Electronic Colloquium on Computational Complexity, Revision 1 of Report No. 162 (2015)

A note on Graph Automorphism and Smart Reductions

Eric Allender* Joshua A. Grochow† Dieter van Melkebeek‡
Cristopher Moore§ Andrew Morgan¶

February 9, 2018

Abstract

It is well-known [KST93] that the complexity of the Graph Automorphism problem is characterized by a special case of Graph Isomorphism, where the input graphs satisfy the "promise" of being rigid (that is, having no nontrivial automorphisms). In this brief note, we observe that the reduction of Graph Automorphism to the Rigid Graph Isomorphism problem can be accomplished even using Grollman and Selman's notion of a "smart reduction".

1 Prologue

This paper consists of an orphan theorem.

The history of this work begins with a study of the complexity of time-bounded Kolmogorov complexity as it relates to the Graph Automorphism problem [AGM15]. The authors of [AGM15] continued to explore this topic with other collaborators after [AGM15] was posted on ECCC, and at one point it was found that the exposition could be simplified by proving that Graph Automorphism can be reduced to the Rigid Graph Isomorphism promise problem via a smart reduction, which is the topic of the current note. The project eventually developed into a significantly stronger paper [AGvM$^+$18]. (A more complete version of this work is available as [AGvM$^+$17].) But the proofs as presented in [AGvM$^+$18, AGvM$^+$17] no longer make any reference to smart reductions.

What to do?

The observation that Graph Automorphism reduces to Rigid Graph Isomorphism might be useful in some future situation, but this fact by itself falls somewhat short of the Least Publishable Unit threshold. However, it also is not compatible with inclusion in [AGvM$^+$17]. Since the proof was already written, and since the availability of this proof might save someone some effort in the future, it was decided to provide space in this ECCC comment (which also serves the purpose of pointing out that [AGM15] has been superseded by [AGvM$^+$17]) to archive this fact about smart reductions and Graph Automorphism.

*Rutgers University, Piscataway, NJ, USA, allender@cs.rutgers.edu
†University of Colorado at Boulder, Boulder, CO, USA, joshua.grochow@colorado.edu
‡University of Wisconsin–Madison, Madison, WI, USA, dieter@cs.wisc.edu
§Santa Fe Institute, Santa Fe, NM, USA, moore@santafe.edu
¶University of Wisconsin–Madison, Madison, WI, USA, amorgan@cs.wisc.edu

ISSN 1433-8092

2 Preliminaries

We assume that the reader is already familiar with the Graph Isomorphism problem (GI) and the Graph Automorphism problem GA; see [KST93] for more background. It is well-known that $\mathsf{GA} \leq_m^p \mathsf{GI}$ (i.e., GA Karp-reduces to GI) but the converse is not known.

A *promise problem* consists of a pair of disjoint subsets $Y, N \subseteq \Sigma^*$ where, as usual, Σ is a finite alphabet. A language B is a *solution* to the promise problem (Y, N) if $Y \subset B \subset \overline{N}$. (Note that a language L is simply the promise problem $(L, \overline{L})$.) An algorithm A *solves* a promise problem (Y, N) if $A(x) = 1$ for all $x \in Y$, and $A(x) = 0$ for all $x \in N$. Of particular interest to us is the promise problem known as the *Rigid Graph Isomorphism Problem*. A graph is *rigid* if it has no nontrivial automorphisms, i.e., none other than the identity. Rigid Graph Isomorphism (Rigid GI) is a promise version of GI: namely, to decide whether two graphs are isomorphic, given the promise that they are rigid. That is, Y is the set of pairs of rigid graphs (G, H) such that G and H are isomorphic, and N is the set of pairs of rigid graphs such that G and H are not isomorphic. Thus an algorithm that solves Rigid GI can have arbitrary output if one of its inputs is not rigid.

We will refer to the following "promise complexity classes".

- Promise-BPP is the class of all promise problems (Y, N) for which there is a probabilistic polynomial-time Turing machine M such that, for all $x \in Y$, M accepts x with probability at least $2/3$, and for all $x \in N$, M rejects x with probability at least $2/3$.
- Promise-RP is the class of all promise problems (Y, N) for which there is a probabilistic polynomial-time Turing machine M such that, for all $x \in Y$, M accepts x with probability at least $2/3$, and for all $x \in N$, M rejects x with probability 1.
- Promise-coRP is the class of all promise problems (Y, N) for which (N, Y) is in Promise-RP.
- Promise-ZPP is the class of all promise problems (Y, N) for which there is a probabilistic polynomial-time Turing machine M such that, for all $x \in Y$, M accepts x with probability at least $1/2$ and outputs the failure symbol $\perp$ otherwise, and for all $x \in N$, M rejects x with probability at least $1/2$ and outputs the failure symbol $\perp$ otherwise. Promise-ZPP = Promise-RP $\cap$ Promise-coRP.
- Promise-NP is the class of all promise problems (Y, N) for which there is a nondeterministic polynomial-time Turing machine M such that, for all $x \in Y$, some computation path of M accepts x, and for all $x \in N$, M has no computation path accepting x.
- Promise-coNP is the class of promise problems (Y, N) for which $(N, Y) \in$ Promise-NP.
- Promise-(NP $\cap$ coNP) is the class of all promise problems (Y, N) for which there is a nondeterministic polynomial-time Turing machine M such that, for all $x \in Y$, some computation path of M accepts x and any that do not accept x output $\perp$, and for all $x \in N$, some computation path of M rejects x while any that do not reject x output $\perp$. Promise-(NP $\cap$ coNP) = Promise-NP $\cap$ Promise-coNP.
- Promise-UP is the class of all promise problems (Y, N) for which there is a nondeterministic polynomial-time Turing machine M such that, for all $x \in Y$, M has exactly one accepting computation path on x, and for all $x \in N$, M has no accepting computation paths.

- Promise-coUP is the class of all promise problems (Y, N) for which (N, Y) is in Promise-UP.

- Promise-(UP $\cap$ coUP) is the class of all promise problems (Y, N) for which there is a nondeterministic polynomial-time Turing machine M such that, for all $x \in Y$, M has exactly one accepting computation path on x and all others output $\perp$, and for all $x \in N$, M has exactly one rejecting computation path on x and all others output $\perp$. Promise-(UP $\cap$ coUP) = Promise-UP $\cap$ Promise-coUP.

An important part of these definitions is that, on inputs outside of $Y \cup N$, the Turing machine M might not satisfy the "promise" (by having acceptance probability not bounded away from 1/2, or by having more than one accepting computation path).

The main topic of this note concerns "smart" reductions. In order to motivate this notion, let us first define what it means to reduce one promise problem to another. Let (Y', N') and (Y, N) be promise problems. We say that $(Y', N') \leq_T^p (Y, N)$ if there is a polynomial-time oracle machine M, such that for *every* solution B of (Y, N), the language accepted by $M^B(x)$ is a solution to (Y', N'). Note that, on any input $x \in Y' \cup N'$, the output of M with oracle B is the same as with any other oracle B' that agrees with B on $Y \cup N$; any query that is asked by M that lies outside of $Y \cup N$ can be answered arbitrarily, without affecting the final outcome, and thus in some sense it is not very "smart" for M to bother asking such "useless" queries. This motivated Grolman and Selman to formulate the following definition:

Definition 1. [GS88] A polynomial-time Turing reduction to a promise problem (Y, N) is called a *smart reduction* if it makes queries only to elements of $Y \cup N$.

Smart reductions pull back membership in promise complexity classes, as exemplified in the following fact:[1]

Proposition 1. *Let $\mathcal{C}$ be any of* Promise-BPP, Promise-ZPP, Promise-(NP $\cap$ coNP), *or* Promise-(UP $\cap$ coUP). *If (Y, N) has a smart reduction to a problem in $\mathcal{C}$, then $(Y, N) \in \mathcal{C}$.*

As Grollman and Selman observe in [GS88], smartness seems to be a significant restriction on the space of all possible reductions to promise problems. In general, reductions to promise problems do not seem to be able to avoid making "useless" queries where the promise does not hold, although such queries cannot affect the ultimate decision of whether to accept or reject. That is, reductions to promise problems can probably not always be smart. (For more on this topic, including applications to the Graph Isomorphism problem, see [CGRS04, GSS05].)

As a warm-up, we sketch the known smart reduction of search to decision for the promise problem of graph isomorphism on rigid graphs. (That is, there is a deterministic polynomial-time oracle machine that, when given two rigid graphs G_0 and G_1, will either determine that the graphs are not isomorphic, or else produce an isomorphism between the two graphs, making only oracle queries to the graph isomorphism problem where all of the queries consist of pairs of rigid graphs.) This is completely trivial if the graphs are not isomorphic; thus assume that the rigid graphs G_0 and G_1 are isomorphic. There is a (unique) vertex i such that vertex 1 of G_0 maps to vertex i of G_1 via an isomorphism. We can find i using the decision oracle as follows: For each $j \in [n]$, attach a rigid "label" r to vertex 1 of G_0

[1] We remark that for $\mathcal{C}$ = Promise-BPP, the fact also holds for non-smart reductions, but is a bit more cumbersome to prove.

(call this graph H), and attach the same label to vertex j in G_1 (call this graph H_j). Note that both H and H_j are rigid. Query the decision oracle for each pair (H, H_j), let i be the (unique) j for which the answer is positive, and set $\pi(1) = i$. We keep the label r, i.e., we continue with (H, H_i), and repeat the process to find $\pi(2)$, etc.

3 Main Theorem

It is shown in [KST93] that $\mathsf{GA} \leq_T^p \mathsf{Rigid\ GI}$, but the reduction given there is not a smart reduction. Theorem 1 gives a smart reduction.

Theorem 1. *There is a smart reduction reducing Graph Automorphism to the Rigid Graph Isomorphism Problem.*

Proof of Theorem 1. Our proof is patterned after the proof of [KST93, Theorem 1.31], which presents a reduction of Graph Automorphism to Rigid GI.

Let G be an n-vertex graph that is input to the Graph Automorphism problem. G has a non-trivial automorphism if and only if there is an automorphism that sends some vertex i to a vertex $j \neq i$. Any automorphism fixes some (possibly empty) set of vertices.

Using the notation of [KST93], let $G_{[j]}^{(i-1)}$ be the graph (easy to construct in polynomial time, as presented in [KST93, pages 8 and 31]) with distinct labels on vertices $\{1, \ldots, i-1\}$ (so that no automorphism can move any of those vertices), and a distinguishing label on vertex $j \geq i$. As a slight modification of this notation, let $G_{[j,k]}^{(i-1)}$ again have distinct labels on vertices $\{1, \ldots, i-1\}$ and with two new colors (i.e., labels) r and b (red and blue), with j colored r and k colored b, where $j \geq i$ and $k \geq i$.

Let i be the largest index for which some automorphism exists that fixes vertices $\{1, 2, \ldots, i-1\}$, and sends i to some $j > i$. Then for some $k > i$, $G_{[i,k]}^{(i-1)}$ and $G_{[j,i]}^{(i-1)}$ are isomorphic, and for all $j > i$ and $k > i$, $G_{[i,k]}^{(i-1)}$ and $G_{[j,i]}^{(i-1)}$ are rigid (since the first i vertices have distinct labels). Furthermore, for all $\ell > i, G_{[\ell,k]}^{(\ell-1)}$ and $G_{[j,\ell]}^{(\ell-1)}$ are rigid and non-isomorphic for every $j > \ell$ and $k > \ell$.

Thus if we start with $i = n-1$ and pose queries of the form $(G_{[i,j]}^{(i-1)}, G_{[k,i]}^{(i-1)})$ (for $j, k \in \{i+1, \ldots, n\}$) to an oracle for the Rigid Graph Isomorphism Problem, it holds that for all large values of i the graphs are rigid and non-isomorphic (and thus satisfy the promise of the promise problem (Y, N), until we encounter the first triple (i, j, k) such that the graphs $(G_{[i,j]}^{(i-1)}, G_{[k,i]}^{(i-1)})$ are isomorphic. These graphs are also rigid, and thus they also satisfy the promise. If the computation ends with all queries determined to be non-isomorphic, then this is a proof that G has no nontrivial automorphism.

This algorithm works correctly on all inputs, and all queries satisfy the promise. Thus it is a smart reduction. □

Theorem 1 entails the following complexity-theoretic consequences:

Corollary 1. *The following implications hold:*

1. *If* Rigid GI *is in* Promise-BPP*, then* $\mathsf{GA} \in \mathsf{RP}$.
2. *If* Rigid GI *is in* Promise-coRP*, then* $\mathsf{GA} \in \mathsf{ZPP}$.
3. *If* Rigid GI *is in* Promise-coNP*, then* $\mathsf{GA} \in \mathsf{NP} \cap \mathsf{coNP}$.

4. If Rigid GI *is in* Promise-coUP*, then* GA $\in$ UP $\cap$ coUP.

Each of these implications follows by combining known search-to-decision reductions and closure properties of the relevant complexity classes with Theorem 1. The argument is captured by the following claim:

Claim 1. *Let* $\mathcal{C}$ *be any of* BPP*,* RP*,* NP*, or* UP*. If* Rigid GI $\in$ promise-co$\mathcal{C}$*, then for any language* L*, if there is a smart reduction* $L \leq_T^p$ Rigid GI*, then* $L \in \mathcal{C} \cap$ co$\mathcal{C}$*.*

The claim makes use of the following fact regarding search-to-decision reductions (which easily follows from essentially the same proof as that of [Ko82], showing that if SAT is in BPP, then it is in RP.):

Proposition 2. *Let* (Y, N) *be a promise problem for which there is a smart reduction from search to decision. Then* $(Y, N) \in$ Promise-BPP *implies* $(Y, N) \in$ Promise-RP*.*

Given Claim 1, implications 2–4 of Corollary 1 follow immediately. For implication 1, the claim implies that, under the hypothesis of implication 1, GA $\in$ BPP. GA has a search-to-decision reduction, so we may moreover conclude by Proposition 2 that GA $\in$ RP. (We may drop the promise- qualifier from classes containing GA, since GA is a language.)

It remains to prove Claim 1:

Proof of Claim 1. For the case of $\mathcal{C} =$ BPP, we have co$\mathcal{C} =$ BPP $= \mathcal{C}$, so we just need to show that Rigid GI $\in$ Promise-BPP and $L \leq_T^p$ Rigid GI implies L in BPP. This follows from Proposition 1. (We may drop the promise- qualifier from classes containing L, since L is a language.)

When $\mathcal{C} =$ RP, we recall that Rigid GI has a search-to-decision reduction, so the hypothesis combined with Proposition 2 implies that Rigid GI $\in$ Promise-ZPP. The claim now follows from Proposition 1.

For the cases where $\mathcal{C} =$ NP or $\mathcal{C} =$ UP, we note that Rigid GI is (unconditionally, essentially by definition) in promise-$\mathcal{C}$. So if Rigid GI is in promise-co$\mathcal{C}$, it is in promise-$\mathcal{C} \cap$ promise-co$\mathcal{C}$ = promise-($\mathcal{C} \cap$ co$\mathcal{C}$). The claim now follows from Proposition 1. □

We observe that we do *not* know how to prove the implication "if Rigid GI is in Promise-UP and L smartly reduces to Rigid GI, then $L \in$ UP." This is especially disappointing, since the hypothesis "Rigid GI is in Promise-UP" follows easily from the definitions.

Since the "promise" in the Rigid GI promise problem is precisely the problem solved by GA, hypotheses regarding the complexity of the promise problem Rigid GI yield conclusions about rigid graph isomorphism that do not need to be phrased in terms of promise problems:

Corollary 2. *Let* $A = \{(G, H) : G$ *and* H *are rigid, and* G *is isomorphic to* $H\}$*, and* $B = \{(G, H) : G$ *and* H *are rigid, and* G *is not isomorphic to* $H\}$*.*

- *If* Rigid GI *is in* Promise-BPP*, then* A *and* B *are in* BPP*.*
- *If* Rigid GI *is in* Promise-coRP*, then* A *and* B *are in* ZPP*.*
- *If* Rigid GI *is in* Promise-coNP*, then* A *and* B *are in* NP $\cap$ coNP*.*
- *If* Rigid GI *is in* Promise-coUP*, then* A *and* B *are in* UP $\cap$ coUP*.*

Proof. Let $(Y_1, N_1) = \mathsf{Rigid\ GI}$ and $(Y_2, N_2) = \mathsf{GA}$. Define $\mathsf{Rigid\ GI} \sqcup \mathsf{GA} \doteq (Y, N)$ where

$$Y = (\{1\} \times Y_1) \cup (\{2\} \times Y_2)$$
$$N = (\{1\} \times N_1) \cup (\{2\} \times N_2)$$

Both A and B have smart reductions to $\mathsf{Rigid\ GI} \sqcup \mathsf{GA}$: check rigidity by queries to GA, and then, if the graphs are rigid, use $\mathsf{Rigid\ GI}$ to determine isomorphism. Moreover, Theorem 1 implies $\mathsf{Rigid\ GI} \sqcup \mathsf{GA}$ has a smart reduction to $\mathsf{Rigid\ GI}$, so A and B both smartly reduce to $\mathsf{Rigid\ GI}$. The implications now follow from Claim 1. □

Acknowledgments. E. A. acknowledges the support of National Science Foundation grant CCF-1555409. J. A. G. was supported by an Omidyar Fellowship from the Santa Fe Institute and National Science Foundation grant DMS-1620484. D. v. M. and A. M. acknowledge the support of National Science Foundation grant CCF-1319822. We thank V. Arvind for helpful comments about the graph automorphism problem and rigid graphs.

References

[AGM15] Eric Allender, Joshua Grochow, and Cristopher Moore. Graph isomorphism and circuit size. Technical Report TR15-162, Electronic Colloquium on Computational Complexity, 2015.

[AGvM+17] Eric Allender, Joshua A. Grochow, Dieter van Melkebeek, Cristopher Moore, and Andrew Morgan. Minimum circuit size, graph isomorphism, and related problems. Technical Report TR17-158, Electronic Colloquium on Computational Complexity, 2017.

[AGvM+18] Eric Allender, Joshua A. Grochow, Dieter van Melkebeek, Cristopher Moore, and Andrew Morgan. Minimum circuit size, graph isomorphism, and related problems. In *9th Innovations in Theoretical Computer Science Conference, ITCS*, volume 94 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 20:1–20:20, 2018.

[CGRS04] Marcel Crâsmaru, Christian Glaßer, Kenneth W. Regan, and Samik Sengupta. A protocol for serializing unique strategies. In *Mathematical Foundations of Computer Science (MFCS)*, volume 3153 of *Lecture Notes in Computer Science*, pages 660–672. Springer, 2004.

[GS88] Joachim Grollmann and Alan L. Selman. Complexity measures for public-key cryptosystems. *SIAM J. Comput.*, 17(2):309–335, 1988.

[GSS05] Christian Glaßer, Alan L. Selman, and Samik Sengupta. Reductions between disjoint NP-pairs. *Information and Computation*, 200(2):247–267, 2005.

[Ko82] Ker-I Ko. Some observations on the probabilistic algorithms and NP-hard problems. *Information Processing Letters*, 14(1):39–43, 1982.

[KST93] Johannes Köbler, Uwe Schöning, and Jacobo Torán. *The Graph Isomorphism Problem: Its Structural Complexity*. Birkhauser Verlag, Basel, Switzerland, Switzerland, 1993.

ECCC ISSN 1433-8092

https://eccc.weizmann.ac.il