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

Graph Isomorphism and Circuit Size

Eric Allender[*], Joshua A. Grochow[†], and Cristopher Moore[‡]

October 5, 2015

Abstract

We show that the Graph Automorphism problem is ZPP-reducible to MKTP, the problem of minimizing time-bounded Kolmogorov complexity. MKTP has previously been studied in connection with the Minimum Circuit Size Problem (MCSP) and is often viewed as essentially a different encoding of MCSP. All prior reductions to MCSP have applied equally well to MKTP, and vice-versa, and all such reductions have relied on the fact that functions computable in polynomial time can be inverted with high probability relative to MCSP and MKTP. Our reduction uses a different approach, and consequently yields the first example of a problem in $\mathsf{ZPP}^{\mathsf{MKTP}}$ that is not known to lie in $\mathsf{NP} \cap \mathsf{coNP}$. We also show that this approach can be used to provide a reduction of the Graph Isomorphism problem to MKTP.

1 Introduction

Graph Isomorphism (GI)—determining if two given graphs are isomorphic—and the Minimum Circuit-Size Problem (MCSP)—determining if a given truth table can be computed by a circuit of a given size—are not only natural and interesting computational problems in their own right, but are some of the best-known examples of potentially "NP-intermediate" problems. In fact, they are among the problems that motivated many of the researchers originally studying NP and NP-completeness (see, e. g., [AD14, AKRR10, Tra84] for some of the history). Both of these problems are well-studied [KC00, MW15, AD14, AHK15, ABK$^+$06], and have applications both in theory (e.g., [KST93] and references therein) and outside of computer science (see [Irn05, p. 3] for an overview and further references). Despite the fact that interest in these problems goes back to the foundations of computational complexity [Tra84], only very recently was any direct relationship realized between them: namely, $\mathsf{GI} \in \mathsf{RP}^{\mathsf{MCSP}}$ [AD14].

We give a new, direct reduction from GI to a variant of MCSP called MKTP: the problem of minimizing a form of time-bounded Kolmogorov complexity. More specifically, we show that Graph Automorphism (GA) is in $\mathsf{ZPP}^{\mathsf{MKTP}}$, giving the first ZPP-style reduction from a problem closely related to GI to MKTP, as asked for in [AD14]. As a consequence, Graph Isomorphism is in $\mathsf{BPP}^{\mathsf{MKTP}}$; this is weaker than the inclusion $\mathsf{GI} \in \mathsf{RP}^{\mathsf{MKTP}}$ shown in [AD14], but the reduction is quite different than earlier reductions to MKTP.

Our understanding of the complexity of MCSP has been crippled by the fact that—until now—*all* reductions of supposedly-intractable problems to MCSP have proceeded along the

[*]Department of Computer Science, Rutgers University, Piscataway, NJ, USA, `allender@cs.rutgers.edu`
[†]Santa Fe Institute, Santa Fe, NM, USA, `jgrochow@santafe.edu`
[‡]Santa Fe Institute, Santa Fe, NM, USA, `moore@santafe.edu`

ISSN 1433-8092

same well-trodden path. Namely, MCSP is used as an efficient statistical test to distinguish random distributions from pseudorandom distributions. Thus, as in [HILL99], MCSP can be used in a subroutine to invert a polynomial-time-computable function f on a significant portion of its range. Note that any ZPP-reduction that relies on finding elements in $f^{-1}(z)$ for various strings z necessarily results in an $\mathsf{NP} \cap \mathsf{coNP}$ upper bound for the problem being reduced to MCSP, since a zero-error probabilistic algorithm can never receive assurance that z is *not* in the range of f; the only reliable answers that it can receive from an inversion subroutine take the form of strings x such that $f(x) = z$. If one then simulates the ZPP reduction and merely guesses such strings x, this yields a $\mathsf{NP} \cap \mathsf{coNP}$ algorithm. Thus this "traditional" approach cannot be used in order to ZPP-reduce GA to MCSP without first putting GA in coNP.

One of the main contributions of this work is that we present a fundamentally different way to use problems like MCSP as an oracle to solve natural problems. We say "problems *like* MCSP", because our techniques do not seem to apply directly to MCSP itself. Rather, we make use of a related problem, called MKTP: for a given string, decide if its time-bounded Kolmogorov complexity (specifically, a measure KT defined in [ABK+06]) is at most a given number. Because of the close connection between KT and circuit size, MKTP and MCSP have often been studied in tandem [AD14, AHK15, ABK+06], and MKTP is often regarded as a mere reformulation of MCSP. Indeed, in all cases except ours, theorems that apply to MCSP have applied immediately to MKTP and vice versa, even though no efficient reduction is known in either direction between MCSP and MKTP. Although we believe that GA is in $\mathsf{ZPP}^{\mathsf{MCSP}}$, our proof does not carry over to MCSP; we discuss the obstacles in Section 6.

2 Preliminaries

2.1 Promise problems and their complexity classes

A *promise problem* consists of a pair of disjoint subsets $Y, N \subseteq \Sigma^*$ where, as usual, Σ is a finite alphabet. A Turing machine M solves a promise problem if M accepts all instances in Y and rejects all instances in N; M can have arbitrary output for instances that are neither in Y nor N. A promise problem (Y, N) is in $\mathsf{Promise\text{-}ZPP}$ if there is a probabilistic Turing machine M such that M solves the promise problem (Y, N), and furthermore on all inputs in $Y \cup N$, M runs in expected polynomial time, where the expectation is taken over the coin flips of M; on inputs outside of $Y \cup N$, M need not run in expected polynomial time. In contrast, we say that a promise problem (Y, N) "has a solution in ZPP", if there is a probabilistic Turing machine M that solves (Y, N) and runs in expected polynomial time on *all* inputs. These definitions relativize in the natural way; our result for rigid graphs uses $\mathsf{Promise\text{-}ZPP}^{\mathsf{MKTP}}$. As with their non-promise variants, $\mathsf{Promise\text{-}ZPP} = \mathsf{Promise\text{-}RP} \cap \mathsf{Promise\text{-}coRP}$, and this result relativizes.

2.2 Graphs, automorphism groups, and isomorphism-style problems

A *graph* for us is a simple, undirected graph (though our results are easily adapted to most standard variants of graphs), that is, a vertex set $V(G)$, and a set $E(G)$ of unordered pairs of vertices. An *isomorphism* between two graphs G, H is a bijection $\pi \colon V(G) \to V(H)$ that preserves both edges and non-edges: $(v, w) \in E(G)$ if and only if $(\pi(v), \pi(w)) \in E(H)$. An isomorphism from a graph to itself is an *automorphism*; the set of all automorphisms of a given graph G form a group under composition, denoted $\mathrm{Aut}(G)$. A graph is *rigid* if it has no nontrivial automorphisms, i.e., none other than the identity.

Graph Isomorphism (GI) is the computational problem of deciding whether two graphs, given as input, are isomorphic. Rigid Graph Isomorphism ($\mathsf{Rigid\ GI}$) is a promise version of GI: namely, to decide whether two graphs are isomorphic, given the promise that they are rigid. Thus an algorithm that solves $\mathsf{Rigid\ GI}$ can have arbitrary output if one of its inputs is not rigid. Graph Automorphism (GA) is the problem of deciding whether a graph given as input has any nontrivial automorphisms, that is, whether it is *not* rigid. It is well-known that $\mathsf{GA} \leq_m^p \mathsf{GI}$ but the converse is not known.

By the Orbit–Stabilizer Theorem, there is a natural bijection between the set of (labeled) graphs isomorphic to a given n-vertex graph G, and the set of cosets of $\mathrm{Aut}(G)$ in S_n. Thus there are $n!/|\mathrm{Aut}(G)|$ such graphs.

A *permutation group* is a subgroup of the symmetric group S_n for some n. For us, permutation groups will always be given by a set of generating permutations, i.e., given a list of permutations $\pi_1, \ldots, \pi_k \in S_n$, we write $\Gamma = \langle \pi_1, \ldots, \pi_k \rangle \leq S_n$ for the subgroup they generate. (We use "$\leq$" between groups to denote "is a subgroup of"). Given a permutation group $\Gamma \leq S_n$ and a point $i \in [n]$, the Γ-orbit of i is $\Gamma i = \{g(i) : g \in \Gamma\}$ and the Γ-stabilizer of i is the subgroup $\mathrm{Stab}_\Gamma(i) = \{g \in \Gamma : g(i) = i\} \leq \Gamma$.

2.3 Circuit size and Kolmogorov (KT) complexity

We provide only a very brief introduction to Kolmogorov complexity here. The reader should refer to some of the excellent in-depth treatments of the subject (such as [LV08, DH10]) or to some of the informal introductions to the topic (such as [For01]) if more background is required.

The (plain) Kolmogorov complexity of a string x relative to a Turing machine U is denoted $C_U(x)$, and it is defined as the length of the shortest "description" d such that $U(d) = x$. We pick one so-called "universal" machine U: then for every machine U' there is a constant $c_{U'}$ such that, for all x,

$$C_U(x) \leq C_{U'}(x) + c_{U'} .$$

The (plain) Kolmogorov complexity of x is denoted $C(x)$, and it is defined to be $C_U(x)$ for this choice of universal Turing machine.

The Kolmogorov complexity is not computable, precisely because it is defined without regard to computational resources. There have been many definitions of different types of time-bounded Kolmogorov complexity, which require that x be obtained "efficiently" from the description d. The particular version of time-bounded Kolmogorov complexity that will concern us here is the KT measure defined and studied in [ABK$^+$06], where connections were drawn between KT and MCSP.

Definition 1 (KT). *Let U be a Turing machine. For each string x, define* $\mathrm{KT}_U(x)$ *to be*

$$\mathrm{KT}_U(x) = \min\{\, |d| + t : \quad \forall b \in \{0,1,*\}\ \forall i \leq |x|+1, \textit{ the machine } U^d(i,b) \textit{ accepts in } t \textit{ steps iff } x_i = b \,\}.$$

We define $x_i = *$ if $i > |x|$; thus, for $i = |x|+1$ the machine accepts iff $b = *$. The notation U^d indicates that the machine U has "oracle access" to the description d, which essentially means that the Turing machine has random access to the bits of d.

As for the plain Kolmogorov complexity, $\mathrm{KT}(x)$ is defined to be equal to $\mathrm{KT}_U(x)$ for a fixed choice of universal Turing machine U. However, due to the overhead with which Turing machines simulate each other, $\mathrm{KT}(x)$ and $\mathrm{KT}_U(x)$ generally differ by more than an additive constant: we only are able to guarantee that for every machine U' there exists a constant $c_{U'}$ such that for every x, $\mathrm{KT}(x) \leq c_{U'}\mathrm{KT}_{U'}(x)\log \mathrm{KT}_{U'}(x)$.

We refer the reader to [ABK$^+$06] for more about the history of other approaches to defining time-bounded Kolmogorov complexity, and the considerations that motivate this particular definition. One of these considerations is the connection between KT and circuit size. It is noted in [ABK$^+$06] that if x is a string of length m representing the truth table of a function f with minimum circuit size s, then it holds that

$$\left(\frac{s}{\log m}\right)^{1/4} \leq \mathrm{KT}(x) \leq O(s^2(\log s + \log\log m)).$$

We can now present the definition of MKTP:

Definition 2. $\mathsf{MKTP} = \{(x,k) \mid \mathrm{KT}(x) \leq k\}$.

For completeness, we also give a more complete definition of the minimum circuit size problem MCSP:

Definition 3. *If f is a string of length 2^m encoding the entire truth-table of some Boolean function of m variables, then* $\mathsf{MCSP} = \{(f,s) \mid f$ *has circuits of size* $\leq s\}$.

We decline to specify precisely the type of circuits that we consider, as well as the precise size measure—such as counting the number of wires or the number of gates, etc. Theorems that are proved using one variant usually carry over to any other reasonable variant, even though it is not known whether these variants are efficiently reducible to one another.

3 The result for rigid graphs

Define the *Rigid Graph Nonisomorphism Promise Problem* to be the promise problem defined by the set Y of "yes-instances" and the set N of "no-instances":

$Y = \{(G,H) : G$ and H are rigid, and G is not isomorphic to $H\}$.

$N = \{(G,H) : G$ is isomorphic to $H\}$.

It might seem more natural to restrict the set N to only consist of pairs of rigid matrices, but our reduction to MKTP actually works for the larger set, with the same proof. This extra generality allows us to apply the following result to Graph Automorphism (Corollary 1).

Theorem 1. *The Rigid Graph Nonisomorphism Promise Problem is in* $\mathsf{Promise\text{-}ZPP}^{\mathsf{MKTP}}$.

Proof outline and intuition. It is shown in [AD14] that Graph Isomorphism is in $\mathsf{RP}^{\mathsf{MCSP}}$. The same proof suffices to show that Graph Isomorphism is in $\mathsf{RP}^{\mathsf{MKTP}}$; in particular this shows that the Rigid Graph Nonisomorphism Problem has a solution in $\mathsf{coRP}^{\mathsf{MKTP}}$. Thus it suffices to show that the Rigid Graph Nonisomorphism Problem is also in $\mathsf{Promise\text{-}RP}^{\mathsf{MKTP}}$.

That is, we need to present a probabilistic oracle Turing machine M with an oracle for MKTP, so that, M accepts with high probability any pair $(G,H) \in Y$, and M *always* rejects any pair $(G,H) \in N$.

Our reduction is easy to present. The probabilistic machine M, given two n-vertex graphs G_0 and G_1, will flip $t = 3n^5$ coins, obtaining the bit string $w = w_1 w_2 \cdots w_t$. It then chooses t random permutations $\pi_1, \pi_2, \ldots, \pi_t$, and produces the string $x = \pi_1(G_{w_1})\pi_2(G_{w_2})\ldots\pi_t(G_{w_t})$. M will append some extra 0's or 1's—whichever is different from the last bit of x—to obtain a string x' of length 2^m for some m, which can thus be viewed as the truth table of some m-variate Boolean function. Next M computes a threshold θ (specified below), and accepts iff the string $(x', \theta) \notin$ MKTP, i.e., iff $\mathrm{KT}(x') > \theta$.

The intuition for why this reduction should work is also easy to present. If G_0 and G_1 are rigid and are not isomorphic, then the string x contains all of the information about the random string w and the t random permutations $\pi_1, \ldots, \pi_t$. Letting $s = \log n!$, we thus have (with high probability over the choice of random bits) that w contains about $t + st$ bits of information. On the other hand, if $G_0 \equiv G_1$, then w can be described by G_0 and the t permutations, which should be about $n^2 + ts$ bits of information, which with some luck should correspond to KT-complexity around $n^2 + ts$. Since we have chosen t to be sufficiently large, it will follow that $n^2 + ts$ will be much less than $t(s+1)$, and we can select the threshold θ to be $t(s+1) - 4n \log n$ so that it separates the two cases. $\square$

Proof details. The proof begins as above; we now provide the details needed to fill in the preceding informal argument.

First, we consider the case when G_0 and G_1 are both rigid, and are not isomorphic. Let $C(z)$ denote the Kolmogorov complexity of the string z. We claim that $C(w, \pi_1, \pi_2, \ldots, \pi_t) \leq C(x) + 2n \log n + 2 \log n$. This is because, given a description d of x (having length $C(x)$) and given the encodings of two permutations π_1^{-1}, ρ (having length at most $2n \log n$) and with an additional $\log n + 2 \log \log n + O(1) < 2 \log n$ bits (to describe the boundaries between these information fields and how to use them), one can construct the tuple $(w, \pi_1, \ldots, \pi_t)$ as follows:

1. Use d to construct x, which is a string of the form $H_1 H_2 \ldots H_t$ for some sequence of n-vertex graphs $H_1, \ldots, H_t$.
2. Apply π_1^{-1} to H_1 to obtain one of $\{G_0, G_1\}$. (Assume for the moment that $H_1 = \pi_1(G_0)$; equivalently, assume for the moment that the first bit of w is 0. The other case is handled similarly.)
3. Find the first $i > 1$ such that H_i is not isomorphic to G_0; apply ρ to H_i to obtain G_1.
4. For each j, find the unique $i_j \in \{0, 1\}$ such that H_j is isomorphic to G_{i_j}, and obtain the unique permutation π_j such that $\pi_j(G_{i_j}) = H_j$. (The permutation is unique because both G_0 and G_1 are rigid.)
5. The j-th bit of w is equal to i_j.

The machine M chooses the elements $(w, \pi_1, \pi_2, \ldots, \pi_t)$ uniformly at random from a set of size $2^t(n!)^t$, and thus with probability greater than 99%, $C(w, \pi_1, \pi_2, \ldots, \pi_t) \geq \log 2^t(n!)^t - 6 = t + ts - 6$ (recall that $s = \log(n!)$). Hence $C(x) \geq t + ts - 6 - 2n \log n - 2 \log n$. Observe that $C(x') = C(x) \pm O(1)$. Hence $\mathrm{KT}(x') \geq C(x') \geq t(s+1) - 3n \log n > \theta$.

Suppose, conversely, that G_0 and G_1 are isomorphic. Let d be a description containing all of the information about G_0 and the permutations $\pi_1, \pi_2, \ldots, \pi_t$. Since we need to be careful about the length of this description, it will be necessary to give some details about the way this is encoded. Let $t = bt'$; we will divide the sequence $\pi_1, \pi_2, \ldots, \pi_t$ into t' "blocks" consisting of b permutations each.

A block of b permutations will be encoded (in binary) as a number B between zero and $(n!)^b - 1$. To obtain the j-th permutation in a block, first obtain the j-th digit of B

in $(n!)$-ary notation; this digit is a number between 0 and $(n! - 1)$, and as such can be efficiently decoded as a representation of a permutation π using the well-known "Lehmer code". Each block requires $\lceil \log(n!)^b \rceil \leq bs + 1$ bits. Given a number j, and an encoding of a block B, a Turing machine can obtain an encoding of the permutation π in time easily bounded by $(bs)^3$.

The description d will have the form $(b, t', G_0, B_1 \ldots B_{t'})$, of length less than $2n^2 + t'(bs + 1) = 2n^2 + ts + t'$.

In order to obtain a bound on $\mathrm{KT}(x')$, we need to bound not only the length of d, but also the time required by a Turing machine that takes as input the number i (in binary), and computes the i-th bit of x', given random access to the description d.

Our machine will first read the initial bits of d, to determine the numbers b and t'. This enables us to compute t, which in turn determines n. All of this takes time at most $O(\log^2 n)$. (At this time, we can also determine if $i > |x|$, and we can compute $|x'|$. The discussion below deals with the most interesting case, when $i \leq |x|$. If $i > |x|$, then the machine should compute the last bit of x, and return the complement of that bit as the i-th bit of x'. If $i > |x'|$, then the machine should return $*$.)

Given the numbers i and n, it is easy to compute the numbers j and ℓ such that the i-th bit of x' corresponds to the ℓ-th bit of the adjacency matrix of $\pi_j(G_0)$. It is also easy to compute the numbers k_1 and k_2 such that π_j is encoded as permutation number k_1 in block B_{k_2} of d. Again, this computation takes time at most $O(\log^2 n)$.

With random access to the input, we can obtain the encoding of block k_2 using time $O(bs + 1)$, and then an additional $(bs)^3$ steps suffice to obtain the encoding of π_j. The i-th bit of x' corresponds to the ℓ-th bit of the adjacency matrix of $\pi_j(G_0)$. Time $O(n^3)$ is sufficient to determine the entry of matrix G_0 that corresponds to the ℓ-th bit of $\pi_j(G_0)$, and to query that bit of G_0, which determines the i-th bit of x'.

Thus the entire computation takes time bounded by $O(\log^2 n) + (bs)^3 + O(n^3)$. $\mathrm{KT}(x')$ is the sum of this running time and the length of d. Thus $\mathrm{KT}(x') \leq 2n^2 + ts + t' + O(\log^2 n) + (bs)^3 + O(n^3) < ts + O(n^3) + t/b + b^3(n \log n)^3$.

Picking the block size $b = 3$, and recalling that $t = 3n^5$, we thus have, for all large inputs, $\mathrm{KT}(x') \leq ts + t/3 + n^4 < \theta = ts + t - 4n \log n$. $\square$

Corollary 1. *Graph Automorphism is in* $\mathsf{ZPP}^{\mathsf{MKTP}}$.

A related result (reducing Graph Automorphism to a somewhat different promise problem) is presented as Theorem 1.31 in [KST94]. Our proof is patterned after the proof presented there.

Proof. 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 [KST94], let $G^{(i-1)}_{[j]}$ be the graph (easy to construct in polynomial time, as presented in [KST94, 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.

Let i be the largest index for which some automorphism exists that fixes vertices $\{1, 2, \ldots, i-1\}$, and sends i to some $j \neq i$. Then $G^{(i-1)}_{[i]}$ and $G^{(i-1)}_{[j]}$ are isomorphic, and for all $k > i, G^{(k-1)}_{[\ell]}$ and $G^{(k-1)}_{[m]}$ are rigid and non-isomorphic for every $\ell \neq m$.

Thus if we use our Promise-$\mathsf{ZPP}^{\mathsf{MKTP}}$ algorithm for the Rigid Graph Nonisomorphism Problem on queries of the form $(G^{(i-1)}_{[i]}, G^{(i-1)}_{[j]})$ (for $j \in \{i+1, \dots, n\}$) we get reliable "non-isomorphic" answers for all large values of i, (starting with $i = n-1$), since all of those graphs are rigid and non-isomorphic, until we encounter the first pair (i, j) such that the graphs $(G^{(i-1)}_{[i]}, G^{(i-1)}_{[j]})$ are isomorphic, in which case we also get a reliable answer, because in the definition of Rigid Graph Nonisomorphism we had $N = \{(G, H) | G \cong H\}$, with no restriction on their rigidity. (If all queries are determined to be non-isomorphic, then this is a proof that G has no nontrivial automorphism.)

This algorithm works correctly on all inputs, and thus it is a $\mathsf{ZPP}^{\mathsf{MKTP}}$ algorithm; there is no need to invoke any promise (of rigidity or otherwise). □

Similarly, the "promise" notation can be removed from a weaker version of Theorem 1, as follows. By the "Graph Isomorphism Problem restricted to rigid graphs", we mean the promise problem with the following Yes and No instances:

$Y = \{(G, H) : G$ and H are rigid, and G is isomorphic to $H\}$.

$N = \{(G, H) : G$ and H are rigid, and G is not isomorphic to $H\}$.

Corollary 2. *The Graph Isomorphism Problem restricted to rigid graphs has a solution in* $\mathsf{ZPP}^{\mathsf{MKTP}}$.

Proof. We need to present a $\mathsf{ZPP}^{\mathsf{MKTP}}$ algorithm that has ZPP-like behavior on *all* inputs—not merely the inputs that lie in Y or N. Thus, consider an input (G, H) that lies outside of $Y \cup N$. This means that at least one of $\{G, H\}$ has a non-trivial automorphism. But testing if one of $\{G, H\}$ has a non-trivial automorphism can be determined in $\mathsf{ZPP}^{\mathsf{MKTP}}$, by the preceding corollary.

Thus the algorithm is as follows: On input (G, H) determine (in $\mathsf{ZPP}^{\mathsf{MKTP}}$) whether G and H are rigid.

If both are rigid, run the Promise-$\mathsf{ZPP}^{\mathsf{MKTP}}$ algorithm to determine whether they are isomorphic or not. The promise is satisfied, so the probability of success is guaranteed to be high

Otherwise, reject. Note that, on all inputs, we terminate with high probability, with either a proof of non-rigidity, or a proof of isomorphism or non-isomorphism. □

4 The result for general graphs

Theorem 2. *Graph Isomorphism is in* $\mathsf{BPP}^{\mathsf{MKTP}}$.

Proof outline and intuition. The basic idea of the proof is similar to that of Theorem 1, but we now must handle nontrivial automorphisms of the graphs. The reduction is in fact more or less the same. To recall: Given two graphs G_0, G_1 on n vertices, we choose t random bits $w_1, \dots, w_t$ and t random permutations $\pi_1, \dots, \pi_t \in S_n$ (for sufficiently large t to be chosen later, still bounded by a polynomial in n). We construct the string $x = \pi_1(G_{w_1})\pi_2(G_{w_2}) \cdots \pi_t(G_{w_t})$, and extend x trivially, as in Theorem 1, to x' that has length a power of 2. We would then like to choose an appropriate threshold θ (a different value from the rigid case), and accept iff $\mathrm{KT}(x') > \theta$; this latter part will turn out to not quite work, but can be fixed, as we explain below.

The intuition is similar to before: if G_0 and G_1 are not isomorphic, then the string x contains all the information about t. Now, however, it does not contain *all* the information about the permutations $\pi_1, \ldots, \pi_t$, but only that information not captured by the automorphism groups of the G_i. That is, x contains the information of which coset of $\mathrm{Aut}(G_i)$ the permutation π_i is in, for each $1 \leq i \leq t$. Let $s_i = \log(n!/\mathrm{Aut}(G_i))$, for $i = 0, 1$. With high probability, x thus contains about $t + s_1 t_1 + s_0 t_0$ bits of information, where t_i is the number of bits of w that are i. On the other hand, if G_0 and G_1 are isomorphic, then w can be described by the t cosets of $\mathrm{Aut}(G_0)$, along with some some extra information about G_0 of size n^4—or approximately $n^4 + ts_0$ bits. As before, by choosing t sufficiently large, $(s_0 + 1)t$ will be larger than $n^4 + ts_0$, so there is a threshold θ in between the two.

The first new idea here is used, in the isomorphic case, to describe x in a *time-efficient* manner using fewer than $n^4 + ts_0$ bits. We cannot naively describe the t permutations, because that would result in $n^2 + t\log(n!)$ bits, which could be significantly larger than $n^2 + t(\log(n!) - \log|\mathrm{Aut}(G_0)|)$, throwing off the whole argument. (Note, in both the preceding argument and here, how important it is for the coefficients of t in the upper and lower bounds to match *nearly exactly.*) This is achieved in Lemma 1 below.

The second new idea here is needed because the natural threshold θ seems to depend on s_0, but knowing s_0 is equivalent to knowing $|\mathrm{Aut}(G_0)|$, and computing the size of the automorphism group of a graph is equivalent to GI in the first place. We get around this by estimating the KT complexity in two different cases, as follows. For each of the following three strings, compute the KT complexity exactly (using binary search and the MKTP oracle):

1. A sequence of randomly permuted graphs consisting of both G_0 and G_1, where which graph is used is determined by the random string w of length t,

2. A sequence of t randomly permuted copies of G_0, and

3. A sequence of t randomly permuted copies of G_1.

Do this several times with independent randomness, and for each of (1)–(3), report the maximum seen as our estimate of the KT complexity of such a construction. If G_0 and G_1 are isomorphic, with high probability there will be only a difference of $O(1)$ between these three. If G_0 and G_1 are not isomorphic, the true difference in KT complexity is at least $t - n^4 - 3n\log n - 2\log n - 6 \geq t/2$ for t sufficiently large, and the estimated difference should only be $O(1)$ off from this, with high probability. Thus, if the difference is at least $t/2$ for appropriately chosen value of t, we report that the graphs are not isomorphic, and otherwise we report that they are isomorphic. □

The argument relies on the following technical lemma, which may be of independent interest. It essentially says that, given some side information about a graph G, there is a "compressed" description p_π of any isomorphic copy $\pi(G)$ of G, whose size is *exactly* the information-theoretic minimum $\log(n!/|\mathrm{Aut}(G)|)$, and from which $\pi(G)$ can be reconstructed efficiently.

Lemma 1. *There is a polynomial-time "decoder" algorithm D such that for every graph G on n vertices, there is an auxiliary string aux_G of length $O(n^3 \log n)$, such that for every permutation $\pi \in S_n$ there is a number $p_\pi \in \{1, \ldots, n!/|\mathrm{Aut}(G)|\}$ such that $D(aux_G, p_\pi) = \pi(G)$.*[1]

[1]The input p_π to D is given in binary; we only specify that $p_\pi \in [n!/|\mathrm{Aut}(G)|]$ to show how exactly this matches the information-theoretic lower bound, which is essentially necessary for its use in Theorem 2.

First, we make use of the technical lemma in order to turn our informal argument for Theorem 2 into a proof. We will prove the technical lemma in Section 5.

Detailed proof of Theorem 2. Recall the basic outline from above. Let $s = \log(n!)$, $a_i = |\mathrm{Aut}(G_i)|$, and $s_i = s - a_i = \log(n!/|\mathrm{Aut}(G_i)|)$.

First, we consider the case when G_0 and G_1 are not isomorphic. Choose a random string w of length t (specified below). For $b = 0, 1$, let t_b be the number of bits of w that are equal to b. We will give a lower bound on $C(x)$ (the Kolmogorov complexity of x), which is in turn a lower bound on $\mathrm{KT}(x')$. We claim that

$$C(w, \pi_1, \ldots, \pi_t) \leq C(x) + 2n \log n + 2 \log n,$$

where the π_i are each the lexicographically least element in their $\mathrm{Aut}(G_{w_i})$-coset. To establish the claim, we will show how to take as input a description d of x (having length $C(x)$), and permutations π_1^{-1} and ρ (having length at most $2n \log n$), along with an additional $\log(n) + 2\log\log(n) + O(1) < 2\log n$ bits (to describe the boundaries between these information fields and how to use them), and then construct $(w, \pi_1, \ldots, \pi_t)$. Note that this construction need not be efficient: since we are only bounding $C(x)$, it need only be computable.

1. Use d to construct x, which is a string of the form $H_1, \ldots, H_t$ for some sequence of n-vertex graphs $H_1, \ldots, H_t$.
2. Apply π_1^{-1} to H_1 to obtain one of $\{G_0, G_1\}$. Assume for the moment that $H_1 = \pi_1(G_0)$, that is, that $w_1 = 0$. The other case is handled similarly.
3. Find the first $i > 1$ such that H_i is not isomorphic to G_0. Apply ρ to H_i to obtain G_1.
4. For each j in order, find the unique $b_j \in \{0, 1\}$ such that H_j is isomorphic to G_{b_j}, and find the lexicographically least permutation π_j such that $\pi_j(G_{b_j}) = H_j$.
5. The j-th bit of w is equal to b_j.

We have chosen w uniformly at random from a set of size 2^t, and then the sequence $(\pi_i : w_i = 0)$ is chosen uniformly at random from a set of size $\left(\frac{n!}{|\mathrm{Aut}(G_0)|}\right)^{t_0}$ and similarly the remaining π_i uniformly at random from a set of size $\left(\frac{n!}{|\mathrm{Aut}(G_1)|}\right)^{t_1}$. Suppose $s_0 \leq s_1$. (The other case is dealt with below.) Then with probability greater than 99%, $C(w, \pi_1, \ldots, \pi_t) \geq t + t_1 s_1 + t_0 s_0 - 6 \geq t + t_1 s_0 + t_0 s_0 - 6 = t(1 + s_0)$. Hence

$$C(x) \quad \geq \quad t(1 + s_0) - 6 - 2n \log n - 2 \log n$$

Conversely, if G_0 and G_1 are isomorphic, then we can use Lemma 1, together with the "blocking" trick for encoding a sequence of numbers in $[n!/|\mathrm{Aut}(G_0)|]$ from the second half of the proof of Theorem 1 to get a description of x of length at most $|aux_{G_0}| + ts_0 + t/b \leq n^4 + t(s_0 + 1/b)$ (using block size $b = O(1)$), from which x can be constructed in polynomial time, say time $\leq n^c$ for some $c \geq 4$. Thus $\mathrm{KT}(x) \leq 2n^c + t(s_0 + 1/b)$.

We choose t to be n^{c+1}. Recall this part of the informal discussion of our reduction: For each of the following three strings, we compute the KT complexity exactly:

1. A sequence of randomly permuted graphs consisting of both G_0 and G_1, where which graph is used is determined by the random string w of length t,

2. A sequence of t randomly permuted copies of G_0, and

3. A sequence of t randomly permuted copies of G_1.

Do this several times with independent randomness, and for each of (1)–(3), report the maximum seen as our estimate of the KT complexity of such a construction.

If G_0 and G_1 are isomorphic, with high probability there will be only a difference of $O(1)$ between these three.

If G_0 and G_1 are not isomorphic, then in the case $s_0 \leq s_1$, the analysis above shows that (with high probability) the difference between the first and the second value is at least $t(1+s_0) - 6 - 2n\log n - 2\log n - 2n^c - t(s_0 + 1/b) = t(1-\frac{1}{b}) - O(n^c) \geq n$ for all large n. (If $s_0 \geq s_1$, then a large difference will be noticeable between the first and third value.) $\square$

The last step—needed because we cannot choose the threshold without knowing $|\mathrm{Aut}(G_0)|$—is the only source of two-sided error in this reduction.[2]

5 Main Technical Lemma

This section is devoted to a proof of Lemma 1. The intuitive idea of the proof should be made clear from the following examples; making it precise unfortunately seems to require notation that makes the proof more unwieldy than we would like, so we start with the examples.

Example 1 (Rigid graphs)**.** *For rigid graphs, there is no choice but to describe $\pi(i)$ for every individual vertex i, which is the same as describing π itself, so p_π is a number between 1 and $n! = n!/|\mathrm{Aut}(G)|$.*

Example 2 (Cycles)**.** *The automorphism group of an n-cycle is the dihedral group of order $2n$, so we want to describe an isomorphic copy of the cycle using a number in $[(n-1)!/2]$. Suppose our initial graph G is the n-cycle with vertex set $[n]$ and edges $(i, i+1)$ for all $i = 1, \dots, n-1$ and the edge $(n, 1)$. Let π be any permutation. Without loss of generality, we may assume that $\pi(1) = 1$: There is always an automorphism of G that sends any vertex to any other vertex, and in particular there is some $\rho \in \mathrm{Aut}(G)$ that sends 1 to $\pi^{-1}(1)$, and taking $\pi' = \pi \circ \rho$ we have $\pi'(G) = \pi(\rho(G)) = \pi(G)$, since ρ is an automorphism, and $\pi'(1) = \pi(\rho(1)) = \pi(\pi^{-1}(1)) = 1$; replace π by π'. To describe $\pi(G)$, we then need to know where the neighbors of 1, namely 2 and n, get mapped to. But because there is an automorphism that swaps 2 and n and fixes 1, we don't need to know their individual images (which would take a number between 1 and $2\binom{n-1}{2}$ to describe), but rather just their image as a set, which only requires a number in $[\binom{n-1}{2}]$ to describe. Next, as before, by*

[2] One can get a ZPP reduction from a different promise version of GI that generalizes Rigid GI. Namely, the inputs consist of pairs $((G_0, a_0), (G_1, a_1))$, with the promise that $a_i = |\mathrm{Aut}(G_i)|$ for each i, the yes-instances are those where G_0 and G_1 are isomorphic, and the no instances are where they are not isomorphic. Note that when $a_0 = a_1 = 1$ this is exactly Rigid GI. Although this problem might at first seem trivial because computing $|\mathrm{Aut}(G)|$ is equivalent to GI, on further reflection it is not clear what the relationship is between this promise problem and GI; for example, how does knowing $|\mathrm{Aut}(G_i)|$ help to determine isomorphism, if we cannot compute $\mathrm{Aut}(G)$ for general graphs G? The same techniques in this paper can be used to show that this promise problem is in $\mathsf{Promise\text{-}ZPP}^{\mathsf{MKTP}}$.

pre-composing with an automorphism of G that fixes 1 and swaps $2 \leftrightarrow n$ if needed, we may assume that $\pi(n)$ is the larger of $\pi(2)$ and $\pi(n)$, and that $\pi(2)$ is the smaller. After this, we need to describe the image of every vertex remaining, for there are no nontrivial automorphisms that fix 1, 2, and n. But this can now be described by a permutation of $n-3$ elements, together with our previous knowledge of $\pi(1), \pi(2), \pi(n)$. So in total, we've used a number between 1 and $\binom{n-1}{2} \times (n-3)! = (n-1)!/2$ to describe a copy of the n-cycle.

Example 3 (Dumbell graph). *Let $n \geq 6$ be congruent to 2 (mod 4) and consider the "dumbbell graph," consisting of 2 $n/2$-cycles (one on $\{1, \ldots, n/2\}$, and one on $\{n/2+1, \ldots, n\}$) together with an edge between them, say $(1, n/2+1)$. The automorphism group here has size 8, generated by reflecting the first cycle around 1 and by swapping the two cycles, so our goal is to describe any isomorphic copy using a number in $[n!/8]$, rather than $[n!]$. Unlike the previous example, the graph is not vertex-transitive (there is not an automorphism sending any one vertex to any other vertex). So we start by describing the π-images of the orbits under the automorphism group; the vertices 1 and $n/2+1$ form one "central" orbit, and the remaining vertices form $(n-2)/4$ orbits of size 4 (two vertices from each cycle). Describing the π-images of these sets uses a number in $[\binom{n}{2}\binom{n-2}{4}\binom{n-6}{4}\cdots\binom{4}{4}] = [n!/(2\cdot(4!)^{(n-2)/4})]$. By the same argument as before, we may assume that $\pi(n/2+1) > \pi(1)$. We then, for each orbit of size 4, specify which pair of elements is in the cycle containing 1 and which is in the cycle containing $n/2+1$, for a total of $\binom{4}{2}^{(n-2)/4}$. (These sets are the non-singleton orbits of the stabilizer of 1 and $n/2+1$.) Next, without loss of generality, within the orbits adjacent to 1 and $n/2+1$, we may assume that the smaller element gets mapped by π to the smaller element. At this point, we can consider that we are working in the pointwise stabilizer of 1, 2, $n/2+1$, and $n/2+2$; the non-singleton orbits at this point are the $2(n-6)/4$ sets that resulted by partitioning the original orbits of size four into two equal-size subsets. In each of these $2(n-6)/4$ orbits of size two, we must describe the order, using another $2^{2(n-6)/4}$. In total, we've thus used a number between 1 and $\frac{n!}{2\cdot(4!)^{(n-2)/4}} \times \left(\frac{4!}{2!2!}\right)^{(n-2)/4} \times 2^{2(n-6)/4} = n!/8$, as desired.*

Proof of Proposition 1. We first describe aux_G, p_π, and the decoding algorithm, and then we analyze their sizes and running times. Let $A = \mathrm{Aut}(G)$, and for any $i \in \{1, \ldots, n\}$, let A_i be the pointwise stabilizer in A of $1, 2, \ldots, i$, that is, those permutations in $A = \mathrm{Aut}(G)$ that send 1 to 1, send 2 to 2, ..., send i to i.

The string aux_G consists of:

1. The subsets of $V(G) = [n]$ that are the $A = \mathrm{Aut}(G)$-orbits
2. For each $i \in [n]$, the subsets of $V(G)\backslash[i]$ that are the A_i-orbits, as well as flags indicating which of the A_i-orbits are neighbors of vertex i (note that for any A_i-orbit, either all vertices in the orbit are adjacent to i or none of them are).

Now we describe the number p_π. First we state two simple observations regarding using numbers as encodings. Observation 1: A subset of $[n]$ of size k can be encoded as a number in $[\binom{n}{k}]$ in a manner that allows efficient encoding and decoding (see, e. g., [Knu11, Section 7.2.1.3]). Observation 2: Any list of numbers $x_1, \ldots, x_\ell$, where each $x_i \in [n_i]$, can be encoded as a single number in $[n_1 n_2 \cdots n_\ell]$ in the obvious way. For clarity, consider just the case $\ell = 3$: A tuple $(i, j, k) \in [n_1] \times [n_2] \times [n_3]$ can be encoded as a number $d \in [n_1 n_2 n_3]$ by expressing d in $n_2 n_3$-ary notation as $i + e(n_2 n_3)$, then expressing e in n_1-ary notation as $j + kn_1$. (The generalization to larger tuples is obvious.) The string aux_G will contain enough information to determine the sequence $(n_1, \ldots, n_\ell)$ that is necessary in order to decode d as the tuple $(x_1, \ldots, x_\ell)$ in this way.

The number p_π will contain a description of the π-image of each $\mathrm{Aut}(G)$-orbit under π, *as sets*, where a subset of size k of a set of size n can be described by a number between 1 and $\binom{n}{k}$. These sets are described in the same order in which they appear in aux_G. For $i = 1$, p_π then describes the image of each A_1-orbit under π, *as a set within its A-orbit*. For $i = 2, \ldots, n$, p_π then describes each A_{i+1}-orbit as a subset of its A_i-orbit.

The decoder works as follows. It first uses aux_G and the first part of p_π to determine the π-images of the $\mathrm{Aut}(G)$-orbits on $[n]$. Then within each $\mathrm{Aut}(G)$-orbit, the decoder extracts from aux_G a description of the A_1-orbits, and uses the data in p_π to decide where each of those is mapped to by π. For each A_1-orbit that aux_G indicates is adjacent to vertex 1, the decoder adds edges from the vertices in the π-image of that orbit to the smallest vertex in the π-image of the A-orbit of 1. Call this vertex v; note that, without loss of generality, we may assume that $\pi(1) = v$. For if not, there is some automorphism $\rho \in \mathrm{Aut}(G)$ such that $\rho(1) = \pi^{-1}(v)$; replacing π by $\pi' = \pi \circ \rho$, we then have that $\pi'(G) = \pi(\rho(G)) = \pi(G)$ (since $\rho \in \mathrm{Aut}(G)$) and $\pi'(1) = \pi(\rho(1)) = \pi(\pi^{-1}(v)) = v$. After this stage, the π-images of all edges incident to $\pi(1)$ have been described. The algorithm continues in this fashion for $i = 2, \ldots, n$. Correctness is clear: after the last stage, the decoder has described the π-images of all edges incident to $\pi(i)$ for all i.

Even a naive upper bound shows that $|aux_G| \leq O(n^3 \log n)$. The decoder clearly runs in time polynomial in $|aux_G|$ and $|p_\pi|$ (the bit-length of p_π), so all that remains is to verify the claim made in the lemma about the size of p_π, which we must do *exactly*, not just up to big-Oh. To do this, it is useful to take the following pair of "dual" viewpoints on what's happening in the decoder.

We will describe two sequences of vertex-colored graphs, $H_{-1}, H_0, H_1, \ldots, H_n = \pi(G)$ and $G_{-1} = G, G_0, \ldots, G_n$, corresponding to the stages of the decoding algorithm. The colors are used to indicate the partial information about the isomorphism $\pi\colon G \mapsto \pi(G)$, viz. at stage i, the set of vertices of color c in G_i are known to map to the set of vertices of color c in H_i. $G_{-1} = G$, G_0 is G with the $\mathrm{Aut}(G)$-orbits colored (that is, each $\mathrm{Aut}(G)$-orbit is a single color class), and G_i is G with the A_i-orbits colored, for $i = 1, \ldots, n$. Thus, the coloring of G_{i+1} refines that of G_i, and similarly the coloring of H_{i+1} refines that of H_i for all i. The H_i will be the graphs built by the decoder, starting with H_{-1} as the empty graph on n vertices with no colors; H_n will end up as $\pi(G)$. H_0 is still the empty graph, but now with colors corresponding to the π-images of the $\mathrm{Aut}(G)$-orbits. For $i = 1, \ldots, n$, H_i has the π-images of all edges incident to $\pi(1), \ldots, \pi(i)$, and colors corresponding to the π-images of the A_i-orbits.

Let $r_i(p_\pi)$ be the part of p_π used up to and including the i-th stage of the construction ("r" for the **r**ange of numbers used). As in the rest of this proof, we measure this by the size of the range of numbers that has been used in the description so far (rather than their logarithm, the number of bits). So $r_{-1}(p_\pi) = 1$, $r_0(p_\pi) = n!/(n_1! n_2! \cdots n_k!)$ where the n_i are the sizes of the $\mathrm{Aut}(G)$-orbits, etc.

We claim that the following invariants hold:

1. $\mathrm{Aut}(H_i)$ is always a direct product of symmetric groups in their natural action on the π-images of the A_i-orbits; for $i \geq 1$, there is one factor for each A_i-orbit, and for $i = 0$ there is one factor for each $\mathrm{Aut}(G)$-orbit.

2. $r_i(p_\pi) \times |\mathrm{Aut}(H_i)|/|\mathrm{Aut}(G_i)| = n!/|\mathrm{Aut}(G)|$ for all i

Note that in H_n and G_n, every vertex has received its own color, since A_n is the trivial group. So if we can show that (2) holds, then we are done, for then p_π is a number between 1 and $r_n(p_\pi) = r_n(p_\pi) \times |\mathrm{Aut}(H_n)|/|\mathrm{Aut}(G_n)| = n!/|\mathrm{Aut}(G)|$.

To see that (1) holds, consider an auxiliary sequence of graphs H'_i. For $i \in \{-1, 0\}$, $H'_i = H_i$, and for $i \in \{1, \ldots, n\}$, H'_i is the subgraph of H_i induced by the vertices $i+1, i+2, \ldots, n$. Since $1, \ldots, i$ are fixed by A_i, each vertex in $[i] \subset V(H_i)$ has its own color class. Furthermore, since the edges from these vertices to the rest of the H_i are completely determined by the A_i-orbits on the rest of the graph, the automorphism group of H_i is *the same* (not just isomorphic) to that of H'_i, merely by removing i points that are fixed by the entire group. Now note that, by construction, in H'_i there are *no edges*, only vertex colors. Thus it is clear that $\mathrm{Aut}(H'_i)$, and therefore $\mathrm{Aut}(H_i)$, is a direct product of symmetric groups in their natural action on the π-images of the A_i-orbits.

Now for (2). Let $n_{i,j}$ denote the size of the j-th color class in G_i (and thus also in H_i). For example, $n_{-1,1} = n$, and the $n_{0,j}$ are the sizes of the $\mathrm{Aut}(G)$-orbits. By the preceding argument, we see that $\mathrm{Aut}(H_i) = \mathrm{Aut}(H'_i) = S_{n_{i,1}} \times S_{n_{i,2}} \times \cdots \times S_{n_{i,j_i}}$, where j_i is the number of A_i-orbits. In particular, $|\mathrm{Aut}(H_i)| = n_{i,1}!n_{i,2}! \cdots n_{i,j_i}!$.

For $i = -1$, we have $r_{-1}(p_\pi) = 1$, $\mathrm{Aut}(H_i) = S_n$, and $\mathrm{Aut}(G_i) = \mathrm{Aut}(G)$, so $r_{-1}(p_\pi) \times |\mathrm{Aut}(H_{-1})|/|\mathrm{Aut}(G_{-1})|$ indeed equals $n!/|\mathrm{Aut}(G)|$. For $i = 0$, we've used a number between 1 and $\binom{n}{n_{0,1}}\binom{n-n_{0,1}}{n_{0,2}} \cdots \binom{n_{0,j_0}}{n_{0,j_0}}$, where j_0 is the number of $\mathrm{Aut}(G)$-orbits. Thus $r_0(p_\pi) = n!/(n_{0,1}!n_{0,2}! \cdots n_{0,j_0}!)$. As $|\mathrm{Aut}(H_0)| = n_{0,1}! \cdots n_{0,j_0}!$ (from the preceding paragraph), we have $r_0(p_\pi) \times |\mathrm{Aut}(H_0)|/|\mathrm{Aut}(G_0)| = n!/|\mathrm{Aut}(G_0)|$. But note that G_0 is G with the $\mathrm{Aut}(G)$-orbits colored. Since the color classes are just the $\mathrm{Aut}(G)$-orbits, we have that $\mathrm{Aut}(G_0) = \mathrm{Aut}(G)$, and (2) holds for $i = 0$ as well.

For $i = 1, \ldots, n$, we need a slightly more structural argument. Let $A_0 = \mathrm{Aut}(G_0) = \mathrm{Aut}(G)$. For all $i \geq 1$, A_i is the stabilizer in A_{i-1} of the vertex i. Let B_i be the stabilizer of $\pi(i)$ in $\mathrm{Aut}(H_{i-1})$; clearly $\mathrm{Aut}(H_i) \leq B_i$, but in general these two will not be equal, for in B_i the $S_{n_{i-1,\cdot}}$ factor corresponding to the π-image of the A_i-orbit of $\pi(i)$ gets replaced by $S_{n_{i-1,\cdot}-1}$, but in H_i this orbit gets split into the π-images of the A_i-orbits of i, which can be smaller. By the Orbit–Stabilizer Theorem, $|A_{i-1}|/|A_i|$ is the size of the A_{i-1}-orbit of i, and similarly $|\mathrm{Aut}(H_{i-1})|/|B_i|$ is the size of the $\mathrm{Aut}(H_{i-1})$-orbit of $\pi(i)$. But by construction, these two orbits have the same size, so, after re-arranging, we have that $|B_i|/|\mathrm{Aut}(G_i)| = |\mathrm{Aut}(H_{i-1})|/|\mathrm{Aut}(G_{i-1}|)$. Note that to get to this point, we didn't have to use any additional information from p_π (this follows from the "without loss of generality" argument above, that $\pi(i)$ can always be taken to be the smallest element in the π-image of the A_{i-1}-orbit of i). Finally, to construct H_i—equivalently, to get from B_i to $\mathrm{Aut}(H_i)$—we use information from p_π to specify the π-images of the A_i-orbits (except the singleton orbit $\{i\}$) as subsets of the A_{i-1} orbits. Without loss of generality, assume that $n_{i-1,1}$ corresponds to the A_{i-1}-orbit of i. Then this requires an additional range of $(n_{i-1,1}-1)!n_{i-1,2}! \cdots n_{i-1,j_{i-1}}!/(n_{i,1}!n_{i,2}! \cdots n_{i,j_i}!)$ to describe. But the latter is exactly the ratio $|B_i|/|\mathrm{Aut}(H_i)|$, as desired. □

Now that we've described the procedure, it may be useful for the reader to revisit the examples above; they involved somewhat simplified descriptions for their particular cases, and it is an instructive exercise to describe an isomorphic copy of a cycle by actually following the procedure of the proof. In the examples above, it was always the case that $\mathrm{Aut}(G_i)$ was itself a product of symmetric groups in their natural action, for all i. We include our next example to aid intuition, to give an example actually following the procedure of the proof to the letter, and to have an example where $\mathrm{Aut}(G_i)$ is not always just a product of symmetric groups in the natural action.

Example 4 (The Petersen graph)**.** *The Petersen graph has 10 vertices and 15 edges: the vertices can be taken to be pairs of distinct numbers from* $[5]$*, with an edge between two such vertices if the corresponding pairs are disjoint from one another. It is well-known that its automorphism group is given by the natural action of* S_5 *on this description, so our goal is to have* $p_\pi \in [10!/5!]$*. Note, however, already that* $\mathrm{Aut}(G)$ *is not a symmetric group in its natural action, but rather is an action of* S_5 *on a set of size 10. We will denote each vertex by a two-digit number corresponding to the pair of numbers from* $[5]$*. As the Petersen graph is vertex-transitive, without loss of generality we may assume that* $\pi(12) = 12$*.* A_1 *is the stabilizer of* 12 *in* $\mathrm{Aut}(G)$*, which is the same as the setwise stabilizer of* $\{1,2\}$ *in* S_5*, namely* $\mathrm{Sym}(\{1,2\}) \times \mathrm{Sym}(\{3,4,5\}) \cong S_2 \times S_3$*. It is readily verified that the* A_1*-orbits are* $\{34, 35, 45\}$ *and* $\{13, 14, 15, 23, 24, 25\}$*, with the first of these being neighbors of* 12 *and the latter not. So we use* $\binom{9}{3}$ *to describe these orbits.*

Next, we individualize 13*; then* A_2 *is the stabilizer of the set* $\{1,3\}$ *within* $\mathrm{Sym}(\{1,2\}) \times \mathrm{Sym}(\{3,4,5\})$*, which is readily seen to be just* $\mathrm{Sym}(\{4,5\})$*. The* A_2*-orbits that neighbor* 13 *are then* $\{24, 25\}$ *and* $\{45\}$*, and the remaining orbits are* $\{14, 15\}, \{23\}, \{34, 35\}$*. To describe all these orbits takes* $\binom{5}{2}$ *(to describe* $\{24, 25\}$ *within its* A_1*-orbit less* $\{13\}$*),* $\binom{3}{2}$ *to describe* $\{14, 15\}$*, and* $\binom{3}{2}$ *to describe* $\{34, 35\}$*, for a total of* $\binom{5}{2}\binom{3}{2}\binom{3}{2}$*.*

Without loss of generality, we may assume that $\pi(14) < \pi(15)$*. Then we individualize* 14*, which involves specifying* $\pi(25)$ *within* $\pi(\{24, 25\})$ *and specifying* $\pi(35)$ *within* $\pi(\{34, 35\})$*, for an additional factor of* 4*. All remaining orbits are already singletons. The rest of the description involves no new information from* p_π*, just information from* aux_G *as to where there are edges between various* A_3*-orbits. Thus, we've described any copy of the Petersen graph with a number between* 1 *and* $\binom{9}{3}\binom{5}{2}\binom{3}{2}\binom{3}{2}4 = \frac{9\cdot8\cdot7\cdot5\cdot4\cdot3\cdot2\cdot3\cdot2\cdot4}{6\cdot2\cdot2\cdot2} = 9\cdot8\cdot7\cdot5\cdot4\cdot3 = 10\cdot9\cdot8\cdot7\cdot6 = 10!/5!$*, as desired.*

Lemma 1 is a special case (for automorphism groups) of the more general question: Given two permutation groups $H \le G$, can any coset of H in G be described in a time-efficient manner by a number between 1 and $|G|/|H|$? However, it's not clear how to modify the proof of Lemma 1 to work for more general groups, as it relies heavily on the fact that $\mathrm{Aut}(H_i)$ is always a product of symmetric groups in its natural action. We leave this as an open question.

6 What about MCSP?

Why don't the proofs of Theorems 1 and 2 apply to MCSP? Perhaps the easiest way to answer this question is to define a variant of MCSP for which the proofs *do* carry over; this will expose the barrier that will need to be overcome, in order to apply our techniques to (the more customary version of) MCSP.

Define a multiplexer circuit to be a directed acyclic graph with n input gates and an array A with m entries (for some values of n and m), where the array A has values in $\{0,1\}$ stored in each of the m locations. The vertices of a multiplexer circuit are labeled, where the labels describe what kind of gate resides at each vertex. There are five kinds of gates: NOT gates (of fan-in one), AND and OR gates (of fan-in two), INPUT gates (of fan-in one), and MULTIPLEXER gates (of fan-in $\log m$). The size of a multiplexer circuit with an m-entry array circuit is m plus the number of gates in the circuit. The function of each gate should be clear, except for MULTIPLEXER gates: If the wires that feed into a MULTIPLEXER gate evaluate to the binary representation of a number i, then the gate takes on the value stored in the i-th element of the array A.

If we define MCSP using this model of circuit, then it is a routine exercise to show that the proofs of Theorems 1 and 2 carry over with only minor adjustments: A string x has small KT complexity if and only if there is a short description (which can be stored in the array A of a multiplexer circuit) and a small multiplexer circuit that takes a number j as input and computes the j-th bit of x.

However, a multiplexer circuit with an array of size m cannot be implemented by a standard AND, OR circuit without adding at least another term of m to the size—and there simply isn't enough of a gap between the upper and lower bounds on KT complexity in our proofs to absorb this additional term.

It would, indeed, be a very useful thing if there were some way to to boost the "gap" between the upper and lower bounds in our arguments. This would not only show that $\mathsf{GA} \in \mathsf{ZPP}^{\mathsf{MCSP}}$, but it could also form the basis of an argument that would show how to reduce different versions of MCSP (defined in terms of subtly-different circuit models, or in terms of different size parameters) to each other, and to clarify the relationship between MKTP and MCSP. Until now, all of these problems have been viewed as morally-equivalent to each other, although no efficient reduction is known between *any* two of these problems, in either direction. Given the central role that MCSP occupies, it would be very desirable to have a theorem that indicates that MCSP is fairly "robust" to minor changes to its definition. Currently, this is completely lacking.

On a related point, it would be good to know how the complexity of MKTP compares with the complexity of the KT-random strings: $R_{\mathrm{KT}} = \{x : \mathrm{KT}(x) \geq |x|\}$. Until now, all prior reductions from natural problems to MCSP or MKTP carried over to R_{KT}—but this would seem to require even stronger "gap amplification" theorems. The relationship between MKTP and R_{KT} is analogous to the relationship between MCSP and the special case of MCSP that is denoted MCSP' by Murray and Williams [MW15]: MCSP' consists of truth tables f of m-ary Boolean functions that have circuits of size at most $2^{m/2}$.

Acknowledgments

The first author acknowledges the support of National Science Foundation grant CCF-1555409. The second author was supported by an Omidyar Fellowship from the Santa Fe Institute during this work. We thank V. Arvind for helpful comments about the graph automorphism problem and rigid graphs. We thank Alex Russell and Yoav Kallus for helpful ideas on encoding and decoding graphs

References

[ABK$^+$06] E. Allender, H. Buhrman, M. Koucký, D. van Melkebeek, and D. Ronneburger. Power from random strings. *SIAM Journal on Computing*, 35:1467–1493, 2006.

[AD14] Eric Allender and Bireswar Das. Zero knowledge and circuit minimization. In *Mathematical Foundations of Computer Science (MFCS)*, volume 8635 of *Lecture Notes in Computer Science*, pages 25–32. Springer, 2014.

[AHK15] Eric Allender, Dhiraj Holden, and Valentine Kabanets. The minimum oracle circuit size problem. In *32nd International Symposium on Theoretical Aspects* *of Computer Science, STACS 2015, March 4-7, 2015, Garching, Germany*, volume 30 of *LIPIcs*, pages 21–33. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.

[AKRR10] E. Allender, M. Koucký, D. Ronneburger, and S. Roy. The pervasive reach of resource-bounded Kolmogorov complexity in computational complexity theory. *Journal of Computer and System Sciences*, 77:14–40, 2010.

[DH10] R. Downey and D. Hirschfeldt. *Algorithmic Randomness and Complexity.* Springer, 2010.

[For01] Lance Fortnow. Kolmogorov complexity. In *Aspects of Complexity: Minicourses in Algorithmics, Complexity, and Computational Algebra, NZMRI Mathematics Summer Meeting*, volume 4 of *de Gruyter Series in Logic and Its Applications*, pages 73–86. de Gruyter, 2001.

[HILL99] J. Håstad, R. Impagliazzo, L. Levin, and M. Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28:1364–1396, 1999.

[Irn05] Christophe-André Mario Irniger. *Graph matching—filtering databases of graphs using machine learning techniques.* PhD thesis, Universität Bern, 2005.

[KC00] V. Kabanets and J.-Y. Cai. Circuit minimization problem. In *ACM Symposium on Theory of Computing (STOC)*, pages 73–79, 2000.

[Knu11] Donald E. Knuth. *The art of computer programming, Volume 4A, Combinatorial algorithms: Part 1.* Addison–Wesley, Upper Saddle River, NJ, USA, 2011.

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

[KST94] Johannes Köbler, Uwe Schöning, and Jacobo Torán. *The graph isomorphism problem: its structural complexity.* Birkhäuser Verlag, 1994.

[LV08] M. Li and P. Vitanyi. *Introduction to Kolmogorov Complexity and its Applications.* Springer, third edition, 2008.

[MW15] Cody Murray and Ryan Williams. On the (non) NP-hardness of computing circuit complexity. In *Conference on Computational Complexity*, 2015. to appear.

[Tra84] B. A. Trakhtenbrot. A survey of Russian approaches to perebor (brute-force searches) algorithms. *IEEE Annals of the History of Computing*, 6(4):384–400, 1984.

ECCC ISSN 1433-8092

http://eccc.hpi-web.de