An Efficient Algorithm and Monte Carlo Methods for Inferring Functional Dependencies

Ryan Marcus
ryan@rmarcus.info

Shaughan Lavine
shaughan@arizona.edu
Department of Philosophy
University of Arizona

May 26, 2014

Abstract

We present a formalization of functional dependencies that enables a greedy algorithm for set cover approximations to efficiently find functional dependencies within databases. This algorithm is augmented with Markov chain Monte Carlo methods. The result is an algorithm capable of performing fast automated data analysis on large databases.

1 Introduction

Finding functional dependencies within data can be useful for table decomposition and gaining insight into the data [3]. Unfortunately, inferring functional dependencies within data can be difficult [3]. Most algorithms use a sort-based or hyper-graph-based approach [6]. However, Akutsu et al. have developed a set-cover based approximation [1] algorithm for functional dependency inference that we expand upon (and formulate slightly differently).

The scale of many real-world databases indicates that Monte Carlo sampling techniques could be useful. In the past, such techniques have only been used to form functional dependency hypotheses that are then checked against the entirety of the data [6].

Here, we present a fast Markov chain Monte Carlo approach for probabilistic sampling and determining of functional dependencies that hold with high probability based in part on the `GREEDY` algorithm [1] presented by Akutsu et al.

Let databases be sets of tables, and let a table be a finite indexed set of rows. Rows in a table t are finite indexed sets with indexes contained in $C(t)$. The cardinality of any row in table t is equal to the number of columns (arity) of table t. The cardinality of t (denoted $|t|$) is the number of rows in table t. The function $C(t)$ gives the set $\{1, ..., n\}$ of column indexes of t, where n is the arity of t.

When t is a table, the value t_i is the ith row of t for $i = 1, ..., |t|$, and the value $t_{i,j}$ is the data in the ith row and jth column of the table t. When h is a set of column indexes such that $h \subseteq C(t)$, then t_{i_h} is a set containing $t_{i,j}$ for all $j \in h$.

The algorithm presented in this paper and by Akutsu et al. expresses the problem of detecting functional dependencies as a set cover problem and approximates a solution to that set cover problem using algorithm 1 [1]. We prove in the next section that this algorithm will find, for any column y of a table t, a set X of columns of t such that $\{y\}$ functionally depends on X and such that X is inclusion-minimal among such sets (that is, such that $\{y\}$ does not functionally depend on any proper subset of X), assuming that such an X exists. If no such X exists, the algorithm will properly report that $\{y\}$ does not functionally depend on any column in t besides itself. When such an X exists, the set X is a good approximation to a smallest set such that $\{y\}$ functionally depends on it. Then, in section 3, we give a simple Markov-chain based method of accelerating this algorithm.

2 Set cover approximation

2.1 Expressing functional dependencies

When $X \subseteq C(t)$ and $Y \subseteq C(t)$, we say that the columns of t indexed by Y are functionally dependent on the columns of t indexed by X if

$$\forall t_i, t_j \in t(t_{i_X} = t_{j_X} \rightarrow t_{i_Y} = t_{j_Y}). \tag{1}$$

This is equivalent to stating that some function $F : U \to V$ exists where U is the set $\{t_{i_X} \mid 0 < i < |t|\}$ and V is the set $\{t_{i_Y} \mid 0 < i < |t|\}$ such that $\forall t_i \in t(F(t_{i_X}) = t_{i_Y})$. This motivates the term functional dependency [3].

Equation 1 is equivalent to:

$$\forall t_i, t_j \in t(t_{i_Y} \neq t_{j_Y} \to t_{i_X} \neq t_{j_X}) \tag{2}$$

2.2 Finding functional dependencies with set covers

2.2.1 The Algorithm

The set cover problem is as follows: Given a universe U and a set S of subsets of U such that the union of the members of S equals (that is, *covers*) U, find a minimal-cardinality $\alpha \subseteq S$ such that the union of α equals (covers) U. Say that a set α is *inclusion minimal* if α covers U and no proper subset of α covers U. The set cover problem has been shown to be NP-complete [2]. However, a greedy approximation to the set cover problem can find, in polynomial time, at least one inclusion-minimal subset of S that covers U.

For a given U and S, the standard greedy approximation of the set cover problem is given in algorithm 1 [4]. It has been shown that this algorithm

Algorithm 1 The standard greedy approximation of the set cover problem

```
$\alpha \leftarrow \{\}$
while $|U| > 0$ do
    $\lambda \leftarrow$ the member of $S$ such that $\lambda \cap U$ is maximized
    $U \leftarrow U - \lambda$
    $\alpha \leftarrow \alpha \cup \{\lambda\}$
end while
```

achieves an approximation ratio of $H(|U|)$, where $H(n)$ is the nth harmonic number and $|U|$ is the cardinality of U [4]. Note that $|U| < |t|^2$.

To express the task of finding X given y as a set cover problem, let U be the indexed set of pairs (t_i, t_j), $i < j$, of rows of table t that have differing values in the column indexed by y. Let S_i be the set of pairs $(t_k, t_m) \subseteq U$, $k < m$, of rows of table t that have differing values in the column indexed by i, where $i \neq y$. Formally,

$$U = \{(t_i, t_j) \mid t_{i,y} \neq t_{j,y} \wedge 0 < i < j < |t|\} \tag{3}$$

$$\forall i \in (C(t) - \{y\}) S_i = \{(s,p) \mid (s,p) \in U \wedge s_i \neq p_i\} \tag{4}$$

Note that, while S is a set in the traditional phrasing of the set cover problem, this algorithm represents S as an indexed set.

Let α be, if one exists, an inclusion-minimal subset of S that covers U. Note that each member of α is equal to S_i for some i. Let the set β be the set of the indexes into S of each element of α, that is, $\forall i((S_i \in \alpha) \rightarrow (i \in \beta))$. In the next section we show that the union of all columns indexed by members of β determines $\{y\}$ (that is, a function F exists such that $F(t_{i_\beta}) = t_{i,y}$ for all i), or, if no cover exists (there is at least one member of U that is not a member of any S_i, and thus no α exists), then $\{y\}$ is not functionally dependent on any combination of columns other than itself. Since there may be many such inclusion-minimal subsets, this implementation will find only one such inclusion-minimal subset (this complication will be discussed later).

The algorithm presented here will find functional dependencies between a set of columns indexed by $X \subset C(t)$ and a set containing the single column indexed by y with $y \notin X$, if such a functional dependence exists. One possible implementation of this algorithm that finds a inclusion-minimal X such that $\{y\}$ is functionally dependent on X is given in algorithm 2. Note that if no functional dependence exists, this implementation returns the empty set.

2.2.2 Running time

Loops L1 and L2 take a maximum combined time of $O(r^2)$. L3 can run at most c times, because after all columns have been used, the algorithm terminates. L4 can run at most c times, because it is iterating over columns. L5 can run at most r^2 times, because U is bounded by r^2. Therefore, L3, L4, and L5 run in $O(c * c * r^2) = O(c^2 * r^2)$ time, where c is the number of columns and r is the number of rows.

2.2.3 Proofs

We can derive the formulation of the problem of finding functional dependencies as a set cover problem from equation 2. Define the function τ as:

$$\tau(X, y) = \{t_i \mid \forall j((t_{i_X} = t_{j_X}) \rightarrow (t_{i,y} = t_{j,y}))\} \tag{5}$$

Algorithm 2 A possible implementation

```
$U \leftarrow \{\}$
for  $i = 0 \rightarrow |t|$  do                          ▷ L1: Calculate the entire universe
    for  $j = i \rightarrow |t|$  do                      ▷ L2
        $d \leftarrow t_i$
        $f \leftarrow t_j$
        if  $d_y \neq f_y$  then                         ▷ Test if the rows differ
            $U \leftarrow U \cup \{(t_i, t_j)\}$          ▷ Add to universe
        end if
    end for
end for
$\alpha \leftarrow \{\}$
$Z \leftarrow C(t) - \{y\}$
while $|U| > 0$ do                                    ▷ L3
    $S \leftarrow []$                                 ▷ Construct S with only uncovered elements
    for  $x \in Z$ do                                 ▷ L4
        $\kappa \leftarrow \{\}$
        for  $(i, j) \in U$  do                       ▷ L5
            if  $i_x \neq j_x$  then
                $\kappa \leftarrow \kappa \cup \{i, j\}$
            end if
        end for
        $S$.append($\kappa$)
    end for
    if  $|S| = 0$  then return None                   ▷ No function exists
    end if
    next $\leftarrow max(S)$                          ▷ the highest-cardinality member of S
    $U \leftarrow U -$ {next}
    $Z \leftarrow Z -$ argmax($S$)                    ▷ where argmax is the index of the max
    $\alpha \leftarrow \alpha \cup$ argmax($S$)
end while
return $\alpha$
```

where t is a fixed table, $X \subset C(t)$, $y \in C(t)$, and $y \notin X$. In other words, $\tau(X, y)$ is the set of rows where X determines $\{y\}$. Note that if $\tau(X, y) \subseteq t$ and $\forall i(t_i \in \tau(X, y))$, then X determines $\{y\}$ in table t.

Proposition 1. *If $\tau(X, y) = r_1$ and $\tau(Z, y) = r_2$ then $\tau(X \cup Z, y) = r_1 \cup r_2$.*

Proof.

$$\begin{aligned}
\forall t_i, t_j \in r_1(t_{i,y} \neq t_{j,y} \rightarrow t_{iX} \neq t_{jX}) & \qquad \text{(By equation 2)} \\
\forall t_i, t_j \in r_2(t_{i,y} \neq t_{j,y} \rightarrow t_{iZ} \neq t_{jZ}) & \qquad \text{(By equation 2)} \\
\forall t_i, t_j \in r_1 \cup r_2(t_{i,y} \neq t_{j,y} \rightarrow t_{iX \cup Z} \neq t_{jX \cup Z}) & \\
\tau(X \cup Z, y) = r_1 \cup r_2 &
\end{aligned}$$

□

It is immediate from proposition 1 that, by induction:

Proposition 2. *$\bigcup\{\tau(x, y) \mid x \in X\} = \tau(X, y)$.*

Next, let $\tau^*(X, y)$ be the indexed set of all the elements of $\tau(X, y) \times \tau(X, y)$ in which the first element of each pair precedes the second element of each pair in the table. Note that $idx(u, v)$ is the index of u in the indexed set v, and $idx(u, v)$ is undefined if $u \notin v$.

$$\tau^*(X, y) = \{a \mid a \in (\tau(X, y) \times \tau(X, y)) \wedge idx(a_1, t) < idx(a_2, t)\} \tag{6}$$

Lemma 1. *A pair of rows is only in S_x if the pair of rows: do not match on column x, do not match on column y, and the first row comes before the second row in t.*

Proof. By equation 4, a pair of rows is only in S_x if the pair of rows do not match on column x and the pair of rows is in U.

By equation 3, a pair of rows is only in U if the rows do not match on column y and the first row comes before the second row in t. □

Proposition 3. *$S_x = \tau^*(\{x\}, y) \cap U$*

Proof. By equation 6, any row in a pair of rows in $\tau^*(\{x\}, y)$ must be in $\tau(\{x\}, y)$ and the first row of every pair must come before the second row in the table.

By equation 5 and 2, $\tau(\{x\}, y)$ contains every row such that if any two rows differ on column y, they must differ on column x.

By equation 3, all pairs of rows in U differ on column y. Thus, all pairs of rows in $\tau^*(\{x\}, y) \cap U$ differ on columns x and y.

Thus, $\tau^*(t, \{x\}, y) \cap U$ contains all pairs of rows for which the rows differ on columns x and y and the first row in each pair comes before the second row in the table.

By lemma 1, $S_x = \tau^*(\{x\}, y) \cap U$. □

From these propositions, we can prove that algorithm 2 computes functional dependencies as claimed.

Proof. After the initial construction of U, each iteration of `L3` will construct a set S such that $S_i \cup U \subseteq \tau^*(C(t)_i, y)$ (by proposition 3). We add members of $C(t)$ to α until $U = \emptyset$. Assuming the greedy approximation for set cover works as specified, When $U = \emptyset$, $\tau(\alpha, Y) = t$. We can add as many members of $C(t)$ to α as needed by proposition 2. Thus, the algorithm will either terminate (because it is unable to find a member of $C(t)$ such that $U_j \in \tau(t, x \in C(t), y)$ for some j or because $\tau(t, \alpha, y) = t$, in which case β determines $\{y\}$ in t.

□

Note that this demonstrates that finding a smallest set of columns that determines another column is `NP-hard`.

3 Markov chain

3.1 Construction

Since the set cover algorithm provided only finds one inclusion-minimal set of attributes that determine a single attribute, finding more complex functional dependencies requires additional computation. Additionally, since databases generally contain very large amounts of data, being able to sample from the database to perform faster calculations can be useful [6].

The following technique could utilize any function that discovers functional dependencies within a table.

Let T be a (sparse) matrix in which $T(A, b)$ with $A \subset C(t)$ and $b \in C(t)$ is the number of samples in which the columnset A determined $\{b\}$. Let

$S(t, n)$ be a sampling function that returns an indexed set of n random rows of t.

To build the Markov chain, we take several samples of t and look for multiple functional dependencies in each sample. We compile the results into T, and then use Markov chain analysis to determine a large number of functional dependencies. Given a sample size of n and a request to take N samples, our algorithm is shown in algorithm 3.

Algorithm 3 Sampling algorithm

```
for $i \to N$ do
    $s \leftarrow S(t, n)$
    for $c \in C(t)$ do
        $\kappa \leftarrow C(t) - \{c\}$
        $\kappa \leftarrow \kappa \cup \{\emptyset\}$
        for $d \in \kappa$ do
            $\alpha \leftarrow \texttt{cover}(s, \kappa - \{d\}, c)$
            $T(\alpha, c) \leftarrow T(\alpha, c) + 1$
        end for
    end for
end for
```

The order and permutations of columns tested is arbitrary, and many other possibilities likely exist. Good sampling functions ought capture a sufficient amount of initial data and not be biased. However, we find that this particular algorithm produces good results for our randomly generated datasets and datasets from various applications.

We will generate an absorbing Markov chain from our table T. Every set of columns will be a state (so there is one state for each member of $\mathcal{P}(C(t))$), and one absorbing state A will be created. We can define the transition probability P between any two transient states x and y as:

$$P(x, y) = \frac{T(x, y)}{N} \tag{7}$$

And define the transition probability between any transient state x to the absorbing state A as:

$$P(x, A) = \frac{[\sum \forall i(T(x, i))] - N}{N} \tag{8}$$

In order to show that P represents the transition probabilities of an absorbing Markov chain, we must show that P is normalized and that there is a path from every transient state to the absorbing state [5].

To show that P is normalized, we must show that

Proposition 4. $\forall i \sum \forall j P(i, j) = 1$

Proof. For any transient state r, $P(r, A) + \sum \forall j P(r, j) = 1$ because $1 - \sum \forall j P(r, j) = P(r, A)$. For A, $P(A, A) = 1$ by definition. $\square$

There is a path from every transient state to A as long as there was at least one sample in which one columnset did not determine one column. Otherwise, every sample taken had unique values in every column, indicating that the sample size needs to be increased, or that the table contains all unique values. Thus, P represents a proper absorbing Markov chain.

It is important here that `cover` returns minimal sets that determine columns. It is easy to see that $C(t)$ determines any member of $C(t)$ $(\tau(t, C(t), y \in C(t)) = t)$. By returning minimal subsets, we build our Markov chain out of more interesting data points, and thus the approximations produced will be more interesting.

3.2 Calculations

Using P, we can calculate the fundamental matrix F of our Markov chain. $F = (I - Q)^{-1}$, where Q is a state transition matrix for the transient members of P [5].

The fundamental matrix contains a probabilistic representation of the functional dependencies within a table t. It is difficult to determine the formula for the confidence of a given matrix, but given the nature of this type of analysis, confidence should be proportional to $n * N$.

In addition to the obvious usefulness of F, there are several interesting statistics that one can calculate. First, consider the normalized row sum of F. The row with the highest normalized row sum corresponds to the columnset that comes closest to determining all the other columns. This result can be useful for determining primary keys and for query optimization [3]. In the case where the set $C(t)$ has the highest normalized row sum, there is likely not enough sampling, or the data has no functional dependencies.

4 Conclusions

We have presented a fast algorithm for finding functional dependencies in data as well as mechanisms to apply Markov chain analysis to both generate additional results and process data in more manageable subsets (which can enable data mining under memory or time constraints). The set cover algorithm and its Monte Carlo counterpart can make useful additions to any data analysis toolbox.

References

[1] Tatsuya Akutsu, Satoru Miyano, and Satoru Kuhara. A simple greedy algorithm for finding functional relations: efficient implementation and average case analysis. *Theoretical Computer Science*, 292(2):481 – 495, 2003. Theoretical Aspects of Discovery Science.

[2] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to Algorithms.* McGraw-Hill Higher Education, 2nd edition, 2001.

[3] Ramez Elmasri and Shamkant B. Navathe. *Fundamentals of Database Systems, Fourth Edition.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[4] Olivier Goldschmidt, Dorit S. Hochbaum, and Gang Yu. A modified greedy heuristic for the set covering problem with improved worst case bound. *Inf. Process. Lett.*, 48(6):305–310, December 1993.

[5] C. M. Grinstead and J. L. Snell. *Introduction to Probability.* American Mathematical Society, Providence, 1997.

[6] Heikki Mannila and Kari-Jouko Raiha. Algorithms for inferring functional dependencies from relations. *Data Knowl. Eng.*, 12(1):83–99, February 1994.