An Optimally Data Efficient Isomorphism Inference Algorithm

PETER A. FLANAGAN*

CLA Computer Science Dept., Boston University, Boston, Massachusetts 02215

The time, space, and data complexity of an optimally data efficient isomorphism identification algorithm are presented. The data complexity, the amount of data required for an inference algorithm to terminate, is analyzed and shown to be the minimum possible for all possible isomorphism inference algorithms. The minimum data requirement is shown to be \( \lceil \log_2(n) \rceil \), and a method for constructing this minimal sequence of data is presented. The average data requirement is shown to be approximately \( 2 \log_2(n) \). The time complexity is \( O(n^2 \log_2(n)) \) and the space requirement is \( O(n^2) \).

1. INTRODUCTION

Inference algorithms read data and output a structure which generalizes the data, although the amount of input data required to find the correct generalization is not generally known in advance. Many inference algorithms have been proposed from the early work of Solomonoff (1964) to the wide variety of inference work done currently (Angluin and Smith, 1983). However, little has been done to determine the amount of data necessary to infer a structure. The minimum and average amount of data required to infer an isomorphism is analyzed in this paper.

The following game illustrates the problem solved by this inference algorithm. Suppose there is a box with \( n \) lights and \( n \) switches on it. Each light is controlled by a single switch and each switch controls a single light. When a switch is off, the light it controls is also off. When a switch is on however, the corresponding light is on only if the power button is depressed. No switch can be moved while the power button is depressed. The problem is: how many times must the power button be depressed in order to discover which switch controls each light. In a variation of this problem, we can not alter the switches ourself, but can only look over the shoulder of a monkey, who continually sets the switches randomly and then presses the power button.

The analysis of the data complexity of an algorithm is instructive since it allows us to measure how efficiently an algorithm uses the data available to it, and thus compare algorithms along another dimension.

* Present address: Digital Equipment Corporation, CADM Group, 100 Minuteman Road APO-2/D4, Andover, Massachusetts 01810.
Although there has been little work done on the analysis of the data complexity of inference algorithms, Gold (1967; 1972) has laid down much of the theory and terminology. Barzin, Biermann, Blum and Blum, Feldman, Grenander, and Kugel have also been very active in related theoretical inference issues.

2. Definitions, Notation, and Terminology

Upper case letters are used to denote sets, lower case letters denote elements of those sets. An (finite) isomorphism $F$ is a one-to-one function from a finite domain set $D(F)$ onto a finite range set $R(F)$. For example,

$$F = \{(1, 3), (2, 1), (3, 2)\}$$

is an isomorphism, where $D(F) = \{1, 2, 3\}$ and $R(F) = \{1, 2, 3\}$ and $F$ takes the element 1 in the domain set to the element 3 in the range set, etc. The inverse image of a subset $Y$ of the range set, denoted $F^{-1}(Y)$ is the set of elements $X$ in the domain such that $Y = F(X)$. The intersection of the sets $X$, $Y$ is denoted $X \cap Y$ and contains only the elements which occur in both $X$ and $Y$. The set difference $X - Y$ contains only the elements which occur in $X$ and do not occur in $Y$. The notation $\lceil r \rceil$ where $r$ is a real number, is called the ceiling of $r$ and is the smallest integer greater than or equal to $r$. The cardinality of $X$ where $X$ is a set, is denoted $|X|$ and is the number of elements in $X$.

By a sample of $F$ we mean an ordered pair $(X, Y)$ such that $X \subset D(F)$, and $Y = F(X)$. If $(A, B)$ is a sample of $F$, then $A$ is called the $X$-part of the sample, and $B$ is called the $Y$-part. Note $F$ is a function which maps a set to another set. For example if $|D(F)| = 5$, and

$$F = \{(1, 4), (2, 2), (3, 5), (4, 3), (5, 1)\}$$

then $\{(2, 3, 5), \{1, 2, 5\}\}$ is one possible sample of $F$, and $\{(1, 4), \{3, 4\}\}$ is another of the $2^5 = 32$ possible samples of $F$, one for each possible subset of $D(F)$. We adopt the convention that $F(\{\}\} = \{\}$, that is, $F$ maps the empty set into the empty set. Clearly $F(D(F)) = R(F)$. A sample $(X, Y)$ is said to be consistent with an isomorphism $F$ when $X \subset D(F)$ and $Y = F(X)$.

3. Overview of the Inference Model

Suppose an isomorphism $F$ exists between the two sets $D(F)$, the domain of $F$, and $R(F)$, the range of $F$. When the cardinality of $D(F)$ is known, i.e. $|D(F)| = n$, then $|R(F)| = n$ since $F$ is an isomorphism.
There are $n!$ different possible isomorphisms from $D(F)$ to $R(F)$. The job of the inference algorithm is to decide which isomorphism is correct by observing samples of $F$.

An inference algorithm for $F$ is any algorithm which when presented with $|D(F)|$ and a sequence of samples, terminates when enough samples have been read to uniquely determine $F$.

4. Algorithm Introduction

The algorithm presented in this section learns (finds, discovers, infers) which isomorphism $F$ exists between the domain of $F$, $D(F)$, and the range of $F$, $R(F)$. The cardinality of $D(F)$ is explicitly part of the algorithm, and the domain and range sets are assumed to be $\{1, 2, \ldots, n\}$. When the cardinality of $D(F)$ is known in advance, the termination criteria is clear, it is otherwise indeterminate since a future input could alter our solution. This algorithm shares many properties with other inference algorithms—potentially infinite sequence of input, a search through a space, and a desire to make optimal use of input data. We analyze the algorithm to (1) show it uses its input as efficiently as possible, (2) show the input does not need to be saved, (3) find the minimum number of samples (the maximum is infinite, of course), and (4) find the average number of samples required to learn the unknown function, when the distribution over the sample space is uniform.

5. Inference Deductions

Suppose a particular isomorphism $F$ exists between two finite sets, $D(F)$ and $R(F)$. Suppose further that the only information about the isomorphism available to us is in the form of a sequence of samples. For example we might be presented with the sample

$$(\{1, 3, 4\}, \{3, 4, 5\}).$$

This sample tells us that the elements 1, 3, and 4 are in $D(F)$ and the elements 3, 4, and 5 are in $R(F)$, and that one of $3! = 6$ possible mappings must exist in the isomorphism $F$. Namely,

$$\{(1, 3), (3, 4), (4, 5)\} \quad \text{or} \quad \{(1, 3), (3, 5), (4, 4)\}$$

or

$$\{(1, 4), (3, 3), (4, 5)\} \quad \text{or} \quad \{(1, 4), (3, 5), (4, 3)\}$$
or
\{(1, 5), (3, 3), (4, 4)\} or \{(1, 5), (3, 4), (4, 3)\}.

A subsequent sample might be
\{(1, 2, 4), (2, 3, 4)\}.

Combining this new information with the above, we know that the element 3 in \(D(F)\) must map to the element 5 in \(R(F)\), and the element 2 in \(D(F)\) must map to the element 2 in \(R(F)\), since that mapping is the only one consistent with the two samples above. This method is essentially exhaustive enumeration. A more efficient method is to use the following:

**Lemma 1.** Given that \(F(X_1) = Y_1\), and \(F(X_2) = Y_2\), then
\[
F(X_1 \cap X_2) = Y_1 \cap Y_2
\]
and
\[
F(X_1 - X_2) = Y_1 - Y_2.
\]

**Proof.** Part (1) by definition of isomorphism, says that if an element of \(D(F)\) appears in any two subsets \(X_1, X_2\) of \(D(F)\), then its image must appear in both \(F(X_1)\) and \(F(X_2)\), and if an element of \(R(F)\) appears in any two subsets \(Y_1, Y_2\) of \(R(F)\) then its inverse image must appear in both \(F^{-1}(Y_1) = X_1\) and \(F^{-1}(Y_2) = X_2\) since its inverse image is unique.

Part (2). Similar to Part (1).

Put another way, if an element \(x\) is in both \(X\) sets then its image \(F\{x\}\) must be in both \(Y\) sets and vice versa. Similarly if an element \(x\) appears in \(X_1\) but not in \(X_2\) then \(F\{x\}\) must appear in \(Y_1\) but can not appear in \(Y_2\), and vice-versa.

Thus to deduce the value of \(F(2)\) and \(F(3)\) in the example above, we have by rule (1)
\[
F\{1, 4\} = F\{(1, 3, 4) \cap \{1, 2, 4\}\}
= \{3, 4, 5\} \cap \{2, 3, 4\}
= \{3, 4\}.
\]

Using this derived information and rule (2) we have:
\[
F\{3\} = F\{(1, 3, 4) - \{1, 4\}\}
= \{3, 4, 5\} - \{3, 4\}
= \{5\}.
\]
Using rule (2) again with the second sample and the derived information, we see that \( F(\{2\}) = \{2\} \).

These two rules when applied to manageably small sets, roughly model a limited form of human learning. Consider a security guard examining a sign-in sheet, and then glancing at the cars in the parking lot. Rule (1) corresponds to ‘if a person is here, his car should be in the lot, and vice-versa.’ Rule (2) corresponds to ‘if a person is not here, his car is not in the lot, and vice-versa.’ Thus after a while, he knows which cars belong to a few of the people.

6. OVERVIEW OF THE ALGORITHM

The algorithm presented in Fig. 1 begins (lines 1–3) with only the knowledge of the elements in \( D(F) \) and \( R(F) \), and the knowledge that there is an isomorphism between them. The main loop (lines 4–13) reads and processes samples until enough have been read to deduce the unknown function. Finally the function is printed (lines 14–15). The algorithm maintains all its current knowledge about \( F \) as the set of partitions \( P_1, \ldots, P_k \) and \( Q_1, \ldots, Q_k \) by ensuring that \( F(P_i) = Q_i \) for all \( i \). Thus the image for each element in \( P_i \) must be in \( Q_i \). As more samples are read, existing partitions are split into two smaller partitions, thus narrowing the set of possible images for a given element.

At termination, each partition contains exactly one element. Furthermore the partitions \( P_1, \ldots, P_k \) are mutually exclusive, as are the \( Q_1, \ldots, Q_k \) partitions. Thus each element of \( D(F) \) is in exactly one \( P_i \) at termination, and its image is in \( Q_i \). This allows the algorithm to output the unknown \( F \).

Algorithm LEARN

1. \( P_1 = \{1,2,\ldots,n\} \) Initialize first domain partition
2. \( Q_1 = \{1,2,\ldots,n\} \) Initialize first range partition
3. \( k = 1 \) Initialize number-of-partitions count
4. While \( k < n \) { While there are fewer than \( n \) partitions
5. \( \text{Read}(X,Y) \) Request next sample
6. \( \text{kk} = k \) Save current partition count for possible split
7. For \( i = 1, \text{kk} \) { Test each existing partition
8. If \( (P_i \cap X \neq \emptyset \) and \( P_i \subset X \) \) If partition \( P_i \) can be split
9. \( k = k + 1 \) Increase partition count
10. \( P_k = P_i \cap X \) Create new domain partition (apply rule (1))
11. \( P_i = P_i - P_k \) Remove duplicates from \( P_i \) (apply rule (2))
12. \( Q_k = Q_i \cap Y \) Create new range partition (apply rule (1))
13. \( Q_i = Q_i - Q_k \) Remove duplicates from \( Q_i \) (apply rule (2))
14. For \( i = 1, n \) { Output the discovered function and
15. \( \text{Output}(P_i, Q_i) \) Terminate

**FIGURE 1**
7. Operation of the Algorithm

After \( P_1 \) and \( Q_1 \) are initialized and the number of partitions, \( k \), is set to one, we enter a loop (lines 4–13) which is exited when the number of partitions reaches \( n \). At that point, by assumption, we are done since there are \( n \) \( P \) partitions and \( n \) \( Q \) partitions, each containing a single element, which completely describes \( F \).

So while the number of \( P \) and \( Q \) partitions is less than \( n \), the algorithm reads the next sample (line 5). If the sample read, \((X, Y)\), is capable of splitting an existing \( P \) partition, then it contains new information. It can split a \( P \) partition if its \( X\)-part contains some, but not all, of the elements in an existing \( P \) partition. This condition is tested in line 8.

If such a partition is found, then the next five lines (lines 9–13) simply increment the number-of-partitions count \( k \) (line 9) and split \( P_i \) into a new \( P_i \) and \( P_k \) (lines 10–11) and split \( Q_i \) into a new \( Q_i \) and \( Q_k \).

Notice that the new \( P_i \) and \( P_k \) have no elements in common since line 11 set the new \( P_i \) to contain only what was in the old \( P_i \) but not in \( P_k \). Similarly \( Q_i \) is split by lines 12–13.

To see that \( F(P_k) = Q_k \) and \( F(P_i) = Q_i \) still holds after splitting, assume \( F(P_i) = Q_i \) as an inductive hypothesis, since \( F(P_1) = Q_1 \) initially. By definition, \( F(X) = Y \) holds for each sample read. Thus

\[
\begin{align*}
P_k &= P_i \cap X \quad \text{line 10,} \\
F(P_k) &= F(P_i \cap X) \quad \text{substitution,} \\
&= F(P_i) \cap F(X) \quad \text{lemma 1 rule (1),} \\
&= Q_i \cap Y \quad \text{inductive hypothesis,} \\
&= Q_k \quad \text{line 12.}
\end{align*}
\]

Thus, \( F(P_k) = Q_k \). Let the new contents of \( P_i \) be represented as \( \hat{P}_i \), and the new contents of \( Q_i \) as \( \hat{Q}_i \). Then we have

\[
\begin{align*}
\hat{P}_i &= P_i - P_k \quad \text{line 11} \\
F(\hat{P}_i) &= F(P_i - P_k) \quad \text{substitution} \\
&= F(P_i) - F(P_k) \quad \text{lemma 1 rule (2)} \\
&= Q_i - Q_k \quad \text{inductive hypothesis and above proof} \\
&= \hat{Q}_i \quad \text{line 13.}
\end{align*}
\]

Since the single original \( P \) and \( Q \) partitions were \( P_1 = D(F) \) and \( Q_1 = R(F) \), and \( F(D(F)) = R(F) \) by definition, we see that \( F(P_i) = Q_i \) for all \( i \) by induction. The algorithm is illustrated by the example in Fig. 2.
OPTIMAL ISOMORPHISM INFERENCE

Thus $F=\{(1,7),(2,1),(3,6),(4,2),(5,5),(6,3),(7,4)\}$

* Indicates that the partition has split.

Fig. 2. Example of algorithm LEARN execution.

Assuming the elements in each sample and the initial $P_1$ and $Q_1$ are in sorted order, the set operations of intersection and difference can be performed in linear time. Since $kk$ (line 7) is at most $n$, the algorithm presented uses only $O(kn^2)$ time where $k$ is the number of samples read and $n=|D(F)|$, and $O(n^2)$ space. The execution speed was not optimized to allow a clearer presentation.

8. DATA EFFICIENCY

Gold coined the term "optimally data efficient" (Gold, 1967) to describe an inference algorithm which discovers the sought after structure (a grammar, function, Turing machine, etc.) in as few as, or fewer samples than any other possible inference algorithm.

He demonstrates that exhaustive search is one such optimally data efficient algorithm applicable when the search space can be enumerated, since no result which is consistent with the sequence of samples observed will be overlooked. A brute force enumeration algorithm chooses the "first"
structure in the search space and reads samples until it encounters a sample inconsistent with the currently chosen structure. The algorithm then chooses the "next" structure in the search space. It must now test this new structure on every sample already read to see if it is consistent with them.

A better algorithm chooses not the next structure in the search space, but rather the next structure consistent with all observed samples. An algorithm which extracts and efficiently represents the information contained in the samples read might be called a "knowledge preserving" inference algorithm, since it limits the choice of next structures to only those consistent with all samples thus far observed. Algorithm LEARN possesses this useful property as shown in

**Lemma 2.** Algorithm LEARN is knowledge preserving.

The algorithm does not save each sample. Instead, all the knowledge gained by observing samples is reflected in the set of $P$ and $Q$ partitions. After reading a sample $(X, Y)$, and before reading the next sample, partitions are split if necessary (lines 10–13) to ensure that each sample can be reconstructed by unioning some of the $P$ and corresponding $Q$ partitions. That is, the algorithm ensures before reading the next sample, that

$$X = \bigcup_{j \in I} P_j \quad \text{and} \quad Y = \bigcup_{j \in I} Q_j.$$  

Since all the elements of $X$ ($Y$) are already in one or more $P$ ($Q$) partition(s), then such an index set fails to exist only if there was some $P_i$ such that:

$$P_i \cap X \neq \{\} \quad \text{and} \quad P_i \cap X \neq Y.$$

That is, if $P_i$ contained some element not in $X$ as well as one or more which are in $X$. But this is exactly the condition tested for in line 8. If the condition is true, then $P_i$ and $Q_j$ are split so that the elements in both $P_i$ ($Q_j$) and $X$ ($Y$) are placed into another partition $P_k$ ($Q_k$). Only those elements in $P_i - X$ ($Q_j - Y$) are left in $P_i$ ($Q_j$). Thus the knowledge reflected in the $P$ and $Q$ partitions is sufficient to completely reproduce all the samples (and generally many more than) it has read.

**Theorem 1.** Algorithm LEARN is optimally data efficient.

**Proof.** We show that if algorithm LEARN has not learned $F$ after $r$ samples, there are at least two isomorphisms $F_1$, $F_2$ which are consistent with all samples observed. If algorithm LEARN did not terminate after $r$ samples, then the number of sets in the partition of $D(F)$ is less than $n$, since otherwise by line 4 the algorithm would terminate. Since we have $n$
elements and \( k < n \) partitions to put them in, we must have one partition \( P_j \) (say) with two or more elements, \( x_1, x_2 \). We know then that there are two distinct elements \( y_1, y_2 \) in \( Q_j \). But then by the knowledge preserving property, \( F_1(x_1) = y_1 \) and \( F_1(x_2) = y_2 \), or \( F_2(x_1) = y_2 \) and \( F_2(x_2) = y_1 \) are two distinct isomorphisms that are both consistent with all samples read.

9. THE MINIMUM NUMBER OF SAMPLES REQUIRED

**Theorem 2.** The minimum number of samples required to learn an isomorphism \( F \) is \( \lceil \log_2 (|D(F)|) \rceil \).

**Proof.** All elements of \( D(F) \) start out in \( P_1 \). When enough samples have occurred to split \( P_1 \) and all its descendents into singleton sets, the algorithm terminates. This is equivalent to saying that for complete learning, every element in \( D(F) \) must be in a different partition.

In the proof of Theorem 1, we see that if two elements \( x_1, x_2 \) are in the same partition, then if the \( X \)-part of the next sample contains both \( x_1 \) and \( x_2 \), or neither of them, then they both remain in the same partition. However if the next sample contains only one of them, then they are “split up” (placed into separate partitions).

Thus for termination (and thus complete learning), we need only assure that for each of the \( n \) \((n - 1)/2 \) pairs of elements in \( D(F) \), at least one sample occurred which would split them into separate partitions. Note that one sample generally splits many pairs of elements.

This property of a sample sequence is more easily viewed when the \( X \)-parts of the sample sequence \((X_1, Y_1), (X_2, Y_2), \ldots, (X_m, Y_m)\) are represented as an \( n \) by \( m \) binary matrix \( S_x \). Let \( S_x[i,j] = 1 \) if \( x_i \) is in \( X_j \) (the \( X \)-part of the \( j \)th sample) and \( S_x[i,j] = 0 \) if \( x_i \) is not in \( X_j \). Thus the \( X \)-parts of each sample form the COLUMNS of the matrix. The property of the sample sequence which ensures termination of the algorithm may be stated as:

\( F \) is completely learned when each ROW of \( S_x \) is different and \( m \geq 1 \).

To see that this is equivalent to the above requirement for complete learning, note that if two ROWS, \( S_x[i,*] \), \( S_x[j,*] \) are different, then there must be a COLUMN in which they differ. That COLUMN corresponds to the sample which splits up the two elements. For example if \( S_x[i,k] = 1 \) and \( S_x[j,k] = 0 \), then \( x_i \) is in \( X_k \), and \( x_j \) is not in \( X_k \). If each ROW is different, then for any pair of elements in \( D(F) \), we can find a sample (column) in the sequence (matrix) which would split them into separate partitions.

Since we require at least \( \lceil \log_2 (n) \rceil \) columns (binary digits) to represent
\( n \) different binary numbers, and since this number of columns (samples) learns \( F \), then \( \lceil \log_2(|D(F)|) \rceil \) is the minimum number of samples required.

10. Constructing a Minimum Learning Sequence

Using the terminology used in the proof of Theorem 2, a minimum (i.e., fastest) learning sequence is easily constructed. Express the numbers 0 through \( n-1 \) in binary and use one number for each row in the \( S_x \) matrix. Each row is then different from all the others. Then construct a sample from each column.

A sample is constructed from a column in the \( S_x \) matrix by using it as a selector from the domain set. This selected subset (choose element \( x_i \) from \( D(F) \) iff the corresponding column element is one) then becomes the \( X \)-part of a sample \((X_j, Y_j)\). The \( Y \)-part is constructed as \( Y_j = F(X_j) \). For example to construct a fasted (minimum) learning sequence for the function:

\[
F = \{(1, 4), (2, 5), (3, 9), (4, 1), (5, 13), (6, 8), (7, 6), (8, 16),
(9, 15), (10, 12), (11, 11), (12, 14), (13, 10), (14, 3), (15, 7), (16, 2)\}.
\]

We would have the following \( S_x \) matrix:

\[
\begin{array}{cccccccc}
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 \\
0 & 1 & 0 & 0 \\
0 & 1 & 0 & 1 \\
0 & 1 & 1 & 0 \\
0 & 1 & 1 & 1 \\
1 & 0 & 0 & 0 \\
1 & 0 & 0 & 1 \\
1 & 0 & 1 & 0 \\
1 & 0 & 1 & 1 \\
1 & 1 & 0 & 0 \\
1 & 1 & 0 & 1 \\
1 & 1 & 1 & 0 \\
1 & 1 & 1 & 1 \\
\end{array}
\]
OPTIMAL ISOMORPHISM INFERENCE

<table>
<thead>
<tr>
<th>j</th>
<th>X_j</th>
<th>Y_j</th>
<th>k</th>
<th>P and Q partitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>0</td>
<td>P = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16}</td>
</tr>
<tr>
<td>1</td>
<td>[0,9,10,11,12,13,14,15,16],(2,3,7,10,11,12,14,15)</td>
<td>1</td>
<td>Q = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15}</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>[0,9,10,11,12],(2,3,6,7,8,10,13,16)</td>
<td>2</td>
<td>P = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16}</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>(2,4,6,8,10,12,14,16),(1,2,5,7,9,11,14,16)</td>
<td>3</td>
<td>Q = {4,5}</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>(2,4,6,8,10,12,14,16), (1,2,3,5,8,12,14,16)</td>
<td>4</td>
<td>P = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16}</td>
<td></td>
</tr>
</tbody>
</table>

Thus F = {{1,4},{2,5},{3,9},{4,1},{5,12},{6,8},{7,6},{8,16},
{9,15},{10,12},{11,11},{12,14},{13,10},{14,4},{15,7},{16,2}}

* Indicates that the partition has split.

The sample execution for a minimum learning sequence is shown in Fig. 3. Notice that the domain element 1 never appears in the X-part of any sample, and consequently F(\{1\}) = \{4\} was never a subset of any Y-part. When D(F) and R(F) are known in advance, we are able to learn the mapping of one element “for free,” since we can determine it and its image by learning the other n-1 elements of D(F) and their images. When D(F) and R(F) are not known in advance, we can learn a function with |D(F)| = n in \[\lceil \log_2 (n + 1) \rceil\] samples.

When F is known to a “teacher,” then a minimum learning sequence may be thought of as an optimal teaching sequence. Such a sequence allows
an inference algorithm to learn the desired structure in as few samples as possible.

Notice that the $X$-part of a minimum learning sequence is independent of the unknown function. That is, the same sequence of $X$-parts will discover all isomorphisms of $n$ elements. Thus if $F$ is not explicitly known, but is available as an oracle, the $X$-parts of each sample can first be passed to the oracle as a "question" and the result used to construct a sample $(X, F(X))$ which is passed to the learning algorithm. This sequence of $X$-parts may then be thought of as an optimal investigative sequence since it will discover the unknown function in as few "questions" as possible.

11. THE AVERAGE NUMBER OF SAMPLES REQUIRED

Having seen that the minimum number of samples required to learn the unknown function is logarithmic, we seek the average number of samples required. A sample sequence learns $F$ if the corresponding $S_x$ matrix consists of $n = |D(F)|$ different binary numbers (see Theorem 2). Thus the probability that the unknown function has NOT been learned after $k$ samples is:

$$\Pr\{F \text{ not learned after } k \text{ samples} \} = 1 - \frac{2^k(2^k - 1) \cdots (2^k - n + 1)}{2^{nk}}.$$  

Here the numerator of the second term is the number of $n$ by $k$ $S_x$ matrices with $n$ different rows. The denominator is the total number of $n$ by $k$ binary matrices. Thus the average number of samples $L_n$, required for complete learning when $n = |D(F)|$ is

$$L_n = \sum_{k \geq 0} 1 - \frac{2^k(2^k - 1) \cdots (2^k - n + 1)}{2^{nk}}. \quad (3)$$

Equation (3) uses an alternative computational form for computing the average. Thus instead of multiplying the probability that it takes exactly $k$ samples to learn a function, times $k$ and summing over all $k$, we instead sum the probability that it takes more than $k$ samples to learn the function. This is effectively the same, since the probability it takes $k$ samples in the first sum becomes a portion of each term from 0 to $k-1$ of the second sum, and is thus added $k$ times to form the total.

The probability that it takes more than $k$ samples to learn a function is the same as the probability the function is not learned after the $k$th sample. We assume that each sample is equally likely. This is equivalent in this case to saying each element of the domain set is present (or absent) in any sample with probability one-half.
Equation (3) can be solved exactly for small $n$ by multiplying out the factors and summing the terms individually. See Table 1 for exact values for small $n$. While an exact closed form solution to Eq. (3) was not found, a close upper bound is not difficult.

**Theorem 3.** The average number of samples $L_n$ required for algorithm LEARN to completely learn an isomorphism with $n = |D(F)|$ is

$$
\lceil \log_2 (n) \rceil \leq L_n \leq \left\lfloor \log_2 \left( \frac{n(n-1)}{2} \right) \right\rfloor + 2.
$$

**Proof.** Let

$$T_n = \frac{2^k (2^k - 1) \cdots (2^k - n + 1)}{2^{nk}}.
$$

Note that $0 < T_n < 1$ and $T_{n+1} = T_n ((2^k - n)/2^k)$. Since $L_1 = 0$, we can approximate $L_n$ as follows:

$$L_n = \sum_{k \geq 0} 1 - T_n
$$

$$= \sum_{k \geq 0} \sum_{1 \leq j < n} (1 - T_{j+1}) - (1 - T_j)
$$

$$= \sum_{k \geq 0} \sum_{1 \leq j < n} T_j - T_{j+1}
$$

$$= \sum_{k \geq 0} \sum_{1 \leq j < n} T_j \left( 1 - \frac{(2^k - j)}{2^k} \right)
$$

$$= \sum_{k \geq 0} \sum_{1 \leq j < n} T_j \left( \frac{j}{2^k} \right)\]
\[
\sum_{k \geq 0} \sum_{1 \leq j < n} \frac{j}{2^k} \quad (\text{since } T_j \leq 1)
\]
\[
= \sum_{k \geq 0} \frac{1}{2^k} \sum_{1 \leq j < n} j
\]
\[
= \frac{n(n-1)}{2} \sum_{k \geq 0} \frac{1}{2^k}. \quad (4)
\]

Since \(1 - T_n \leq 1\) for all \(k\), (and will be exactly one for all \(0 \leq k < \lceil \log_2 (n) \rceil\) by Theorem 2) Eq. (4) can be split into the two sums:

\[
L_n \leq \sum_{0 \leq k < \lceil \log_2((n(n-1)/2)) \rceil} 1 + \frac{n(n-1)}{2} \sum_{k \geq \lceil \log_2((n(n-1)/2)) \rceil} \frac{1}{2^k}.
\]

The first sum is equal to \(\lceil \log_2((n(n-1)/2)) \rceil\) and the second term is bounded by a constant since

\[
\frac{n(n-1)}{2} \sum_{k \geq \lceil \log_2((n(n-1)/2)) \rceil} \frac{1}{2^k} = \frac{n(n-1)}{2} \left(2 - \frac{1}{2^{\lceil \log_2((n(n-1)/2)) \rceil}}\right)
\]
\[
= \frac{n(n-1)}{2} \left( \frac{2}{2^{\lceil \log_2((n(n-1)/2)) \rceil}} \right)
\]
\[
\leq \frac{n(n-1)}{2} \left( \frac{2}{n(n-1)} \right)
\]
\[
= 2.
\]

Thus,

\[
L_n \leq \left\lceil \log_2 \left( \frac{n(n-1)}{2} \right) \right\rceil + 2.
\]

Since the expected number can not be less than the minimum number \(\lceil \log_2 (n) \rceil\) proved in Theorem 2, the bounds follow.

The average is quite close to the upper bound as can be seen in Table 1.
There are results for the more general case of an arbitrary finite map from $D(F)$ to $R(F)$; the interested reader may consult (Flanagan, 1981).

12. CONCLUSIONS

An inference algorithm which learns isomorphisms is presented. The algorithm is shown to extract as much information as possible from each sample and is thus shown to be "optimally data efficient." The minimum number of samples required is shown to be $\lceil \log_2 (n) \rceil$, and a method for constructing a sequence of samples which allows the most rapid learning possible is demonstrated. This sequence can be thought of as either an optimal teaching sequence or an optimal investigative sequence. The average number of samples required when each sample is equally likely is shown to be approximately $2 \log(n)$.

The rapid rate of learning is primarily due to the efficient way that the remaining potential solution space can be represented. The representation structure in this case is both exact, since all and only solutions consistent with all previous knowledge (observations) are represented, and computationally inexpensive to update.

Finding efficient representations of knowledge for more general inference domains poses an interesting area for further research.

RECEIVED August 25, 1983; ACCEPTED November 1, 1985

REFERENCES


Feldman, J. (1972), Some decidability results on grammatical inference and complexity, Inform. Contr. 20, 244–262.


