

# Some elementary observations on p-isomorphisms in low level complexity classes

Alexander W. Dent\*  
Mathematics Department  
Royal Holloway College

January 23, 2002

## Abstract

The properties of many complexity classes are still unproven or unknown. This uncertainty extends right down to the lowest level of complexity class, with the proposition that  $NP \neq P$  having remained a conjecture for over 20 years. It was once thought that *p-isomorphisms*, an equivalence relation on the set of languages developed by Berman and Hartmanis in the late 1970's, would solve this elementary problem by showing that P and the set of NP-complete languages contained different structures of equivalence classes. This paper will examine some of the consequences of p-isomorphism theory.

## 1 Introduction

This report is primarily concerned with the problem of equivalence amongst algorithms. We generally say that two languages are of equivalent complexity if  $L_1$  reduces  $L_2$  and  $L_2$  reduces to  $L_1$  ( $L_1$  and  $L_2$  are  $\alpha$ -equivalent).

---

\*Supported by a grant from the EPSRC and by the funding of RACAL

This "as hard as each other" approach is very intuitive from a complexity point of view and greatly simplifies the structure of the low level complexity classes (see Figure 1).

However this simplicity comes at a price as this equivalence relation admits the equivalence of the finite and the sparse, and of the sparse and the infinite. It also fails to differentiate between the structure of P and that of the NP-complete languages, making it difficult to show that  $NP \neq P$ .

The concept of p-isomorphisms was introduced in a paper by Berman and Hartmanis in the SIAM journal of computing [1]. In this paper the authors attempted to introduce p-isomorphism as a stricter version of complexity equivalence that kept most of its desirable properties.

**Definition 1 (Berman + Hartmanis)** *Two languages  $L_1 \in \Sigma^*$  and  $L_2 \in \Gamma^*$  are said to be p-isomorphic if there exists a function  $f : \Sigma^* \rightarrow \Gamma^*$  such that:*

1.  *$f$  is invertible.*
2.  *$f$  is a polynomial reduction from  $L_1$  to  $L_2$*
3.  *$f^{-1}$  is a polynomial reduction from  $L_2$  to  $L_1$*

*We shall write  $L_1 \sim L_2$  if  $L_1$  is p-isomorphic to  $L_2$ .*

This definition has the advantage of preserving  $\propto$ -equivalence classes as the union of p-isomorphism classes, as if  $L_1 \sim L_2$  then  $L_1 \propto L_2$  and  $L_2 \propto L_1$ .

## 2 The class P

This section will consider the profound effect of p-isomorphisms on P and, in doing so, will expose some of the basic properties of p-isomorphisms. Under

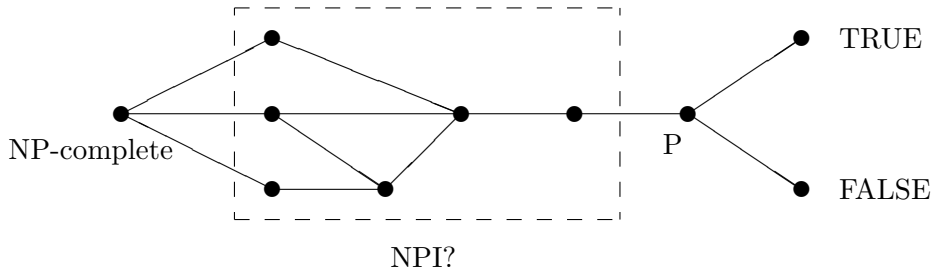


Figure 1: The graph of the standard equivalence classes

the  $\alpha$ -equivalence relation the class P is composed of three distinct equivalence classes:  $\{\text{TRUE}\}$ ,  $\{\text{FALSE}\}$ ,  $P \setminus \{\text{TRUE}, \text{FALSE}\}$ ). However under the p-isomorphism relation the class P splits into a countably infinite number of p-isomorphism classes.

**Lemma 1** *If  $|L_1| = n$ ,  $|L_2| = m$  (where  $n, m \in N \cup \aleph_0$ ) and  $n \neq m$  then  $L_1 \not\sim L_2$*

*Proof* Suppose  $L_1 \sim L_2$  then there exists a bijection  $f$  from  $L_1$  to  $L_2$ . So  $L_1$  and  $L_2$  are equinumerous and hence  $|L_1| = |L_2|$ .

**Lemma 2** *If  $|L| \in N$  then the set of languages p-isomorphic to  $L$  is  $\{L' \subseteq \Sigma^* : |L'| = |L|\}$*

**Corollary 3** *There exists a countably infinite number of p-isomorphism classes in P.*

Whilst this is a possibly useful result it is not actually a very interesting result involving as it does, only the finite languages - which are quite dull from an academic point of view. The structure of the infinite languages contained within P are, however, predictably harder to handle.

**Definition 2** A language  $L$  is sparse if there exists a polynomial  $p(x)$  such that

$$L^{(n)} = \{x \in L : |x| \leq n\} \leq p(n)$$

**Lemma 4** If  $L_1$  is sparse and  $L_2$  is not sparse then  $L_1 \not\sim L_2$ .

*Proof* Let  $p(x)$  be a polynomial that witnesses the sparseness of  $L_1$  and let  $q(x)$  be a polynomial that witnesses that the bijection between  $L_2$  and  $L_1$  is polynomial-time computable, then  $L_2^{(n)} \leq p(q(n))$ . Hence  $L_2$  is sparse.

**Lemma 5**  $L_1 \sim L_2 \iff \Sigma^* \setminus L_1 \sim \Sigma^* \setminus L_2$

*Proof* Any p-isomorphism that witnesses  $L_1 \sim L_2$  also witnesses  $\Sigma^* \setminus L_1 \sim \Sigma^* \setminus L_2$  and vice versa.

Therefore we can summarise the p-isomorphism classes of P as:

- TRUE
- FALSE
- The set of languages of size  $n$ . (This is a countably infinite number of p-isomorphism classes)
- The set of languages whose complement is of size  $n$ . (Also a countably infinite number of p-isomorphism classes)
- P-sparse. The set of infinite languages which are both sparse and contained in P.
- co-P-sparse. The set of languages whose complements are P-sparse.
- P-dense. The set of languages in P which do not fit into any of the above categories.

It should be noted that this is by no means a comprehensive list. It is very unlikely that P-dense is a single p-isomorphism class, it is more probable the union of several p-isomorphism classes and the structure of p-sparse is also very much under debate. However it may be important to note that there exists a countably infinite number of p-isomorphism classes in P and therefore if the set of NP-complete languages contains any less than  $\aleph_0$  p-isomorphism classes then  $NP \neq P$ .

### 3 The class NPI

The class NPI is probably the most elusive of the low level complexity classes, having none of the convenient properties of P or NP-complete languages. However it has a few theorems of note and is therefore worth some small amount of study.

**Definition 3** *The class NPI is the class of problems corresponding to a set of languages that are contained in NP but are neither in P nor NP-complete.*

**Definition 4** *If  $L_1, L_2 \subseteq \Sigma^*$  then*

$$L_1 \oplus L_2 = \{0x : x \in L_1\} \cup \{1x : x \in L_2\}$$

**Theorem 6** *If  $NP \neq P$  then  $NPI \neq \emptyset$*

The proof of this theorem is a corollary of the following larger theorem:

**Theorem 7** *If  $C_1$  and  $C_2$  are two classes of languages such that:*

1.  $C_1$  and  $C_2$  are constructively enumerable.
2.  $C_1$  and  $C_2$  are closed with respect to finite variations.
3. There exists languages  $L_1$  and  $L_2$  such that  $L_1 \notin C_1$  and  $L_2 \notin C_2$ .

then there exists a language  $L$  such that  $L \notin C_1 \cup C_2$  but  $L \propto L_1 \oplus L_2$ .

*Proof* We will define  $L$  to be:

$$L = (L_1 \cap G) \cup (L_2 \cap G^c) \text{ where } x \in G \Leftrightarrow |x| \in A \subseteq \mathbb{N}$$

i.e. for different values of  $|x|$   $L$  is either  $L_1$  or  $L_2$  and the set  $A$ , which has yet to be defined, is just a convenient way of denoting this.

Firstly we require that  $L \propto L_1 \oplus L_2$ , so we define a function  $f : \Sigma^* \rightarrow \Sigma^*$  given by

$$f(x) = \begin{cases} 1x & \text{if } x \in G \\ 0x & \text{if } x \notin G \end{cases}$$

and we stipulate that this must be a polynomial reduction from  $L$  to  $L_1 \oplus L_2$  i.e.  $f$  must be a polynomial-time function. In this case it is enough to show that  $G \in P$ .

Secondly we require that (for  $i = 1, 2$ )  $L \notin C_i$ . As  $C_i$  is constructively enumerable we have  $C_i = \{L_i^j : j \in \mathbb{N}\}$ , and so we may rewrite our condition as  $\forall i \forall j \exists z \in L \Delta L_i^j$ . Again it will be enough to show that  $\forall j \exists z \in G$  s.t.  $z \in L_1 \Delta L_1^j$  and  $\forall j \exists z \in G^c$  s.t.  $z \in L_2 \Delta L_2^j$ .

Let  $z_i^{j,n}$  be the smallest word such that  $|z| \geq n$  and  $z \in L_i \Delta L_i^j$ . We know such a word exists as  $L_i \notin C_i$  and  $C_i$  is closed under finite variations. Let  $r_i(n) = \max_{j \leq n} \{|z_i^{j,n}| + 1\}$  and let  $r(n)$  be a time-constructable function such that  $r(n) \geq \max(r_1(n), r_2(n))$ .

Define  $G$  as:

$$G = \{x : r^{2n}(0) \leq |x| < r^{2n+1}(0), n \geq 0\}$$

First we shall prove that  $G \in P$ . Let  $T$  be a Turing Machine that, given an input  $x$ , halts in time  $r(|x|)$ . Consider the algorithm to decide membership of  $G$  given in Figure 2. This algorithm decides membership of

1. If  $|x| = 0$  then *accept*.
2. Set  $j = 0$ ,  $j$  will be the argument of  $r(j)$  under consideration.
3. Set  $h = 0$ ,  $h$  will be the power of  $r^h(0)$  under consideration.
4. Simulate  $|x|$  steps of  $T$  with input  $1^j$ .
5. If  $T$  does not terminate in this time then  $r^h(0) = r(j) > |x|$ , so *accept* if  $h$  is odd and *reject* if  $h$  is even.
6. If  $T$  terminates in this time then let  $m$  be the number of steps executed by  $T$ , i.e. let  $m = r(j)$ .
7. If  $m = |x|$  then  $r(j) = r^{h+1}(0) = |x|$ , so *accept* if  $h$  is even and *reject* if  $h$  is odd.
8. Set  $j = m$ ,  $h = h + 1$  and return to step 4.

Figure 2: A polynomial-time algorithm for deciding membership of  $G$



$G$  in polynomially bounded time, hence we have satisfied our first criterion: that  $L \propto L_1 \oplus L_2$ .

It therefore remains to show that  $\forall i \forall j \exists z$  such that  $z \in L \Delta L_i^j$ . Let  $n$  be the smallest number such that  $r^{2n}(0) \geq j$  and let  $z$  be the smallest word such that  $|z| \geq r^{2n}(0)$  and  $z \in L_1 \Delta L_1^j$ . Since  $r^{2n+1}(0) = r(r^{2n}(0)) \geq r_1(r^{2n}(0))$  it follows that  $r^{2n+1}(0) > |z|$ . Hence  $z \in G$ . A similar argument holds for  $L_2$ .

*Proof of theorem 6* Let  $C_1 = P$ ,  $C_2 = NP - \text{complete}$ ,  $L_1 = SAT$  and

$L_2 = \emptyset$  then there exists a language  $L$  that is neither in  $P$  nor NP-complete but is a member of NP (as  $L \propto L_1 \oplus L_2 \in NP$ ).

So, if  $NP \neq P$ , we know that there exists at least one p-isomorphism class contained within NPI, however we do not know if there are any other p-isomorphism classes within NPI. Since we have already shown that a sparse language cannot be p-isomorphic to a non-sparse language (lemma 4) it follows that if there exists a sparse NPI language then there exists at least two NPI p-isomorphism classes.

We can also make some deductions about the structure of NPI under p-isomorphisms if we accept some of the more widely accepted conjectures about its structure. Since p-isomorphism preserve the probabilistic complexity classes (i.e. the standard probabilistic complexity classes are closed under p-isomorphisms) we can find at least eight p-isomorphism classes in NPI corresponding to languages with different probabilistic properties (see Figure 3 for a simple graphical representation of how the low level probabilistic complexity classes interact with NP) and this number can be increased if there exists sparse and non-sparse languages with the same probabilistic properties.

## 4 The results of Berman and Hartmanis

The subject of p-isomorphisms was first introduced in a paper published in June 1977 [1]. It contained only two major theorems but also contained detailed descriptions of how most of the major known NP-complete languages interacted and a discussion of sparseness in higher level complexity classes. It was soon followed by a second paper [2] which detailed the p-isomorphisms of more NP-complete languages. In this section we present the two major

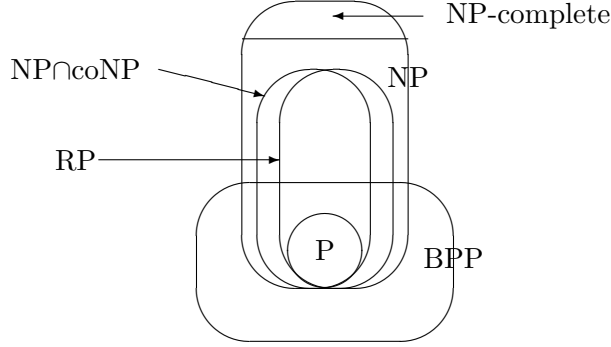


Figure 3: The low level probabilistic complexity classes

theorems detailed in [1], for a more detailed discussion the reader is referred to [3].

**Theorem 8** *Suppose  $L_1$  and  $L_2$  are languages over alphabets  $\Sigma$  and  $\Gamma$  respectively. Let  $p$  and  $q$  be length increasing invertible polynomial reductions from  $L_1$  to  $L_2$  and  $L_2$  to  $L_1$  respectively and let  $p^{-1}$  and  $q^{-1}$  also be polynomial-time functions, then  $L_1$  and  $L_2$  are  $p$ -isomorphic.*

**Lemma 9** *For  $p$  and  $q$  defined above,  $\Sigma^* = R_1 \cup R_2$  and  $R_1 \cap R_2 = \emptyset$  where*

$$R_1 = \{(q \circ p)^k x : k \geq 0 \text{ and } x \notin q(\Gamma^*)\}$$

$$R_2 = \{q \circ (p \circ q)^k x : k \geq 0 \text{ and } x \notin p(\Sigma^*)\}$$

*Proof*  $\{x : x \notin q(\Gamma^*)\} = q(\Gamma^*)^c \subseteq R_1$ .

Suppose  $x \notin R_1 \Rightarrow x \in q(\Gamma^*)$  ( $\Rightarrow \exists y$  s.t.  $q(y) = x$ )

If  $y \notin p(\Sigma^*)$  then  $x \in R_2$  else  $\exists z \in \Sigma^*$  s.t.  $p(z) = y$

$$\Rightarrow (q \circ p)(z) = x$$

$$\Rightarrow z \notin R_1 \text{ (as otherwise } x \in R_1\text{)}.$$

However  $|z| < |x|$ . Therefore, by repeating this process, we can find a smallest element  $u \notin R_1$  such that  $(q \circ p)^k(u) = x$ , and hence a  $v \in \Gamma^*$  such that  $v \notin p(\Sigma^*)$  and  $x = q \circ (p \circ q)^k(v)$ . Hence  $x \in R_2$  and so  $R_1 \cup R_2 = \Sigma^*$ .

Now suppose that  $x \in R_1 \cap R_2$ . Therefore there exists  $x_1 \notin q(\Gamma^*)$  and  $k_1 \in N$  such that  $x = (q \circ p)^{k_1}(x_1)$  as well as  $x_2 \notin p(\Sigma^*)$  and  $k_2 \in N$  such that  $x = (q \circ p)^{k_2} \circ q(x_2)$ .

$$\text{Therefore } (q \circ p)^{k_1}(x_1) = (q \circ p)^{k_2} \circ q(x_2).$$

$$\begin{aligned} \text{If } k_1 \leq k_2 \text{ then } x_1 &= (q \circ p)^{k_2 - k_1} \circ q(x_2) \\ &= q \circ (p \circ q)^{k_2 - k_1}(x_2) \\ &\in q(\Gamma^*) \text{ which is a contradiction.} \end{aligned}$$

$$\begin{aligned} \text{If } k_1 > k_2 \text{ then } (q \circ p)^{k_1 - k_2}(x_1) &= q(x_2) \\ \Rightarrow x_2 &= p \circ (q \circ p)^{k_1 - k_2 - 1}(x_1) \\ &\in p(\Sigma^*) \text{ which is also a contradiction.} \end{aligned}$$

Hence  $R_1 \cap R_2 = \emptyset$ .

*Proof of theorem* We define

$$\begin{aligned} R_1 &= \{(q \circ p)^k x : k \geq 0 \text{ and } x \notin q(\Gamma^*)\} \\ R_2 &= \{q \circ (p \circ q)^k x : k \geq 0 \text{ and } x \notin p(\Sigma^*)\} \\ S_1 &= \{(p \circ q)^k x : k \geq 0 \text{ and } x \notin p(\Sigma^*)\} \\ S_2 &= \{p \circ (q \circ p)^k x : k \geq 0 \text{ and } x \notin q(\Gamma^*)\} \end{aligned}$$

Therefore  $\Sigma^* = R_1 \cup R_2$  and  $\Gamma^* = S_1 \cup S_2$ .

We define a functions  $\phi : \Sigma^* \rightarrow \Gamma^*$  and  $\phi^{-1} : \Gamma^* \rightarrow \Sigma^*$  by

$$\phi(x) = \begin{cases} p(x) & \text{if } x \in R_1 \\ q^{-1}(x) & \text{if } x \in R_2 \end{cases}$$

$$\phi^{-1}(x) = \begin{cases} p^{-1}(x) & \text{if } x \in S_2 \\ q(x) & \text{if } x \in S_1 \end{cases}$$

1. Set  $y = z$
2. Calculate  $q^{-1}(y)$ .
3. If this is calculatable then set  $y$  to be equal to the result, if not then  $\phi(z) = p(z)$
4. Calculate  $p^{-1}(y)$ .
5. If this is calculatable then set  $y$  to be equal to the result, if not then  $\phi(z) = q^{-1}(z)$
6. Repeat from step 2

*NB* This algorithm is polynomial-time as each calculation of  $p^{-1}$  and  $q^{-1}$  reduces the length of  $y$ . Therefore we only have to repeat the algorithm  $O(\log(z))$  times.

Figure 4: A polynomial-time algorithm for calculating  $\phi(z)$



This function is well defined because  $R_1 \cap R_2 = S_1 \cap S_2 = \emptyset$  and is one-one because  $p$  and  $q$  are one-one and if  $\phi(x_1) = \phi(x_2)$  where  $x_1 \in R_1$  and  $x_2 \in R_2$  then  $\phi(x_1) \in S_2$  and  $\phi(x_2) \in S_1$  which is a contradiction. Further  $\phi$  and  $\phi^{-1}$  are inverses. Hence  $L_1$  and  $L_2$  are p-isomorphic provided  $\phi$  and  $\phi^{-1}$  is a polynomial-time function. A polynomial-time algorithm for calculating  $\phi$  is given in Figure 4, a similar construction will give a polynomial-time algorithm for calculating  $\phi^{-1}$ .

**Theorem 10** *Given two languages  $L_1, L_2 \subseteq \Sigma^*$ , a polynomial time reduc-*

tion  $f$  from  $L_1$  to  $L_2$  and that  $L_2$  admits two polynomial-time computable functions  $S_{L_2}$  and  $D_{L_2}$  such that

$$(i) \forall x, y \in \Sigma^* S_{L_2}(x, y) \in L_2 \Leftrightarrow x \in L_2$$

$$(ii) \forall x, y \in \Sigma^* D_{L_2}(S_{L_2}(x, y)) = y$$

then the function  $g(x) = S_{L_2}(f(x), x)$  is a polynomial-time reduction from  $L_1$  to  $L_2$  which admits a polynomial-time computable inverse function.

*Proof* We first need to show that  $g$  is a polynomial-time reduction from  $L_1$  to  $L_2$ . Since  $x \in L_1$  if and only if  $f(x) \in L_2$  and  $f(x) \in L_2$  if and only if  $S_{L_2}(f(x), x) \in L_2$ , we have that  $x \in L_1$  if and only if  $g(x) \in L_2$ . It is obvious that  $g$  is a polynomial-time function, hence  $g$  is a polynomial-time reduction from  $L_1$  to  $L_2$ .

Now suppose  $g(x) = g(y)$  then

$$x = D_{L_2}(S_{L_2}(f(x), x)) = D_{L_2}(g(x)) = D_{L_2}(g(y)) = D_{L_2}(S_{L_2}(f(y), y)) = y$$

and hence  $g$  is injective.

So now we may define a partial inverse of  $g$  by

$$g^{-1}(z) = \begin{cases} D_{L_2}(z) & \text{if } z = S_{L_2}(f(D_{L_2}(z)), D_{L_2}(z)) \\ \text{undefined} & \text{otherwise} \end{cases}$$

It is easy to verify that  $g^{-1}(g(x)) = x$  ( $\forall x \in \Sigma^*$ ) and  $g$  is a polynomial-time function as  $f, S_{L_2}$  and  $D_{L_2}$  are all polynomial-time functions.

**Corollary 11** *Two  $\infty$ -equivalent languages are p-isomorphic if both of them admit polynomial-time computable functions with the properties (i) and (ii) described above and  $S_{L_i}$  is length increasing in the second variable.*

*Proof* The above theorem shows that both languages admit length increasing polynomial-time reductions with polynomial-time computable inverse functions and hence, by theorem 8, they are p-isomorphic.

This can be improved upon for the particular case of *SAT* in NP, a result which will be shown in the next section, however for that result we require the following lemma.

**Lemma 12** *Suppose  $f$  is a one-one polynomial-time function with a polynomial-time inverse that reduces  $L_1$  to  $L_2$  and either  $L_1$  or  $L_2$  admits a one-one polynomial-time function  $Z : \Sigma^* \rightarrow \Sigma^*$  with a polynomial-time inverse such that (i)  $Z(x) \in L_i \Leftrightarrow x \in L_i$  and (ii)  $|Z(x)| > |x|^2 + 1$  for all  $x \in \Sigma^*$  then there exists a one-one, length increasing, polynomial-time function  $g$  with a polynomial-time inverse that reduces  $L_1$  to  $L_2$ .*

*Proof* Let  $q(n)$  be a polynomial bound for  $f$  and  $f^{-1}$ .

Suppose  $L_1$  admits a function  $Z$  as described then there exists an integer  $n$  such that  $|Z^n(x)| > q(|x|)$ . It follows that  $|f \circ Z^n(x)| > |x|$  as if the converse held then  $|f^{-1} \circ f \circ Z^n(x)| \leq q(|x|)$  which is a contradiction to the definition of  $n$ . Hence we define  $g = f \circ Z^n$ .

Similarly suppose  $L_2$  admits a function  $Z$  as described then there exists an integer  $n$  such that  $|Z^n(x)| > q(|x|)$  and again as  $|f^{-1}(x)| \leq q(|x|)$  we have that  $q(|f(x)|) \geq |x|$  and so we may set  $g = Z^n \circ f$  and then  $|Z^n \circ f(x)| > q(|f(x)|) \geq |x|$ .

Since a simple Turing Machine has only one input tape a special encoding system, known as a *pairing function*, will have to be used if the Turing machine is required to consider inputs with more than one component. A pairing function is a function  $\langle \cdot, \cdot \rangle : \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$  which uniquely writes both inputs to a single piece of tape in such a way that both are easily recoverable. A simple example of this is:

$$\langle x, y \rangle = x_1 0 x_2 0 \dots x_n 0 y_1 1 \dots y_m 1 \text{ if } x = x_1 x_2 \dots x_n \text{ and } y = y_1 y_2 \dots y_m$$

**Definition 5** *A set  $C$  is a polynomial cylinder ( $C$  is paddable) if there exists*

a polynomial time function  $p$  with a polynomial time inverse function, a padding function, such that:

$$(i) \forall x, y \in \Sigma^* p(x, y) \in C \Leftrightarrow x \in C$$

$$(ii) \forall x, y \in \Sigma^* p^{-1}(p(x, y)) = \langle x, y \rangle$$

Obviously we have that if  $L_1 \sim L_2$  and  $L_1$  is a polynomial cylinder then  $L_2$  is a polynomial cylinder and if  $L_1$  and  $L_2$  are  $\alpha$ -equivalent polynomial cylinders with length increasing padding functions then  $L_1 \sim L_2$ .

## 5 The class NP-complete

The structure of the class of languages which are NP-complete is something of a holy grail in the subject of p-isomorphism as it is this that would allow us to determine, to a large extent, the similarities or differences between P and NP. Consequently, and in keeping with modern mathematics inability to state with any certainty whether  $P=NP$ , we have more conjectures in this area than results.

**Conjecture 13 (The p-isomorphism conjecture)** *All NP-complete languages are p-isomorphic.*

The wealth empirical evidence to support this result is almost overwhelming, as Theorem 11 has been used to prove that almost every known NP-complete language is p-isomorphic to *SAT* [1][2], but there has been little headway in proving that a general, abstract NP-complete language must be p-isomorphic to *SAT*. As an example of the technique that is generally used we present the following theorem:

**Theorem 14** *SAT is p-isomorphic to CLIQUE*

**Definition 6** *The language SAT is the set of all encodings of boolean statements in conjunctive normal form that admit satisfying truth assignments.*

**Definition 7** *The language CLIQUE is the set of all encodings of the form  $k(\text{blank})G$  where  $k$  is a natural number and  $G$  is an encoding of a graph such that there exists a subgraph of  $G$  with exactly  $k$  vertices and there exists an edge between every vertex in the subgraph.*

For more information on these NP-complete problems the reader is referred to [4]

*Proof of theorem 14* We start by showing that there exists functions  $S_{SAT}$  and  $D_{SAT}$  with the required properties. Consider the function  $S_{SAT}(w, y)$  calculated as follows. First we check to see if  $w$  is a boolean string and set  $r$  to be the number of boolean variables it contains (setting  $r = 0$  if  $w$  is not a boolean string). Let  $y_j$  be the  $j^{th}$  bit of the string  $y$  and let

$$S_{SAT}(w, y) = w \wedge (x_{r+1} \vee \neg x_{r+1}) \wedge z_1 \wedge z_2 \wedge \dots \wedge z_n$$

where  $z_j$  is the literal

$$z_j = \begin{cases} x_{r+1+j} & \text{if } y_j = 1 \\ \neg x_{r+1+j} & \text{if } y_j = 0 \end{cases}$$

We define  $D_{SAT}(x)$  to be the function that examines  $x$  to see if it is a proper suffix of the above style and returns  $x$  if it doesn't and  $x$  stripped of that suffix if it does. Clearly both functions are p-time and  $S_{SAT}$  is length increasing.

Next we must show that there also exists functions  $S_{CLIQUE}$  and  $D_{CLIQUE}$  with the required properties. Consider the function  $S_{CLIQUE}(w, y)$  calculated as follows: First we check if  $w$  is of the form  $k(\text{blank})G$  where  $k$  is an

encoding of a natural number and  $G$  is a graph and then we determine  $r$ , the number of vertices of  $G$  (if  $w$  is not of the required form then we set  $r = 0$ ). If we assume that  $G$  has vertices  $v_1, v_2, \dots, v_r$  then  $S_{CLIQUE}(w, y)$  outputs  $k + 1(\text{blank})G'$  where  $G'$  has vertices  $v_1, \dots, v_r, \dots, v_{r+2|y|+1}$  and all the edges of  $G$  along with  $\forall j \leq r, \forall i \leq |y| (v_j, v_{r+2i})$  if  $y(i) = 1$  (where  $y(i)$  is the  $i^{\text{th}}$  bit of  $y$ ) or  $(v_j, v_{r+2i-1})$  if  $y(i) = 0$  and  $(v_j, v_{r+2|y|+1})$ . Obviously  $S_{CLIQUE}$  is a length increasing polynomial-time function and we may define  $D_{CLIQUE}$  as a polynomial-time function that strips off all vertices that do not connect to the last vertex.

Hence we have satisfied the conditions of corollary 12 and therefore proven that  $CLIQUE \sim SAT$ .

**Theorem 15** *If  $L$  is a paddable NP-complete language then  $L$  is p-isomorphic to  $SAT$ .*

*Proof* We use theorem 10 and lemma 12 from the previous section. We have already shown that  $SAT$  is a polynomial cylinder and, by assumption, so is  $L$ . Therefore there exists one-one polynomial-time reductions from  $SAT$  to  $L$  and  $L$  to  $SAT$  with polynomial-time inverses. Since we may increase the length of a CNF boolean string by adding dummy variables we may find a one-one polynomial-time function  $Z : \Sigma^* \rightarrow \Sigma^*$  with a polynomial-time inverse such that (i)  $Z(x) \in SAT \Leftrightarrow x \in SAT$  and (ii)  $|Z(x)| > |x|^2 + 1$ . Hence we may find length-increasing polynomial-time reductions from  $SAT$  to  $L$  and  $L$  to  $SAT$  with polynomial-time inverses. Therefore  $SAT \sim L$ .

## 6 The one-way isomorphism conjecture

The results of Berman and Hartmanis stood for over five years without being significantly improved upon. It took the work of Deborah Joseph and Paul Young in 1985 [5] to bring to light the implications of p-isomorphism once again. The paper had its roots firmly established in recursive function theory, the underlying study of the functions that can be calculated using a Turing machine, and we will be forced to adopt some of their terminology through the next few sections.

**Definition 8** Let  $\phi_i(x)$  be the output of the recursive function corresponding to the program  $i$  with the input  $x$ . We use the notation  $\phi_i(x) \uparrow$  to indicate that the program  $i$  fails to halt with input  $x$  and  $\phi_i(x) \downarrow$  to indicate that program  $i$  produces an output with input  $x$ . If  $\phi_i(x) \downarrow$  then  $\Phi_i(x)$  is the amount of time taken for a Turing Machine for  $\phi_i$  with input  $x$  to terminate. Hence we may define the sets:

$$W_i = \{x : \phi_i(x) \downarrow\}$$

$$P^{(k)} = \{W_i : \text{program } i \text{ is deterministic and } \phi_i(x) \downarrow \text{ implies } \Phi_i(x) \leq |i| \cdot |x|^k + |i|\}$$

$$NP^{(k)} = \{W_i : \phi_i(x) \downarrow \text{ implies } \Phi_i(x) \leq |i| \cdot |x|^k + |i|\}$$

Thus:

$$P = \bigcup_{k \geq 0} P^{(k)}$$

$$NP = \bigcup_{k \geq 0} NP^{(k)}$$

The main objective of the Joseph and Young paper was to introduce the idea of  $k$ -creative sets.

**Definition 9** Let  $k_0$  be a fixed integer. A set  $C \in NP$  is  $k_0$ -creative if there exists a polynomial-time function  $f$  (a productive function) such that for all  $i$  with  $W_i \in NP^{k_0}$

$$f(i) \in C \cap W_i \text{ or } f(i) \notin C \cup W_i$$

$$(\text{or } f(i) \in C \Leftrightarrow f(i) \in W_i)$$

**Theorem 16** *Every  $k_0$ -creative set is NP-complete.*

*Proof* Let  $C$  be a  $k_0$ -creative set with a productive function  $f$  and let  $A$  be any set in NP. Since  $A \in NP$  we have that  $A \in NP^{(k)}$  for some fixed  $k$ . Define a polynomial computable function  $g$  such that:

$$\phi_{g(y)}(x) = \begin{cases} 1 & \text{if } y \in A \\ \uparrow & \text{otherwise} \end{cases}$$

Therefore, given that  $A \in NP^{(k)}$ , we have that

$$\Phi_{g(y)}(x) \leq |a| \cdot |y|^k + |a|$$

for all  $x \in \Sigma^*$ , where  $a$  is a constant term only slightly bigger than the constant term of the non-deterministic program which accepts  $A$ . Since we would like  $W_{g(y)}$  to be in  $NP^{(k_0)}$  for all  $y$ , we have to prove that

$$\Phi_{g(y)}(x) \leq |g(y)| \cdot |x|^{k_0} + |g(y)|$$

for all  $y$  and  $x$ . This can be easily arranged by padding the function  $g(y)$  so that  $|g(y)| > |a| \cdot |y|^k + |a|$ , and such a  $g$  can be constructed even if we wish to make  $g$  a polynomial-time function.

Hence  $W_{g(y)} \in NP^{(k_0)}$  for all  $y$ .

So, if  $y \in A$  then  $W_{g(y)} = \{0, 1, 2, 3, \dots\}$  and so  $f(g(y)) \in W_{g(y)}$ , however if  $y \notin A$  then  $W_{g(y)} = \emptyset$  and so  $f(g(y)) \notin W_{g(y)}$ . However, as  $C$  is  $k_0$ -creative we have  $f(g(y)) \in W_{g(y)}$  if and only if  $f(g(y)) \in C$ , thus  $y \in A$  if and only if  $f(g(y)) \in C$ .

Hence  $A \propto C$  and so  $C$  is NP-complete.

Hence we know that should a  $k$ -creative set exist then it would be NP-complete, we use the following theorem to prove that  $k$ -creative sets exist.

**Definition 10** A function  $f$  is honest if there exists a polynomial  $p$  such that  $|x| \leq p(|f(x)|)$  for all  $x \in \Sigma^*$

**Theorem 17** If  $f$  is any polynomial time honest, one-one function then

$$K_f^k = \{f(i) : \Phi_i(f(i)) \leq |i| \cdot |f(i)|^k + |i|\}$$

is a  $k$ -creative set, with productive function  $f$ .

*Proof* It is easy to see that  $K_f^k$  is in NP by observing that, since  $f$  is honest, given  $y$  we may guess a value  $x$  such that  $f(x) = y$  in polynomial time. Once this is done it is easy to check the time bound.

To see that  $f$  is a productive function for  $K_f^k$  we note that for any  $W_i \in NP^{(k)}$

$$\begin{aligned} f(i) \in W_i &\Leftrightarrow \phi_i(f(i)) \downarrow \\ &\Leftrightarrow \Phi_i(f(i)) \leq |i| \cdot |f(i)|^k + |i| \\ &\Leftrightarrow f(i) \in K_f^k \end{aligned}$$

The classification of languages in NP is the first step to proving a counter-example to the Berman and Hartmanis conjecture (conjecture 13). However  $k$ -creativity is not closed under  $p$ -isomorphisms, that is a language  $p$ -isomorphic to a  $k$ -creative set need not be  $k$ -creative itself. Inexplicably it was the well known cryptographic concept of one-way functions that allowed Joseph and Young to proceed.

**Definition 11** A function  $f$  is one-way if it is a polynomial time computable function which is one-one, honest and for which a polynomial time inverse function does not exist.

The existence of one-way functions is still under some doubt. Although most scholars and cryptographers accept the existence of one-way functions

there is no proof that they exist even if  $\text{NP} \neq \text{P}$  (obviously if  $\text{NP} = \text{P}$  then one-way functions do not exist).

**Theorem 18** *If one-way functions do not exist then all  $k$ -creative sets are polynomial cylinders.*

*Proof* In the proof of theorem 16 we could have stipulated that  $g$  is a function of two variables  $y$  and  $z$  and

$$\phi_{g(y,z)}(x) = \begin{cases} 1 & \text{if } y \in A \\ \uparrow & \text{otherwise} \end{cases}$$

Since  $g$  is a polynomial time function and there exists no one-way functions then  $g$  must admit a polynomial time inverse  $g^{-1}$  such that  $g^{-1}(g(x, y)) = \langle x, y \rangle$ , by the same reasoning the productive function  $f$  must have a polynomial time inverse  $f^{-1}$ . Hence, and by the same reasoning as in the proof of theorem 16,  $f \circ g$  is the required padding function.

**Corollary 19** *If one-way functions do not exist then all  $k$ -creative sets are  $p$ -isomorphic to SAT.*

**Conjecture 20 (The one-way isomorphism conjecture)** *If one-way functions exist then not all  $k$ -creative sets are  $p$ -isomorphic to SAT.*

It is worth noting that Hemachandra and Hartmanis [7] have produced a relativised counter-example to the one-way isomorphism conjecture by constructing a relativised world that has non-isomorphic NP-complete sets, but lacks one-way functions.

## 7 More on one-way functions

The unexpected link between the existence of one-way functions and led Osamu Watanabe in a paper in Theoretical Computer Science conjecture a

simpler way of constructing of an NP-complete set that is not p-isomorphic to  $SAT$  [6]. The paper is based on the simple idea that if  $f$  is a one-way function then it seems intuitively unlikely that  $f(SAT) \sim SAT$ .

The paper approaches the situation by considering the case where the p-isomorphism conjecture fails, that is where there exists multiple NP-complete p-isomorphism classes and attempting to prove that in this case, for a complete language  $L$ ,  $f(L) \not\sim L$ .

**Lemma 21** *Let  $D, L_1, L_2 \subseteq \Sigma^*$  be any languages such that*

1.  $D \in P$ ,
2.  $L_1, L_2 \subseteq D$  and  $L_1 \cap L_2 = \emptyset$ ,
3.  $L_1$  and  $L_2$  are polynomial cylinders with padding functions  $\pi_1$  and  $\pi_2$  respectively.

*then  $L_1$  has a padding function  $\pi$  whose range is included in  $D$ .*

*Proof* Let

$$\begin{aligned} \pi_3(x, y) &= \pi_2(x_0, \langle x, y \rangle) \text{ for some fixed } x_0 \in L_2 \\ \pi(x, y) &= \begin{cases} \pi_1(x, y) & \text{if } \pi_1(x, y) \in D \setminus \pi_3(\Sigma^*, \Sigma^*) \\ \pi_3(x, y) & \text{otherwise} \end{cases} \end{aligned}$$

$\pi$  is a polynomial time function with a polynomial time inverse function as  $D \in P$  and  $\pi_i$  has a polynomial time inverse function ( $i = 1, 2, 3$ ) and obviously  $\pi(\Sigma^*, \Sigma^*) \subseteq D$  as  $\pi_3(\Sigma^*, \Sigma^*) \subseteq D$ .

**Theorem 22** *Let  $SAT$  and  $L$  be non-isomorphic NP-complete languages and let  $f$  be a polynomial reduction from  $SAT$  to  $L$  such that  $f(\Sigma^*) \in P$ , then either  $f(SAT)$  or  $\overline{f(SAT)}$  is an NP-complete language that is not p-isomorphic to  $SAT$ .*

*Proof* First we show that both  $f(SAT)$  and  $\overline{f(\overline{SAT})}$  are NP-complete sets. Firstly we note that  $f(SAT) = L \cap f(\Sigma^*)$  and  $\overline{f(\overline{SAT})} = L \cup \overline{f(\Sigma^*)}$ . Hence both sets are in NP as  $f(\Sigma^*) \in P$ .

Now assume for a contradiction that  $SAT \sim f(SAT) \sim \overline{f(\overline{SAT})}$ . This means that  $f(SAT)$ ,  $\overline{f(\overline{SAT})}$  are polynomial cylinders and so (using the previous lemma and taking  $D = f(\Sigma^*)$ ) there exists a padding function  $\pi$  for  $f(SAT)$  whose range is included in  $f(\Sigma^*)$ . Now define  $g(x) = \pi(f(x), x)$ , then for all  $x \in \Sigma^*$ :

$$\begin{aligned}
x \in SAT &\Leftrightarrow f(x) \in f(SAT) \\
&\Leftrightarrow \pi(f(x), x) \in f(SAT) \\
&\Leftrightarrow \pi(f(x), x) \in L \text{ (range}(\pi) \subseteq f(\Sigma^*) \text{ and } f(SAT) = L \cap f(\Sigma^*)) \\
&\Leftrightarrow g(x) \in L
\end{aligned}$$

Now by construction  $g$  is a polynomial-time function with a polynomial-time inverse, hence  $g(\pi(g^{-1}(x), y))$  is a padding function for  $L$  and so  $SAT \sim L$  (by theorem 15) which is the required contradiction.

Lastly we produce some counter-examples for the usefulness of one-way functions given by Ganesan in [8]. These serve to show that whilst one-way functions are intrinsically linked to the p-isomorphism conjectures Watanbae's idea that  $f(SAT) \not\sim SAT$  and Joseph And Young's idea that  $K_f^k \not\sim SAT$  for a one-way function  $f$  are not necessarily true. For this will use a more basic NP-complete language *MACHINE* which is p-isomorphic to *SAT*, again for more details of this language the reader is referred to [4].

**Definition 12** *MACHINE* is the set of all elements of  $\Sigma^*$  of the form  $\langle i, x, 0^n \rangle$  where a non-deterministic Turing machine for  $\phi_i$  will accept  $x$  in time bounded above by  $n$ .

**Theorem 23** *If one-way functions exist then there exists a one-way function  $f$  such that  $MACHINE \sim f(MACHINE)$ .*

*Proof* Let  $g$  be any given one-way function and let  $\phi_j$  be the recursive function corresponding to the Turing machine that immediately rejects all inputs. Let  $d$  be an integer that bounds above the number of steps taken by the Turing machine for  $\phi_j$ . Let

$$f(x) = \begin{cases} 2g(y) & \text{if } g \text{ is of the form } \langle j, y, d \rangle \\ 2x + 1 & \text{otherwise} \end{cases}$$

Since  $g$  is a one-way function, so clearly is  $f$ . However as there exists no  $y$  for which  $\langle j, y, d \rangle \in MACHINE$  we find that  $h(x) = 2x + 1$  is a one-one, length increasing polynomial-time reduction from  $MACHINE$  to  $f(MACHINE)$  with a polynomial-time inverse. Hence  $MACHINE \sim f(MACHINE)$ .

**Theorem 24** *If one-way functions exist then there exists a one-way function  $f$  such that for all  $k \geq 1$ ,  $K_f^k$  has a one-one, polynomial-time, length increasing productive function with a polynomial-time inverse. Hence  $K_f^k \sim MACHINE$ .*

*Proof* Again, let  $g$  be any one-way function and  $\phi_j$  be the recursive function corresponding to the Turing machine that immediately rejects all inputs. Using  $j$ , we may find a polynomial-time function  $p(x)$  with a polynomial-time inverse that outputs a new encoding for a Turing machine that rejects all inputs.<sup>1</sup> Let

$$f(x) = \begin{cases} 2g(x) & \text{if } x = p(y) \text{ for some } y \\ 2x + 1 & \text{otherwise} \end{cases}$$

---

<sup>1</sup>This padding function works by adding one extra state to the Turing machine for  $j$  for each bit of  $x$ . These states can never be reached by the Turing machine from the initial state and are connected to one of a final two states depending on the parity of a bit of  $x$ . Obviously this construction is polynomial-time, has a polynomial-time inverse and produces a Turing machine which works exactly like that of  $\phi_j$ .

Clearly  $f$  is one-way. Now, consider the set  $K_f^k$  with  $f$  as a productive function. We will show that  $h(x) = 2x + 1$  is a polynomial-time, length increasing productive function for  $K_f^k$  with a polynomial-time inverse.

If  $x \notin \text{range}(p)$  then  $g(x) = h(x) = 2x + 1$ , hence  $h(x) \in K_f^k \Leftrightarrow h(x) \in W_x$ .

If  $x \in \text{range}(p)$  then  $x$  is an encoding of a Turing machine which rejects all its inputs, i.e.  $W_x = \emptyset$ . Since  $f(x) = 2g(y)$  we have that  $h(x) \notin \text{range}(p)$ , so  $h(x) \notin K_f^k \subseteq \text{range}(f)$ . Since  $W_x = \emptyset$  it follows that  $h(x) \notin W_x$  and therefore that  $h(x) \in K_f^k \Leftrightarrow h(x) \in W_x$ .

So  $h$  is a productive function for  $K_f^k$  but  $h$  is a length increasing, polynomial-time function with a polynomial-time inverse function, therefore  $K_f^k \sim \text{SAT} \sim \text{MACHINE}$ .

## 8 Results of classical studies

So far we have attempted to show that the study of p-isomorphisms has some distance to go before it can be used for its original intention: proving that  $\text{NP} \neq \text{P}$ . The focus of concentration so far has, thanks to two groundbreaking papers, been concentrated on two areas: (i) showing that the NP-complete languages have none of the obvious properties of P under p-isomorphisms [1] and (ii) relating the existence of multiple NP-complete to the existence of one-way functions [5]. If the subject is to be carried forward then it seems likely that a new approach must be examined (such as the algebraic properties that relate the p-isomorphism classes together) or a break through must be made in the study of one-way functions (such as being able to classify necessary and sufficient conditions for a one-way productive function to produce a k-creative set that is not isomorphic to *SAT* and prove that one-way functions exist, a result that would, in itself, prove that  $\text{NP} \neq \text{P}$ ).

We have so far attempted to concentrate on the affects of p-isomorphism theory in the classes NP and P and in particular the cardinality of these sets. It has therefore painted a somewhat bleak picture of the subject. The triumphs of p-isomorphism theory have not been related to NP and P, the triumphs of p-isomorhpism theory have been related to the super-polynomial complexity classes and to the relativised versions of the problems. In these cases many useful and interesting results have been proven, however the largest open problem of complexity theory has been largely unmoved by this interesting area of study.

## References

- [1] L. Berman and J. Hartmanis, *On isomorphism and density of NP and other complete sets*, SIAM Journal on Computing, 6, 305-322, 1977.
- [2] L. Berman and J. Hartmanis, *On polynomial time isomorphisms of some new complete sets*, Journal of Computing System Science, 16, 418-422, 1978.
- [3] D. Bouvet and P. Crescenzi, *Introduction to the theory of complexity*, Prentice Hall, 1994.
- [4] M. Garey and D. Johnson, *Computers and intractability: a guide to the theory of NP-completeness*, Freeman, 1979.
- [5] D. Joseph and P. Young *Some remarks on witness functions for non-polynomial and noncomplete sets in NP*, Theoretical Computer Science, 39, 225-237, 1985
- [6] O. Watanabe, *On the p-isomorphism conjecture*, Theoretical Computer Science, 83, 337-343, 1991.

- [7] J. Hartmanis and L. Hemachandra, *One-way functions and the non-isomorphism of NP-complete sets*, Theoretical Computer Science, 81, 155-163, 1991.
- [8] K. Ganesan, *One-way functions and the p-isomorphism conjecture*, Theoretical Computer Science, 129, 309-321, 1994.