involves the transposition of vertex p_5 and another vertex. Therefore, vertex p_5 remains as a fixed point.

Function 4 obtains a fixed-point-free automorphism. First, we set all vertices of V as fixed points V_c. Next, we obtain the sets of transposition of two vertices from h by using Function 5. Then, we obtain a fixed-point-free automorphism by greedily obtaining transposition by using Function 6 until there are no fixed points left. The call to Function 6 is in a loop, however, the number of attempts to obtain the transpositions between vertices is at most n^2. Thus, the computational complexity of this function is $\mathcal{O}(n^6)$.

4) Obtaining a fixed-point-free automorphism if it exists: Function 5 obtain the set of transpositions of two vertices from h. The computational complexity of this function is $\mathcal{O}(n^2)$.

Function 6 obtains the transpositions from one of the fixed points. First, we set all vertices of V as

Algorithm 4 Obtaining a fixed-point-free automorphism.

function OBTAIN_A_FIXED_POINT_FREE_AUTOMORPHISM($G = (V, E)$,h)
    $Q \leftarrow$ OBTAIN_THE_SET_OF_TRANSPOSITIONS(h)
    $V_c \leftarrow V$ (fixed points)
    clear R (return value)
    while $V_c \neq \emptyset$ **do**
        $(V_c', Q, R') \leftarrow$ OBTAIN_TRANSPOSITIONS_FROM_A_VERTEX(G,Q,$v \in V_c$)
        $V_c \leftarrow V_c \cap V_c'$
        push R' to R
    end while
    return R
end function

Algorithm 5 Obtaining the set of transpositions.

function OBTAIN_THE_SET_OF_TRANSPOSITIONS(h)
    $Q \leftarrow \emptyset$
    for each $T \in h$ **do**
        for $i \leftarrow 1$ **to** $|T| - 1$ **do**
            $v_i \leftarrow i$-th vertex in T
            for $j \leftarrow i + 1$ **to** $|T|$ **do**
                $v_j \leftarrow j$-th vertex in T
                $Q \leftarrow (v_i, v_j)$
            end for
        end for
    end for
    return Q
end function

Algorithm 6 Obtaining the transpositions from a vertex.

function OBTAIN_TRANSPOSITIONS_FROM_A_VERTEX($G = (V, E)$,Q,v)
    clear R' (return value)
    $(V_c', w) \leftarrow (V, 2|V|)$
    $S_a \leftarrow G$ with all vertex weights are 0
    obtain $(v_a, v_b) \in Q$ such that $v = v_a$ or $v = v_b$
    push (v_a, v_b) to R'
    $(S_a, S_b) \leftarrow (Sg(S_a, v_a, w), Sg(S_a, v_b, w))$
    $(V_a, V_b, V_c', w, Q) \leftarrow (\{v_a\}, \{v_b\}, V_c' - \{v_a, v_b\}, w + 2|V|, Q - \{(v_a, v_b), (v_b, v_a)\})$
    loop
        $(V_a, V_b) \leftarrow$ OBTAIN_CONNECTED_VERTICES(V_a,V_b,V_c')
        if $V_a = \emptyset$ **then**
            return (V_c',Q,R')
        end if
        $(S_a, S_b, w, Q, R') \leftarrow$ OBTAIN_VERTEX_TRANSPOSITIONS(V_a,V_b,S_a,S_b,w,Q,R')
        $V_c' \leftarrow V_c' - (V_a \cup V_b)$
    end loop
end function

fixed points V_c' within this function. Next, we obtain transpositions by using Function 7 and 8. The call to Function 8 is in a loop, however, the number of attempts to obtain the transpositions between vertices is at most n^2. Thus, the computational complexity of this function is $\mathcal{O}(n^6)$.

Function 7 obtains two set of vertices V_a' and V_b'. One vertex set V_a' is adjacent to only vertices in V_a. The other vertex set V_b' is adjacent to only vertices in V_b. Vertex transpositions between V_a and V_b are accompanied by the vertex transpositions between V_a' and V_b'. The computational complexity of this function is $\mathcal{O}(n^2)$.

Function 8 obtains the vertex transpositions between V_a and V_b. Let $\lambda = Ev(Sg(S_a, v_a, w)$ with $v_a \in V_a$ and $w > 0$. Since Theorem II.4 and Corollary II.5, any $v_b \in V_b$ such that $Ev(Sg(S_b, v_b, w)) = \lambda$ is graph isomorphism. So, we obtain the transposition between v_a and v_b. Thus, by obtaining the correspondence of vertices one by one, we can obtain the transpositions between vertices belonging to V_a and vertices belonging to V_b. The computational complexity of this function is $\mathcal{O}(n^6)$.

Example III.2. *Figure 2 shows an example of obtaining a fixed-point-free automorphism in G. First, we set all vertices to fixed points and clear R and R'. Next, we obtain one vertex from fixed points. Suppose we obtain p_1. Next, we obtain the transposition of p_1 and other vertex. So, suppose we obtain (p_1, p_2). Then, we push (p_1, p_2) to R'. Next, we obtain the vertex transpositions accompanied by the previously obtained vertex transposition. So, we obtain (p_3, p_5). We push (p_3, p_5) to R'. Now, R' becomes $((p_1, p_2)$, $(p_3, p_5))$. Since there is no other transposition, we push R' to R and apply R' to G. Then, fixed points*

Algorithm 7 Obtaining the set of vertices adjacent to the vertices in V_a and the set of vertices adjacent to the vertices in V_b.

function OBTAIN_CONNECTED_VERTICES(V_a,V_b,V_c')
  $(V_a', V_b') \leftarrow (\emptyset, \emptyset)$
  for each $v \in V_c'$ **do**
    for each $v_a \in V_a$ **do**
      if $(v, v_a) \in E$ **then**
        $V_a' \leftarrow V_a' \cup \{v\}$
        break
      end if
    end for
    for each $v_b \in V_b$ **do**
      if $(v, v_b) \in E$ **then**
        $V_b' \leftarrow V_b' \cup \{v\}$
        break
      end if
    end for
  end for
  $V_d \leftarrow V_a' \cap V_b'$
  return ($V_a' - V_d$, $V_b' - V_d$)
end function

Algorithm 8 Obtaining the vertex transpositions between V_a and V_b.

function OBTAIN_VERTEX_TRANSPOSITIONS(V_a,V_b,$S_a = (V, E, z)$,S_b,w,Q,R')
  for each $v_a \in V_a$ **do**
    $\lambda \leftarrow Ev(Sg(S_a, v_a, w))$
    for each $v_b \in V_b$ **do**
      if $Ev(Sg(S_b, v_b, w)) = \lambda$ **then**
        $(Q, V_b) \leftarrow (Q - \{(v_a, v_b), (v_b, v_a)\}, V_b - \{v_b\})$
        $(S_a, S_b, w) \leftarrow (Sg(S_a, v_a, w), Sg(S_b, v_b, w), w + 2|V|)$
        push (v_a, v_b) to R'
        break
      end if
    end for
  end for
  apply R' to S_a and S_b
  return (S_a,S_b,w,Q,R')
end function

become $\{p_4\}$.

Next, we clear R'. *Next, we obtain one vertex from fixed points. Then, we obtain* p_4. *Next, we obtain the transposition of* p_4 *and other vertex. So, suppose we obtain* (p_4, p_5). *Then, we push* (p_4, p_5) *to* R'. *Next, we obtain the vertex transpositions accompanied by the previously obtained vertex transposition. So, we obtain* (p_1, p_3). *We push* (p_1, p_3) *to* R'. *Now,* R' *becomes* $((p_4, p_5), (p_1, p_3))$. *Since there is no other transposition, we push* R' *to* R *and apply* R' *to* G.

The set of fixed points become an empty set. Thus, we obtain $R = (((p_1, p_2), (p_3, p_5)), ((p_4, p_5), (p_1, p_3)))$ *as a fixed-point-free automorphism.*

5) Computational complexity: The computational complexity for determining the presence of a fixed-point-free automorphism in a graph is $\mathcal{O}(n^5)$. If fixed-point-free automorphism exists, the computational complexity of obtaining a fixed-point-free automorphism is $\mathcal{O}(n^6)$.

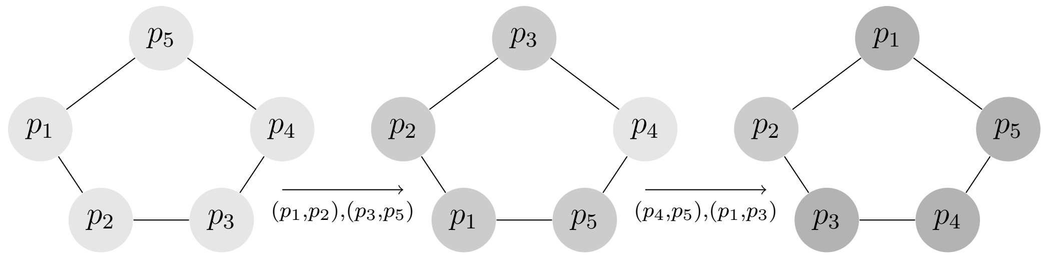

Fig. 2. An example of obtaining a fixed-point-free automorphism.

IV. Discussion

We have shown how to detect whether an input graph G has a fixed-point-free automorphism in polynomial time. After obtaining all automorphisms containing a transposition of two vertices, G has a fixed-free-point automorphism if, and only if, these automorphisms transpose all vertices. Since one of the NP-complete problems is solvable in polynomial time, the complexity classes P and NP are the same.

The method presented in this paper only shows a polynomial time complexity. Obtaining the Frobenius normal form in $\mathcal{O}(n^3)$ has been proposed [16]. Thus, the complexity of the algorithm proposed in this paper may be $\mathcal{O}(n^4)$. However, we think that the complexity will not be less than $\mathcal{O}(n^4)$. Although the method becomes in polynomial time by reducing it into other NP-complete problems in polynomial time, we do not know a lower bound on the complexity for each problem. In addition, if an input size is not so large, existing algorithms may be able to obtain the answer in less time than our proposed algorithm. We believe that the method in this paper is not to collapse the security of cryptography immediately. We think the computational complexity required for cryptography might not be a small order. For this reason, we believe that the security of encryption can be ensured by giving a sufficient input size to the encryption key.

V. Conclusion

In this paper, we have presented the theorems and an algorithm to detect whether a given graph G has a fixed-point-free automorphism. It has polynomial time complexity. Since one of the NP-complete problems is solvable in polynomial time, the complexity classes P and NP are the same.

Appendix A
Definition

In this section, we give the definitions used in this paper.

Definition A.1. A graph $G = (V, E)$ is a pair consisting of a non-empty finite vertex set $V \neq \emptyset$ and an edge set E that is a subset of V^2. The graph's size is the number of its vertices $1 < n = |V|$. The number of vertices in a graph is assumed to be finite. In addition, we align the set V with $\{v_1, \ldots, v_n\}$. There is an edge between vertices v_a and v_b when (v_a, v_b) is an element of the set E. Also, edges have no direction. Moreover, the graph has no multiple edges between a pair of vertices, and there are no loops (i.e., (v_a, v_a) is never an edge).

Definition A.2. A vertex-weighted graph $S = (V, E, z)$ is a graph with a function $z : V \to \mathbb{N}$ that gives the weights of the vertices. Then, a graph is a vertex-weighted graph in which the weights of all its vertices are 0.

Definition A.3. The adjacency matrix A of a vertex-weighted graph $S = (V, E, z)$ with $n = |V|$ is an $n \times n$ symmetric matrix that is given as follows. The entries $a_{i,j}$, $v_i, v_j \in V$, $0 < i, j \leq n$ of A satisfy:

$$\begin{cases} (v_i, v_j) \in E & \text{if} \quad a_{i,j} = a_{j,i} = 1, \\ (v_i, v_j) \notin E & \text{if} \quad a_{i,j} = a_{j,i} = 0, \\ a_{i,i} = z(v_i). \end{cases}$$

REFERENCES

[1] S. A. Cook, "The complexity of theorem-proving procedures," in *Logic, Automata, and Computational Complexity: The Works of Stephen A. Cook*. New York and NY and USA: Association for Computing Machinery, 2023, pp. 143–152.
[2] L. Fortnow, "The status of the p versus np problem," *Communications of the ACM*, vol. 52, no. 9, pp. 78–86, 2009.
[3] T. Baker, J. Gill, and R. Solovay, "Relativizations of the p = np question," *SIAM Journal on Computing*, vol. 4, no. 4, pp. 431–442, 1975. [Online]. Available: https://doi.org/10.1137/0204037
[4] A. A. Razborov and S. Rudich, "Natural proofs," in *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*. New York and NY and USA: Association for Computing Machinery, 1994, pp. 204–213.
[5] S. Aaronson and A. Wigderson, "Algebrization: A new barrier in complexity theory," *ACM Transactions on Computation Theory (TOCT)*, vol. 1, no. 1, pp. 1–54, 2009.
[6] J. Gu, P. W. Purdom, J. V. Franco, and B. W. Wah, "Algorithms for the satisfiability (sat) problem: A survey." *Satisfiability problem: Theory and applications*, vol. 35, pp. 19–152, 1996.
[7] G. J. Woeginger, "Exact algorithms for np-hard problems: A survey," in *Combinatorial Optimization—Eureka, You Shrink! Papers Dedicated to Jack Edmonds 5th International Workshop Aussois, France, March 5–9, 2001 Revised Papers*. Springer, 2003, pp. 185–207.
[8] P. Prosser, "Exact algorithms for maximum clique: A computational study," *Algorithms*, vol. 5, no. 4, pp. 545–587, 2012.
[9] P. D. Dieu *et al.*, "Average polynomial time complexity of some np-complete problems," *Theoretical computer science*, vol. 46, pp. 219–237, 1986.
[10] A. Grzesik, T. Klimošová, M. Pilipczuk, and M. Pilipczuk, "Polynomial-time algorithm for maximum weight independent set on p 6-free graphs," *ACM Transactions on Algorithms (TALG)*, vol. 18, no. 1, pp. 1–57, 2022.
[11] M. R. Garey, D. S. Johnson, and L. Stockmeyer, "Some simplified np-complete problems," in *Proceedings of the sixth annual ACM symposium on Theory of computing*. New York and NY and USA: Association for Computing Machinery, 1974, pp. 47–63.
[12] A. Lubiw, "Some np-complete problems similar to graph isomorphism," *SIAM Journal on Computing*, vol. 10, no. 1, pp. 11–21, 1981.
[13] J. A. Howell, "An algorithm for the exact reduction of a matrix to Frobenius form using modular arithmetic. I," *Mathematics of Computation*, vol. 27, no. 124, pp. 887–904, 1973.
[14] ——, "An algorithm for the exact reduction of a matrix to Frobenius form using modular arithmetic. II," *Mathematics of Computation*, vol. 27, no. 124, pp. 905–920, 1973.
[15] Y. Ohto, "Solving graph isomorphism problem in polynomial time," *FOCS*, submitted.
[16] A. Storjohann, "An o (n 3) algorithm for the frobenius normal form," in *Proceedings of the 1998 international symposium on Symbolic and algebraic computation*, 1998, pp. 101–105.