Theory Comput. Systems **35**, 81–93 (2002)
DOI: 10.1007/s00224-001-1048-9

Theory of Computing Systems

Computing Complete Graph Isomorphisms and Hamiltonian Cycles from Partial Ones*

A. Große,[1] J. Rothe,[2] and G. Wechsung[1]

[1]Institut für Informatik, Friedrich-Schiller-Universität Jena, 07740 Jena, Germany
{grosse, wechsung}@informatik.uni-jena.de

[2]Mathematisches Institut, Heinrich-Heine-Universität Düsseldorf, 40225 Düsseldorf, Germany
rothe@cs.uni-duesseldorf.de

Abstract. We prove that computing a single pair of vertices that are mapped onto each other by an isomorphism φ between two isomorphic graphs is as hard as computing φ itself. This result optimally improves upon a result of Gál, Halevi, Lipton, and Petrank. We establish a similar, albeit slightly weaker, result about computing complete Hamiltonian cycles of a graph from partial Hamiltonian cycles.

1. Introduction

Two of the most central and well-studied problems in NP are the graph isomorphism problem and the Hamiltonian cycle problem. The latter problem is one of the standard NP-complete problems [Ka], [GJ]. In contrast, the graph isomorphism problem currently is the most prominent candidate of a problem that is neither in P nor NP-complete. On the one hand, there is no efficient algorithm known for solving this problem, despite a considerable effort in the past to design such algorithms. On the other hand, due to its well-known lowness properties [Sc3], [KST1], the graph isomorphism problem is very unlikely to be NP-complete. For more information about the graph isomorphism problem, we refer to the book by Köbler et al. [KST2].

* This work was supported in part by Grant NSF-INT-9815095/DAAD-315-PPP-gü-ab. The second author was supported in part by a Heisenberg Fellowship of the Deutsche Forschungsgemeinschaft. A preliminary version of this paper appears as part of [GRW] in the *Proceedings of the Seventh Italian Conference on Theoretical Computer Science*, 2001.

Computational complexity theory and, in particular, the theory of NP-completeness [GJ] traditionally is concerned with the decision versions of problems. For practical purposes, however, to find or to construct a solution of a given NP problem is much more important than merely knowing whether or not a solution exists. For example, computing an isomorphism between two isomorphic graphs (that is, solving the search version of the graph isomorphism problem) is much more important for most applications than merely knowing that the graphs are isomorphic. Therefore, much effort has been made in the past to relate the complexity of solving the search problem to the complexity of solving the corresponding decision problem. This property is known as "search reducing to decision," see, e.g., [HNOS] and the references cited therein. The decisive property enabling search to reduce to decision for NP problems such as the graph isomorphism problem is their self-reducibility.

The present paper builds on the recent work of Gál et al. [GHLP] who studied a property that might be dubbed "complete search reducing to partial search." They showed for various NP problems A that, given an input $x \in A$, computing a small fraction of a solution for x is no easier than computing a complete solution for x. For example, given two isomorphic graphs, computing roughly logarithmically many pairs of vertices that are mapped onto each other by a complete isomorphism φ between the graphs is as hard as computing φ itself.

As Gál et al. note, their results have two possible interpretations. Positively speaking, their results say that to solve the complete search problem efficiently it is enough to provide an efficient algorithm for computing only a small part of a solution. Negatively speaking, their results say that constructing even a small part of a solution to instances of hard problems also appears to be a very difficult task. As Gál et al. [GHLP] further note, their work also has consequences with regard to fault-tolerant computing (in particular, for recovering the complete problem solution when parts of it are lost during transmission), and for constructing robust proofs of membership.

The present paper makes the following contributions. Firstly, we improve the above-mentioned result of Gál et al. [GHLP] by showing that computing even a single pair of vertices that are mapped onto each other by a complete isomorphism φ between two isomorphic graphs is as hard as computing φ itself. This result is a considerable strengthening of the previous result and an optimal improvement. Interestingly, the self-reducibility of the graph isomorphism problem is the key property that makes our stronger result possible. Secondly, we obtain a similar, albeit somewhat weaker, result about computing complete Hamiltonian cycles of a given graph from accessing partial information about the graph's Hamiltonian cycles.

2. Computing Complete Graph Isomorphisms from Partial Ones

2.1. *Main Result*

Gál et al. [GHLP] prove the following result. Suppose there exists a function oracle f that, given any two isomorphic graphs with m vertices each, outputs a part of an isomorphism between the graphs consisting of at least $(3+\varepsilon)\log m$ vertices for some

constant $\varepsilon > 0$. Then, using the oracle f, one can compute a complete isomorphism between any two isomorphic graphs in polynomial time.

We improve their result by showing the same consequence under the weakest assumption possible: assuming that we are given a function oracle that provides *only one vertex pair* belonging to an isomorphism between two given isomorphic graphs, one can use this oracle to compute complete isomorphisms between two isomorphic graphs in polynomial time. Thus, our improvement of the previous result by Gál et al. [GHLP] is optimal.

Definition 2.1. Let G and H be undirected and simple graphs, i.e., graphs with no reflexive and multiple edges.

- The *vertex set of* G is denoted by $V(G)$, and the *edge set of* G is denoted by $E(G)$.
- An *isomorphism between* G *and* H is a bijective mapping φ from $V(G)$ onto $V(H)$ such that, for all $x, y \in V(G)$,

  $$\{x, y\} \in E(G) \iff \{\varphi(x), \varphi(y)\} \in E(H).$$

- Let $\mathrm{ISO}(G, H)$ denote the set of isomorphisms between G and H.

We now state our main result.

Theorem 2.2. *Suppose there exists a function oracle f that, given any two isomorphic graphs $\hat{G}$ and $\hat{H}$, outputs two vertices $x \in V(\hat{G})$ and $y \in V(\hat{H})$ with $\hat{\varphi}(x) = y$, for some isomorphism $\hat{\varphi}$ from* $\mathrm{ISO}(\hat{G}, \hat{H})$. *Then there is a recursive procedure g that, given any two isomorphic graphs G and H, uses the oracle f to construct a complete isomorphism $\varphi \in \mathrm{ISO}(G, H)$ in polynomial time.*

2.2. *Discussion of the Model*

As stated above, our main result optimally improves upon the above-mentioned result of Gál et al. [GHLP]. It is thus clear that, in stating Theorem 2.2, we have to use precisely the same model of accessing partial information via a function oracle that is used in [GHLP]. Two remarks on that model are in order. The first remark concerns the way the function oracle is modeled. Gál et al. write in [GHLP]:

> "We study the complexity of the graph isomorphism problem assuming that we have access to an oracle that provides us with a partial isomorphism on a subset of certain size of the vertices for arbitrary isomorphic pairs of graphs. We stress that the partial information which is provided by the oracle must be part of a complete isomorphism between the graphs G and H."

As noted by a referee, both a preliminary version of the present paper (see also [GRW]) and the paper by Gál et al. [GHLP] are concerned with a certain kind of "promise oracle" that gives reliable information only in the case of an isomorphic pair of input graphs. To remove the "promise" from the oracle and to be also explicit about the case of two nonisomorphic input graphs, we follow this referee's suggestion to define by convention that in that case the function oracle simply outputs a pair of vertices, one from each given graph, without revealing that the graphs are nonisomorphic. Note that,

even though the "promise" is now explicitly removed from the oracle, it is enough to state Theorem 2.2 for the case of two isomorphic input graphs. One may do so, since after having constructed, according to Theorem 2.2, a potential isomorphism φ between the given graphs G and H, one can easily deterministically check it to verify that φ indeed is an isomorphism between G and H.

The second remark concerns the way in which access to the function oracle is modeled. Both the algorithm presented in Theorem 3 of [GHLP] and the recursive procedure g from Theorem 2.2 may access the oracle a polynomial number of times. As noted by a referee, it may be questioned whether this model is realistic in the setting of fault-tolerant computing, where the objective is to recover a complete solution of a hard problem, parts of which are lost during transmission. In this particular scenario, it would indeed appear more realistic to require that the oracle can be accessed only once. Unfortunately, neither the work by Gál et al. [GHLP] nor our work yields results in this very restricted model when the oracle can be asked only once. On the other hand, there are other good motivations as well for which it does make sense to allow a polynomial number of oracle queries. For example, this comment applies to the question of whether the complete search problem reduces to partial search, and it applies to the task of constructing robust proofs of membership (see [GHLP]).

Analogous comments about the model used apply to Section 3 in which similar results on the Hamiltonian cycle problem are established.

2.3. *Informal Description of the Proof of the Main Result*

Before proving Theorem 2.2, we give an informal description of the proof and we explain the main difference between our proof and the proof of Gál et al. [GHLP]. Crucially, to make their recursive procedure terminate, they ensure in their construction that the (pairs of) graphs they construct are of strictly decreasing size in each loop of the procedure. In contrast, for our algorithm this strong requirement is not necessary to make the procedure terminate.

We informally explain why. Our algorithm is inspired by the known self-reducibility algorithm for the graph isomorphism problem; see, e.g., [KST2]. The notion of self-reducibility has been thoroughly studied by many authors; we refer the reader to the work of Schnorr [Sc1], [Sc2], Meyer and Paterson [MP], Selman [Se], and Ko [Ko], and to the excellent survey by Joseph and Young [JY] for an overview and for pointers to the literature.

Informally speaking, a self-reduction for a problem A is a computational procedure for solving A, where the set A itself may be accessed as an oracle. To prevent this notion from being trivialized, one requires that A cannot be queried about the given input itself; usually, only queries about strings that are "smaller" than the input string are allowed. When formally defining what precisely is meant by "smaller," most self-reducibility notions—including those studied by the above-mentioned researchers—employ the useful concepts of "polynomially well-founded" and "length-bounded" partial orders, rather than being based simply on the lengths of strings. This approach is useful in order to "obtain full generality and to preserve the concept under polynomially computable isomorphisms" [JY, p. 84], see also [MP] and [Se]. That means that the strings queried in a self-reduction may be *larger in length* than the input strings as long as they are

predecessors in a polynomially well-founded and length-bounded partial order. It is this key property that makes our algorithm terminate without having to ensure in the construction that the (pairs of) graphs constructed are of strictly decreasing size in each loop.

Here is an intuitive description of how our algorithm works. Let G and H be the given isomorphic graphs. The function oracle will be invoked in each loop of the procedure to yield any one pair of vertices that are mapped onto each other by some isomorphism between the graphs as yet constructed. However, if we were simply deleting this vertex pair, we would obtain new graphs $\hat{G}$ and $\hat{H}$ such that $\mathrm{ISO}(\hat{G}, \hat{H})$ might contain some isomorphism not compatible with $\mathrm{ISO}(G, H)$, which means it cannot be extended to an isomorphism in $\mathrm{ISO}(G, H)$. That is why our algorithm will attach cliques of appropriate sizes to each vertex to be deleted, and the deletion of this vertex, and of the clique attached to it, will be delayed until some subsequent loop of the procedure. That is, the (pairs of) graphs we construct may increase in size in some of the loops, and yet the procedure is guaranteed to terminate in polynomial time.

2.4. *Proof of the Main Result*

We now turn to the formal proof.

Proof of Theorem 2.2. Let G and H be two given isomorphic graphs with n vertices each. Let f be a function oracle as in the theorem. We describe the recursive procedure g that computes an isomorphism $\varphi \in \mathrm{ISO}(G, H)$. Below, we use variables $\hat{G}$ and $\hat{H}$ to denote (encodings of) graphs obtained from G and H according to g, and we refer to the vertices of G and H as the *old* vertices and to the vertices of $\hat{G} - G$ and $\hat{H} - H$ as the *new* vertices.

On input $\langle G, H\rangle$, the algorithm g executes the following steps:

1. Let $\hat{G} = G$ and $\hat{H} = H$, and set i to $n = ||V(G)||$. Let $\varphi \subseteq V(G) \times V(H)$ be a set variable that, eventually, gives the isomorphism between G and H to be constructed. Initially, set φ to the empty set.
2. Query f about the pair $(\hat{G}, \hat{H})$. Let (x, y) be the vertex pair returned by $f(\hat{G}, \hat{H})$, where $x \in V(\hat{G})$ and $y \in V(\hat{H})$ and $\hat{\varphi}(x) = y$ for some isomorphism $\hat{\varphi} \in \mathrm{ISO}(\hat{G}, \hat{H})$.
3. Consider the following two cases:

   Case 3.1: $x \in V(G)$ *is an old vertex*. We distinguish the following two cases:
   (a) y is also an old vertex (in H).
   Set φ to $\varphi \cup \{(x, y)\}$. Modify the graphs $\hat{G}$ and $\hat{H}$ as follows.

   Delete x, all new neighbors of x, and all edges incident to either of these vertices from $\hat{G}$. Attach to each old neighbor $x' \in V(G)$ of x a copy of a clique $C_{i,x'}$ consisting of $i - 1$ new vertices, each of which is connected with x' by an edge; hence, the graph induced by $V(C_{i,x'}) \cup \{x'\}$ forms an i-clique. Make sure that all the new clique vertices are pairwise disjoint and disjoint with (the old) graph $\hat{G}$. Call the resulting graph (the new) $\hat{G}$.

   Modify $\hat{H}$ in the same way: Delete y and all new neighbors of y from $\hat{H}$, and extend each old neighbor $y' \in V(H)$ of y to a clique consisting of the i vertices $V(C_{i,y'}) \cup \{y'\}$.

(b) y is a new vertex in $\hat{H}$.
Let $\tilde{y} \in V(H)$ be the unique old vertex adjacent to y, i.e., y is a member of the clique $C_{j,\tilde{y}}$ that was previously attached to $\tilde{y}$ in the $(j-n+1)$th loop, where $n \leq j < i$. Note that the size of the clique $C_{j,\tilde{y}} \cup \{\tilde{y}\}$ equals j. Since $\hat{\varphi}(x) = y$, the old vertex x must belong to the clique $C_{j,x} \cup \{x\}$ of size j and, thus, cannot have any old neighbors in $\hat{G}$. It follows that $\tilde{y}$ is also not adjacent to any old vertex in the current graph $\hat{H}$. That is, both the clique $C_{j,x} \cup \{x\}$ and the clique $C_{j,\tilde{y}} \cup \{\tilde{y}\}$ are connecting components of their graphs $\hat{G}$ and $\hat{H}$, respectively.

Set φ to $\varphi \cup \{(x, \tilde{y})\}$. Modify the graphs $\hat{G}$ and $\hat{H}$ by deleting the cliques $C_{j,x} \cup \{x\}$ and $C_{j,\tilde{y}} \cup \{\tilde{y}\}$.

Set i to $i+1$.

Case 3.2: $x \notin V(G)$ *is a new vertex in* $\hat{G}$. It follows that x is a member of a clique $C_{j,\tilde{x}}$, where $n \leq j < i$, that was previously attached to some old vertex $\tilde{x} \in V(G)$ in the $(j-n+1)$th loop. Also, by construction, $\tilde{x}$ is the only old vertex adjacent to x. Similarly, it holds that y is a member of a clique $C_{j,\tilde{y}} \cup \{\tilde{y}\}$ in $\hat{H}$ with a uniquely determined old vertex $\tilde{y} \in V(H)$.

If $y = \tilde{y}$, then this case reduces to Case 3.1(a), with x being replaced by $\tilde{x}$.

If $y \neq \tilde{y}$, then $\hat{\varphi}(x) = y$ implies that $\hat{\varphi}(\tilde{x}) = \tilde{y}$ and, thus, that $\tilde{x}$ and $\tilde{y}$ have the same number of old neighbors. Hence, this case also reduces to Case 3.1(a), with x being replaced by $\tilde{x}$ and y being replaced by $\tilde{y}$.

4. If there are no vertices left in $\hat{G}$ and $\hat{H}$, output φ, which gives a complete isomorphism between G and H. Otherwise, go to Step 2.

As alluded to in the above informal description of the algorithm, the intuition behind introducing cliques of increasing sizes in the construction is to keep the isomorphisms $\hat{\varphi} \in \mathrm{ISO}(\hat{G}, \hat{H})$ compatible with $\varphi \in \mathrm{ISO}(G, H)$ when vertices from G and H are deleted. That is, we want to preclude the case that deleting $x \in V(G)$ and $y \in V(H)$ results in reduced graphs $\hat{G}$ and $\hat{H}$ such that there is some $\hat{\varphi} \in \mathrm{ISO}(\hat{G}, \hat{H})$—and our oracle f might pick some vertex pair corresponding to such a $\hat{\varphi}$—that cannot be extended to $\varphi \in \mathrm{ISO}(G, H)$.

The following example illustrates this intuition and shows how the algorithm works.

Example 2.3. Figure 1 gives an example of a pair of isomorphic graphs G and H with $\mathrm{ISO}(G, H) = \{\varphi_1, \varphi_2\}$, where

$$\varphi_1 = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 1 & 5 & 4 & 3 & 2 \end{pmatrix} \quad \text{and} \quad \varphi_2 = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 5 & 1 & 4 & 3 & 2 \end{pmatrix}.$$

Suppose that the function oracle f, when queried about the pair (G, H), returns, e.g., the vertex pair $(5, 2)$. If we were simply deleting vertex 5 from G and vertex 2 from H, then we would obtain graphs $\widehat{G}$ and $\widehat{H}$ such that $\mathrm{ISO}(\widehat{G}, \widehat{H})$ contains six isomorphisms, only two of which are compatible with the pair $(5, 2)$; see Figure 2. However, then f, when queried about the pair $(\widehat{G}, \widehat{H})$, might pick, e.g., the vertex pair $(4, 5)$, which belongs neither to φ_1 nor to φ_2.

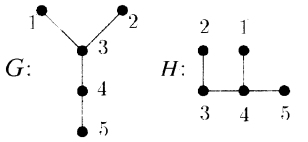

Fig. 1. Two graphs G and H with $\mathrm{ISO}(G, H) = \{\varphi_1, \varphi_2\}$.

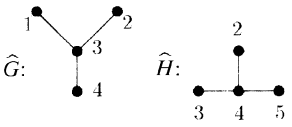

Fig. 2. Two graphs $\widehat{G}$ and $\widehat{H}$ for which $\mathrm{ISO}(\widehat{G}, \widehat{H})$ contains isomorphisms not compatible with the pair $(5, 2)$.

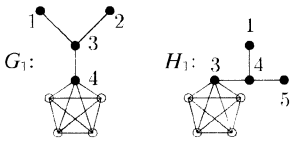

Fig. 3. Two graphs G_1 and H_1 obtained from G and H according to g when $f(G, H)$ returns $(5, 2)$.

To preclude cases like this, our algorithm attaches cliques of size 5 to vertex 4 in G and to vertex 3 in H; see Figure 3. Old vertices are represented by solid circles and new vertices by empty circles. Note that each $\varphi \in \mathrm{ISO}(G_1, H_1)$ is compatible with the vertex pair $(5, 2)$ from $\varphi_1, \varphi_2 \in \mathrm{ISO}(G, H)$.

Figures 3–6 show how g, on input $\langle G, H\rangle$, continues to work for a specific sequence of oracle answers from f. In Figure 6 the only old vertex left in G_4 is vertex 4, and the only old vertex left in H_4 is vertex 3. Hence, whichever vertex pair f when queried about (G_4, H_4) picks, g maps vertex 4 in G_4 to vertex 3 in H_4, which completes the isomorphism

$$\varphi_2 = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 5 & 1 & 4 & 3 & 2 \end{pmatrix}$$

that is in $\mathrm{ISO}(G, H)$. Finally, both G_4 and H_4 are deleted, and the algorithm terminates.

To prove the correctness of the algorithm, we argue that

(a) each pair $\langle \hat{G}, \hat{H}\rangle$ constructed in any loop of g is a pair of isomorphic graphs—hence, f can legally be called in each loop of g; and
(b) the mapping φ computed by g on input $\langle G, H\rangle$ is in $\mathrm{ISO}(G, H)$.

Proof of (a). This assertion follows immediately from the construction and the assumption that G and H are isomorphic.

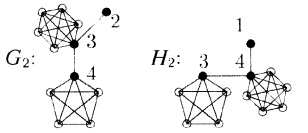

Fig. 4. Two graphs G_2 and H_2 that result from $f(G_1, H_1) = (1, 5)$.

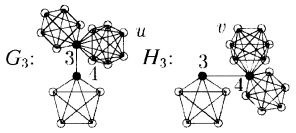

Fig. 5. Two graphs G_3 and H_3 that result from $f(G_2, H_2) = (2, 1)$.

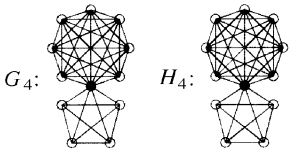

Fig. 6. Two graphs G_4 and H_4 that result from $f(G_3, H_3) = (u, v)$.

Proof of (b). The first call to f yields a valid initial segment (x_1, y_1) of an isomorphism between G and H, since f is queried about the unmodified graphs G and H.

Let $\varphi_i = \{(x_1, y_1), (x_2, y_2), \ldots, (x_i, y_i)\}$ be the initial segment of φ that consists of i vertex pairs for some i, $1 \leq i \leq n$, where (x_i, y_i) is the pair added in the ith loop of g. Let G_i and H_i be the graphs constructed from G and H when loop i is entered; for example, $G_1 = G$ and $H_1 = H$. Fix some i with $1 < i \leq n$. We show that the extension φ_i of φ_{i-1} (obtained by adding the pair (x_i, y_i) in the ith loop of g) is compatible with φ_{i-1}. That is, for each $(x_j, y_j) \in \varphi_{i-1}$, it holds that

$$\{x_i, x_j\} \in E(G) \qquad \text{if and only if} \quad \{y_i, y_j\} \in E(H).$$

Assume $\{x_i, x_j\} \in E(G)$. In loop $j < i$, all neighbors of x_j, including x_i, and all neighbors of y_j were extended to a clique of size $n + j - 1$. Note that, in each loop of g, the clique sizes are increased by one, each clique contains exactly one old vertex, and any two cliques in G_i (respectively, in H_i) can overlap only by having their unique old vertex in common. It follows that any isomorphism between G_i and H_i must map cliques of size $n + j - 1$ in G_i onto cliques of size $n + j - 1$ in H_i. Since y_i is chosen in loop i of g, it follows from our construction that the clique C_{n+j-1,x_i} in G_i was mapped onto the clique C_{n+j-1,y_i} in H_i. Hence, y_i is a neighbor of y_j in H, i.e., $\{y_i, y_j\} \in E(H)$.

The converse implication ($\{y_i, y_j\} \in E(H) \implies \{x_i, x_j\} \in E(G)$) follows by a symmetric argument.

Finally, we estimate the time complexity of the algorithm g. Since, in each loop of g, a pair of old vertices from $V(G) \times V(H)$ is deleted from the graphs and is added to the isomorphism $\varphi \in \mathrm{ISO}(G, H)$, the algorithm terminates after n loops. Within each loop, g makes one oracle call to f, updates φ, and modifies the current graphs $\hat{G}$ and $\hat{H}$ by deleting certain vertices and by adding at most $2(n-1)$ cliques of size at most $2n-1$. Hence, g runs in cubic time. □

3. Computing Complete Hamiltonian Cycles from Partial Ones

3.1. *Informal Description of the Result and Its Proof*

We now turn to the problem of computing complete Hamiltonian cycles in a graph from partial ones. Our result here is similar to the one presented for the graph isomorphism problem in Section 2, although it is technically slightly weaker. The remarks about the model of oracle access that we made in Section 2.2 apply here as well. Our construction for the Hamiltonian cycle problem is easier to describe when we use multigraphs, i.e., graphs with reflexive and multiple edges allowed. We may do so, since for Hamiltonian cycles it does not matter whether simple graphs or multigraphs are used. That is not to say that the two corresponding problems are equivalent in the sense of a reduction; rather, it refers to the simple fact that it does not matter through which one of possibly several edges between two adjacent vertices a Hamiltonian cycle goes. Our result and proof could equally well be given for simple graphs, but the proof would be technically more involved and more cumbersome to describe in its technical details. Unless stated otherwise, we also assume that all graphs are connected and, as before, are undirected.

A referee observed that in the case of directed graphs, there is an easy procedure for obtaining Hamiltonian cycles, which in fact is a particular case of the "left-right-context"

idea that we apply below to the case of undirected graphs. Following this referee's suggestion, we present this procedure for directed graphs here in order to motivate our solution for the undirected case. Suppose that the oracle outputs a directed edge (u, v) that is part of a Hamiltonian cycle of the given graph G. Then deleting this edge and all edges going out of u and all edges going into v and identifying u and v, we obtain a new graph H such that each Hamiltonian cycle in H can be extended to a Hamiltonian cycle of G that contains the edge (u, v). Hence, the Hamiltonian cycles in the new graph H are compatible with the edge (u, v), and thus we may recurse to construct a complete Hamiltonian cycle of the given graph G in polynomial time.

Turning now to the case of undirected graphs, we informally describe how our procedure works. As in the preceding section, suppose we have a function oracle f that, given any multigraph G that contains a Hamiltonian cycle, returns an edge e that is part of a Hamiltonian cycle c of G. We want to reduce G by deleting e and identifying the two vertices incident to e, and then want to recursively apply f to this reduced graph, call it $\hat{G}$. However, this approach would destroy important information about e, namely the "left" and the "right" context of e in G. Thus, in the next recursion loop, the oracle might return an edge contained in a Hamiltonian cycle $\hat{c}$ of $\hat{G}$ that is not compatible with the previously chosen edge e, which means that adding e back to $\hat{G}$ does not necessarily imply that $\hat{c}$ can be extended to a Hamiltonian cycle of G. To preclude cases like this, we require our oracle to return only edges contained in Hamiltonian cycles that are compatible with the left-right-context of the edges previously chosen. This additional requirement regarding f makes Theorem 3.2 somewhat weaker than Theorem 2.2.

First, we define what we mean by a left-right-context of (the edges of) G, and what we mean by Hamiltonian cycles being compatible (or consistent) with a left-right-context of G.

Definition 3.1. Let $G = (V, E)$ be an undirected multigraph with n vertices.

- A *Hamiltonian cycle of* G is a sequence $(v_1, v_2, \ldots, v_n)$ of pairwise distinct vertices from V such that $\{v_n, v_1\} \in E$ and $\{v_i, v_{i+1}\} \in E$ for each i with $1 \leq i \leq n-1$.
- For any set S, let $\mathfrak{P}(S)$ denote the power set of S. For any $v \in V$, let $E(v)$ denote the set of edges in E incident to v.
  A *left-right-context of* G is a function $\pi : V \to \mathfrak{P}(E) \times \mathfrak{P}(E)$ satisfying that, for every $v \in \text{domain}(\pi)$, there exist sets $L(v)$ and $R(v)$ such that
  1. $\pi(v) = (L(v), R(v))$,
  2. $L(v) \cup R(v) \subseteq E(v)$, and
  3. $L(v) \cap R(v) = \emptyset$.
- We say that a Hamiltonian cycle c of G is *consistent with a left-right-context* π *of* G if and only if for every $v \in \text{domain}(\pi)$, c contains exactly one edge from $L(v)$ and exactly one edge from $R(v)$, where $\pi(v) = (L(v), R(v))$.

3.2. *Formal Result and Proof*

We now state and prove our result formally.

Theorem 3.2. *Let $\hat{G}$ be any multigraph, and let π be any left-right-context of $\hat{G}$. Suppose there exists a function oracle f that, given $(\hat{G}, \pi)$, outputs some edge $e \in E(\hat{G})$ such that some Hamiltonian cycle consistent with π contains e (provided $\hat{G}$ has a Hamiltonian cycle consistent with π). Then there is a recursive procedure g that, given any multigraph G that has a Hamiltonian cycle, uses the oracle f to construct a complete Hamiltonian cycle of G in polynomial time.*

Proof. Let G be any multigraph with n vertices that contains a Hamiltonian cycle. Let f be a function oracle as in the theorem.

In the procedure described below, whenever we identify two vertices u and v, deleting the edge(s) connecting u and v, we assume by convention that in the resulting graph the vertex $u = v$ has two name tags, namely u and v. This convention simplifies the description of our construction and does no harm.

We now describe the procedure g on input G:

Step 0. Let $G_0 = (V_0, E_0)$ be the given multigraph G, and let π_0 be the nowhere defined function (on the domain V_0). Set C to the empty set. Note that C will, eventually, contain the complete Hamiltonian cycle of G to be constructed.

Step i, $1 \leq i \leq n-1$. Let $G_{i-1} = (V_{i-1}, E_{i-1})$ be the multigraph and let π_{i-1} be the left-right-context of G_{i-1} constructed in the previous step. Compute the edge $e_i = f(G_{i-1}, \pi_{i-1})$ by querying the oracle, and add e_i to C. Let $e_i = \{u_i, v_i\}$. Consider the following three cases.

Case 1: $e_i \cap \text{domain}(\pi_{i-1}) = \emptyset$. Cancel e_i from G_{i-1}, and identify the vertices u_i and v_i. Call the resulting graph $G_i = (V_i, E_i)$. Define the left-right-context π_i: $V_i \to \mathfrak{P}(E_i) \times \mathfrak{P}(E_i)$ by $\text{domain}(\pi_i) = \text{domain}(\pi_{i-1}) \cup \{u_i\}$ and

$$\pi_i(v) = \begin{cases} \pi_{i-1}(v) & \text{if} \quad v \in \text{domain}(\pi_{i-1}), \\ (L_i(u_i), R_i(u_i)) & \text{if} \quad v = u_i, \end{cases}$$

where

- $L_i(u_i) = E_{i-1}(u_i) - \{e_i\}$ and
- $R_i(u_i) = \{\{u_i, z\} \mid \{v_i, z\} \in E_{i-1} \wedge z \neq u_i\}$.

Case 2: $e_i \cap \text{domain}(\pi_{i-1}) = \{x\}$ *for some vertex* $x \in V_{i-1}$. By our assumption that f returns only edges consistent with the given left-right-context, e_i must belong to exactly one of $L_{i-1}(x)$ or $R_{i-1}(x)$. Assume $x = v_i$ and $e_i \in L_{i-1}(x)$; the other cases—such as the case "$x = u_i$ and $e_i \in R_{i-1}(x)$"—can be treated analogously.

Cancel e_i from G_{i-1}, and identify the vertices u_i and v_i, which equals x. Call the resulting graph $G_i = (V_i, E_i)$. Define the left-right-context π_i: $V_i \to \mathfrak{P}(E_i) \times \mathfrak{P}(E_i)$ by $\text{domain}(\pi_i) = \text{domain}(\pi_{i-1})$ and

$$\pi_i(v) = \begin{cases} \pi_{i-1}(v) & \text{if} \quad v \neq x, \\ (L_i(x), R_i(x)) & \text{if} \quad v = x, \end{cases}$$

where

- $L_i(x) = \{\{x, z\} \mid \{u_i, z\} \in E_{i-1} \ \wedge \ z \neq v_i\}$ and
- $R_i(x) = R_{i-1}(x)$.

Case 3: $e_i \cap \text{domain}(\pi_{i-1}) = \{x, y\}$ *for two vertices* $x, y \in V_{i-1}$ *with* $x \neq y$. It follows that $e_i = \{x, y\}$ in this case. By our assumption that f returns only edges consistent with the given left-right-context, e_i must belong to exactly one of $L_{i-1}(z)$ or $R_{i-1}(z)$, for both $z = x$ and $z = y$. Assume $e_i \in L_{i-1}(x) \cap R_{i-1}(y)$; the other cases can be treated analogously.

Cancel e_i from G_{i-1}, and identify the vertices x and y. Call the resulting graph $G_i = (V_i, E_i)$. Define the left-right-context $\pi_i \colon V_i \to \mathfrak{P}(E_i) \times \mathfrak{P}(E_i)$ by $\text{domain}(\pi_i) = \text{domain}(\pi_{i-1})$ and

$$\pi_i(v) = \begin{cases} \pi_{i-1}(v) & \text{if} \quad v \neq x = y, \\ (L_i(y), R_i(y)) & \text{if} \quad v = x = y, \end{cases}$$

where

- $L_i(y) = L_{i-1}(y)$ and
- $R_i(y) = \{\{y, z\} \mid \{x, z\} \in R_{i-1}(x)\}$.

Step n. Since in each of the $n - 1$ previous steps two vertices have been identified and one edge has been added to C, the graph G_{n-1} constructed in the previous step contains only one vertex, say z, having possibly multiple reflexive edges. Also, C contains $n - 1$ elements, and π_{n-1} is either of the form

- $\pi_{n-1} = (\emptyset, R_{n-1}(z))$ or
- $\pi_{n-1} = (L_{n-1}(z), \emptyset)$,

where any edge in $R_{n-1}(z)$ (respectively, in $L_{n-1}(z)$) can be used to complete the Hamiltonian cycle constructed so far. Thus, we may choose any one edge from $R_{n-1}(z)$ (respectively, from $L_{n-1}(z)$) and add it to C.

This concludes the description of the procedure g. Note that g runs in polynomial time. To prove the correctness of the algorithm, note that, for each $i \in \{1, 2, \ldots, n-2\}$, and for each Hamiltonian cycle c of G_i consistent with π_i, it holds that inserting the edge e_i into c yields a Hamiltonian cycle of G_{i-1}, thus ensuring consistency of the overall construction. □

4. Conclusions and Future Work

In this paper we studied an important property of NP problems: how to compute complete solutions from partial solutions. We in particular studied the graph isomorphism problem and the Hamiltonian cycle problem. We showed as Theorem 2.2 that computing even a single pair of vertices belonging to an isomorphism between two isomorphic graphs is as

hard as computing a complete isomorphism between the graphs. Theorem 2.2 optimally improves upon a result of Gál et al. [GHLP].

We propose to establish analogous results for NP problems other than the graph isomorphism problem. For example, Gál et al. [GHLP] investigated many more hard NP problems, and showed that computing partial solutions for them is as hard as computing complete solutions. However, their results are not known to be optimal, which leaves open the possibility of improvement. Relatedly, what impact does the self-reducibility of such problems have for reducing complete search to partial search?

We obtained as Theorem 3.2 a similar result about reducing complete search to partial search for the Hamiltonian cycle problem. However, this result appears to be slightly weaker than Theorem 2.2, since in Theorem 3.2 we require a stronger hypothesis about the function oracle used. Whether this stronger hypothesis in fact is necessary remains an open question. It would be interesting to know whether, also for the Hamiltonian cycle problem, one can prove a result as strong as Theorem 2.2. More precisely, is it possible to prove the same conclusion as in Theorem 3.2 when we are given a function oracle that is merely required to return any one edge of a Hamiltonian cycle of the given graph, without requiring in addition that the edge returned belong to a Hamiltonian cycle consistent with the edge's left-right-context?

Acknowledgment

We are indebted to Edith and Lane A. Hemaspaandra for introducing us to this interesting topic and for stimulating discussions and comments. We acknowledge interesting discussions about graph theory with Haiko Müller. We thank the anonymous ICTCS '01 referees for their helpful and insightful comments on the paper, and we thank an anonymous *TOCS* referee for his or her very helpful and insightful comments and suggestions that improved the presentation of this paper. In particular, the discussion in Section 2.2 about the model of oracle access used was inspired by this referee's comments, and the solution for the Hamiltonian cycle problem for directed graphs that is given in Section 3.1 is due to this referee.

References

[GHLP] A. Gál, S. Halevi, R. Lipton, and E. Petrank. Computing from partial solutions. In *Proceedings of the* 14*th Annual IEEE Conference on Computational Complexity*, pages 34–45. IEEE Computer Society Press, Los Alamitos, CA, May 1999.

[GJ] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, CA, 1979.

[GRW] A. Große, J. Rothe, and G. Wechsung. Relating partial and complete solutions and the complexity of computing smallest solutions. In *Proceedings of the Seventh Italian Conference on Theoretical Computer Science*, pages 339–356. Lecture Notes in Computer Science #2202, Springer-Verlag, Berlin, October 2001.

[HNOS] E. Hemaspaandra, A. Naik, M. Ogihara, and A. Selman. P-selective sets and reducing search to decision vs. self-reducibility. *Journal of Computer and System Sciences*, 53(2):194–209, 1996.

[JY] D. Joseph and P. Young. Self-reducibility: effects of internal structure on computational complexity. In A. Selman, editor, *Complexity Theory Retrospective*, pages 82–107. Springer-Verlag, New York, 1990.

[Ka] R. Karp. Reducibilities among combinatorial problems. In R. Miller and J. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103, 1972.

[Ko] K. Ko. On self-reducibility and weak P-selectivity. *Journal of Computer and System Sciences*, 26(2):209–221, 1983.

[KST1] J. Köbler, U. Schöning, and J. Torán. Graph isomorphism is low for PP. *Computational Complexity*, 2:301–330, 1992.

[KST2] J. Köbler, U. Schöning, and J. Torán. *The Graph Isomorphism Problem*: *Its Structural Complexity*. Birkhäuser, Basel, 1993.

[MP] A. Meyer and M. Paterson. With what frequency are apparently intractable problems difficult? Technical Report MIT/LCS/TM-126, MIT Laboratory for Computer Science, Cambridge, MA, 1979.

[Sc1] C. Schnorr. Optimal algorithms for self-reducible problems. In S. Michaelson and R. Milner, editors, *Proceedings of the* 3*rd International Colloquium on Automata*, *Languages*, *and Programming*, pages 322–337, University of Edinburgh, July 1976. Edinburgh University Press, Edinburgh.

[Sc2] C. Schnorr. On self-transformable combinatorial problems, 1979. Presented at IEEE Symposium on Information Theory, Udine, and Symposium über Mathematische Optimierung, Oberwolfach.

[Sc3] U. Schöning. Graph isomorphism is in the low hierarchy. *Journal of Computer and System Sciences*, 37:312–323, 1987.

[Se] A. Selman. Natural self-reducible sets. *SIAM Journal on Computing*, 17(5):989–996, 1988.

Received June 18, 2001, *and in revised form October* 29, 2001. *Online publication February* 20, 2002.