Searching for Structure in Multiple Streams of Data

Tim Oates
Computer Science Department, LGRC
University of Massachusetts
Box 34610
Amherst, MA 01003-4610
oates@cs.umass.edu

Paul R. Cohen
Computer Science Department, LGRC
University of Massachusetts
Box 34610
Amherst, MA 01003-4610
cohen@cs.umass.edu

Abstract

Finding structure in multiple streams of data is an important problem. Consider the streams of data flowing from a robot's sensors, the monitors in an intensive care unit, or periodic measurements of various indicators of the health of the economy. There is clearly utility in determining how current and past values in those streams are related to future values. We formulate the problem of finding structure in multiple streams of categorical data as search over the space of dependencies, unexpectedly frequent or infrequent co-occurrences, between complex patterns of values that can appear in the streams. Based on that formulation, we develop the Multi-Stream Dependency Detection (MSDD) algorithm that performs an efficient systematic search over the space of all possible dependencies. Dependency strength is evaluated with a statistical measure of non-independence, and bounds that we derive for the value of that measure allow the search to be pruned. Due to the pruning, MSDD can find the k strongest dependencies in the streams by examining only a fraction of the search space.

1 Introduction

Automating the discovery of structure in multiple streams of data is an important problem. Consider the following examples: there seems to be a consensus among the business and research communities that existing commercial databases (many of which have a temporal component) can be mined to discover structure that might confer a competitive business advantage; an intelligent agent with the ability to discover structure in the values flowing from its sensors can automatically acquire and update a model of its environment (Oates & Cohen 1996); an understanding of how current and past states of a computer network are related to future states may allow proactive network management (Oates 1995). In this paper we cast the problem of finding structure in multiple streams of *categorical* data in terms of search, and we present and evaluate an efficient algorithm, the Multi-Stream Dependency Detection (MSDD) algorithm, based on that formulation.

MSDD finds *dependencies*, unexpectedly frequent or infrequent co-occurrences, between patterns of values that occur in multiple streams of categorical data. Dependencies can be expressed as *rules* of the following form: "If an instance of pattern x begins in the streams at time t, then an instance of pattern y will begin at time $t+\delta$ with probability p." The value of p is determined empirically by counting co-occurrences of x and y in the streams. A dependency is strong if p is very different from the probability of seeing a co-occurrence of x and y under the assumption that x and y are independent, and it is weak if p is roughly the same as that probability. Strong dependencies capture structure in the streams because they tell us that there is a relationship between their constituent patterns, that occurrences of those patterns are not independent.

MSDD finds the k strongest dependencies by performing a systematic search over the space of all possible dependencies. Children of search nodes are expanded in a manner that ensures that no node can ever be generated more than once (Oates, Gregory, & Cohen 1994; Riddle, Segal, & Etzioni 1994; Rymon 1992; Schlimmer 1993; Webb 1996). The structure of the search facilitates formulation of both domain independent (see Section 3) and domain dependent (Oates & Cohen 1996) heuristics that greatly increase efficiency. In particular, because MSDD returns a list of the k strongest dependencies, it is possible to use upper bounds on the values of a node's descendants to prune the search. If none of the descendants of node N

can have a value higher than that of any of the current k best nodes, then N can be pruned. We analytically derive an upper bound on the value of a statistical measure of dependency strength for the descendants of a node. Pruning based on optimistic estimates of the strength of unexplored dependencies allows MSDD to efficiently find the k strongest dependencies in an exponential space.

Our approach to rule induction from databases differs from others in that it does not require the user to specify a set of target concepts to serve as rule right-hand-sides. Most existing rule induction methods return rules that use the values of one or more domain variables (e.g. attribute values), appropriately combined, to characterize one of a small number of pre-specified target concepts (e.g. class labels). In contrast, MSDD explores the space of dependencies between pairs of arbitrary patterns of values; both the left- and right-hand-sides of rules can be complex expressions. MSDD finds structure missed by other rule induction algorithms that have less expressive representations or that limit their search to rules involving a user-specified set of target concepts.

In Section 2 we formally define the space of dependencies, and thus the expressiveness of the rules found by MSDD. We also describe the systematic search performed by MSDD over that space. Section 3 specifies a statistical measure of dependency strength, derives bounds for that measure, and shows how those bounds can be combined with MSDD's systematic search to implement a version of the algorithm that efficiently locates the k strongest dependencies in a set of streams. Section 4 presents the results of using MSDD to find complex dependencies in the multivariate time series of chest volume, heart rate and oxygen saturation taken from a patient suffering from sleep apnea, and among variables in datasets taken from the UC Irvine machine learning repository. We review related work in Section 5, and conclude and explore future directions for this work in Section 6.

2 The MSDD Algorithm

One of the things that sets MSDD apart from other rule induction algorithms is the expressive power of the rules (dependencies) that it finds. This section defines the space of dependencies that MSDD explores, describes how that space is systematically enumerated, and presents pseudo-code for the most basic version of the algorithm.

2.1 The Space of Possible Dependencies

MSDD accepts as input a set of *streams*, time series composed of categorical values. Streams are used to define the space of dependencies the algorithm will search and to evaluate the strength of dependencies. Individual streams could, for example, correspond to the discretized daily closing prices of particular stocks or to the discretized heart rate of a patient as recorded over the last hour by a monitor in an intensive care unit. The set of m input streams is denoted $\mathcal{S} = \{s_1, \ldots, s_m\}$, where the i^{th} stream is composed of categorical values taken from the set $\mathcal{V}_i$. We call these values *tokens*. All of the streams in $\mathcal{S}$ must have the same length, and we assume that all of the tokens occurring at a given position in the streams were recorded synchronously. Consider the following streams:

```
S1: D B B A D C D A B C
S2: X Y Y Z Y X Z Z X Y
S3: 2 1 3 2 2 1 2 3 2 1
```

Stream s_1 is composed of tokens drawn from the set $\mathcal{V}_1 = \{A, B, C, D\}$. Likewise, $\mathcal{V}_2 = \{X, Y, Z\}$ and $\mathcal{V}_3 = \{1, 2, 3\}$. All three streams have length 10. In this case, $\mathcal{S} = \{$ `(D B B A D C D A B C), (X Y Y Z Y X Z Z X Y), (2 1 3 2 2 1 2 3 2 1)`$\}$.

Recall that MSDD searches for dependencies expressed as rules of the following form: "If an instance of pattern x begins in the streams at time t, then an instance of pattern y will begin at time $t+\delta$ with probability p." Such rules are denoted $x \stackrel{\delta}{\Rightarrow} y$. We call x the *precursor* and y the *successor*. p is computed by counting the number of time steps on which an occurrence of the precursor is followed δ time steps later by an occurrence of the successor, and dividing by the total number of occurrences of the precursor. To keep the space of patterns and the space of dependencies finite, we consider patterns that span no more than a constant number of adjacent time steps. We allow precursors to span at most w_p time steps, and we allow successors to span at most w_s time steps. Both w_p and w_s are parameters of the MSDD algorithm.

We represent patterns of tokens (precursors and successors) as sets of 3-tuples of the form $\tau = (v, s, d)$. Each 3-tuple specifies a stream, s, a token value for that stream, v, and a temporal offset, d, relative to an arbitrary time t. Because such patterns can specify token values for multiple streams over multiple time steps, we call them *multitokens*.[1] Tuples that appear in precursors are drawn from the set $T_p = \{(v, s, d) | 1 \leq s \leq m, v \in \mathcal{V}_s, 0 \leq d < w_p\}$. Likewise, tuples that appear in successors are drawn from the set $T_s = \{(v, s, d) | 1 \leq s \leq m, v \in \mathcal{V}_s, 0 \leq d < w_s\}$. For example, the multitoken $x = \{(B, 1, 0), (Y, 2, 1)\}$ specifies a pattern that occurs twice in the streams above. For

[1] The definition of a multitoken given here is an extension of the one given in previous descriptions of the algorithm (Oates, Gregory, & Cohen 1994).

$t = 2$ and $t = 9$, we see token B in stream one at time $t+0$ and token Y in stream 2 at time $t+1$. Likewise, the multitoken $y = \{(X, 2, 0), (2, 3, 0), (Y, 2, 1), (1, 3, 1)\}$ occurs twice in the streams above, once at time $t = 1$ and again at time $t = 9$. The following streams are a copy of the streams above, except we have removed all tokens not involved in occurrences of x or y.

```
S1: B . . . . . . . B .
S2: X Y . . . . . . . X Y
S3: 2 1 . . . . . . . 2 1
```

Assuming m streams and that $|\mathcal{V}_i| = n$ for all i, the number of possible precursor patterns is $(n+1)^{mw_p}$. Precursor patterns can specify values for a block of tokens m streams tall and w_p time steps wide. For each of those mw_p positions, the pattern either specifies one of n token values or leaves the value unspecified (a total of $n+1$ alternatives), resulting in $(n+1)^{mw_p}$ possible patterns. Similarly, the number of possible successor patterns is $(n+1)^{mw_s}$.

Dependency rules ($x \stackrel{\delta}{\Rightarrow} y$) specify two multitokens (x and y) and a *lag* (δ). The lag, which is an input to the algorithm and is therefore the same for all dependencies, is used when counting co-occurrences of the two multitokens; if an occurrence of x is seen in the streams at time t, MSDD checks for an occurrence of y at time $t + \delta$. Therefore, the number of possible dependencies is given by $(n+1)^{mw_p} * (n+1)^{mw_s} = (n+1)^{m(w_p+w_s)}$. Even for relatively small sets of streams, the number of dependencies is potentially enormous. For $m = 10$, $n = 5$, $w_p = w_s = 3$, the space of dependencies contains more than 10^{45} elements.

Although the input parameters w_p, w_s and δ together with the streams in $\mathcal{S}$ define the space of precursors, successors and dependencies that MSDD will explore, any given dependency can be expressed for a large number of different settings of those parameters. This makes MSDD's ability to find structure in streams robust with respect to variations in the settings of w_p, w_s and δ. Figure 1 shows how one particular relationship between two multitokens can be captured by dependency rules for three different settings of w_p, w_s and δ. Unfortunately, it is also the case that any given dependency can often be expressed in different ways within the confines of a single specification of values for w_p, w_s and δ. For example, the patterns in the left-most two blocks in Figure 1 could be shifted right by one time step, resulting in a syntactically distinct yet semantically identical dependency. Therefore, during the search we prune dependencies that can be shifted in the manner just described, knowing that an unshiftable systactic variant exists elsewhere in the search space.

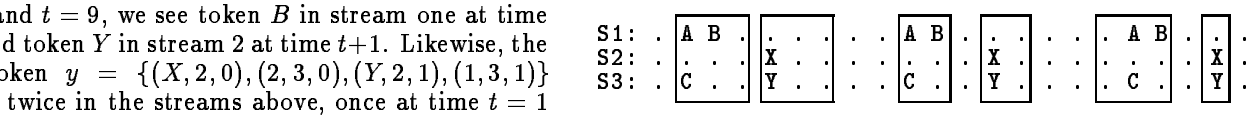

Figure 1: Any given relationship between two multitokens can be captured by dependencies for a wide variety of settings of w_p, w_s and δ. Therefore, MSDD's ability to find structure in streams is robust with respect to variations in those parameters. The figure shows one such relationship as captured by dependencies with the following values: $w_p = w_s = 3$, $\delta = 3$ (the left-most two blocks); $w_p = w_s = 2$, $\delta = 3$ (the middle two blocks); and $w_p = 3$, $w_s = 1$, $\delta = 4$ (the right-most two blocks).

2.2 Systematic Enumeration of Dependency Space

MSDD performs a general-to-specific search over the space of possible dependencies, starting with a root node that specifies no token values for either the precursor or the successor ($\{\} \Rightarrow \{\}$). The children of a node N are generated either by adding a single 3-tuple $\tau_p \in T_p$ to N's precursor, or by adding a single 3-tuple $\tau_s \in T_s$ to N's successor. However, no two tuples in the precursor (or the successor) can specify a value for the same stream on the same time step. That is, it is invalid to say that at time t stream s_i will contain both the value X and the value Y. For precursor x and for all τ_i, $\tau_j \in x$, it is not the case that $s_i = s_j$ and $d_i = d_j$ when $i \neq j$. Likewise for the successor. Note that for any node $x \Rightarrow y$ in the search tree at depth D, it is the case that $|x| + |y| = D$.

MSDD's search space has the property that the value of a node is independent of the path from the root to that node. For example, there are two possible paths to the dependency $\{(A, 1, 0)\} \Rightarrow \{(B, 5, 3)\}$ (movement down through the tree from one node to another is denoted $\rightarrow$): $\{\} \Rightarrow \{\} \rightarrow \{(A, 1, 0)\} \Rightarrow \{\} \rightarrow \{(A, 1, 0)\} \Rightarrow \{(B, 5, 3)\}$ and $\{\} \Rightarrow \{\} \rightarrow \{\} \Rightarrow \{(B, 5, 3)\} \rightarrow \{(A, 1, 0)\} \Rightarrow \{(B, 5, 3)\}$. However, the path taken does not affect the frequency of co-occurrence of the two multitokens in the streams, and therefore does not affect the strength of the dependency. Multitokens and dependencies express conjunctive concepts, and the order in which individual terms are added is irrelevant to the semantics of the final concept. Webb (Webb 1996) calls such search spaces *unordered*.

It is possible to enumerate all of the elements of an unordered search space systematically so that each element of the space is generated exactly once. This is accomplished by imposing an order on the search operators used to generate the children of a node, and applying only those operators at a node that are higher in the ordering than all other operators already applied

along the path to the node. For search spaces consisting of conjunctive concepts, operators add terms to an existing concept. Consider the space of conjunctive concepts defined over the literals $\{A, B, C, D\}$. If we order those literals such that $A < B < C < D$, then the corresponding search space can be enumerated systematically as shown in Figure 2. For the purpose of non-redundantly enumerating the space of concepts, any total ordering will work. However, some total orders may be better than others for other aspects of the search, such as maximizing the benefits of pruning. Note that the children of a node can be generated by considering only the literals in that node's concept and the ordered list of all literals; each node in the space is generated exactly once, without maintaining lists of open and closed nodes for the explicit purpose of checking for redundant generation. The ability to non-redundantly expand children by considering only information local to parent nodes has implications for the development of parallel and distributed systematic search algorithms, a topic that we explore in Section 6.

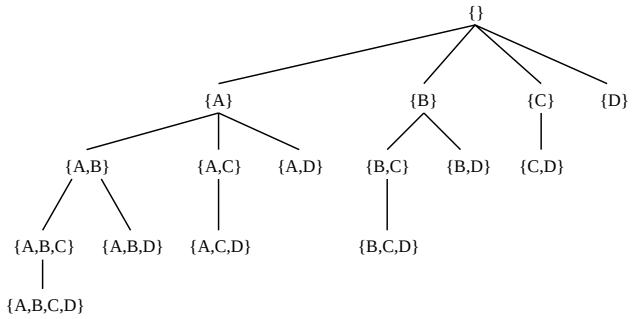

Figure 2: Systematic enumeration of the space of all semantically distinct conjunctive concepts composed of literals from the set $\{A, B, C, D\}$. Note that only one syntactic variant of each semantically distinct concept appears in the tree. For example, only one of $\{A, B\}$ and $\{B, A\}$ appears.

MSDD search operators either add a term from T_p to a node's precursor or add a term from T_s to a node's successor. To perform a systematic search over the space of possible dependencies between multitokens, we impose the following order on the terms in T_p and T_s: All of the terms in T_p are lower than all of the terms in T_s. For any $\tau_i, \tau_j \in T_p$, τ_i is lower than τ_j if $d_i < d_j$ or if $d_i = d_j$ and $s_i < s_j$. That is, terms in T_p are ordered first by their temporal offset, and then by their stream index. Likewise for terms in T_s. By ordering all of the terms in T_p below all of the terms in T_s we force precursors to be constructed before successors. As long as no terms have been added to a node's successor, terms may be added to its precursor. However, as soon as a single term is added to the successor, the precursor must thereafter remain unchanged because all of the search operators that would add new terms to the precursor are lower than the operator that added the term to the successor. This property of the search is used in Section 3 to derive bounds on possible values of a node's descendants.

A pseudo-code specification of MSDD is given below in Algorithm 1. MSDD requires six input parameters: a set of streams, $\mathcal{S}$, that is to be searched for structure; the maximum number of time steps that precursor and successor multitokens can span, w_p and w_s respectively; the lag to be used when counting co-occurrences of precursors and successors, δ; an evaluation function that determines the strength of a given dependency, f; and the number of dependency rules to return, k. As it is presented, MSDD would appear to be a rather unremarkable algorithm that maintains and eventually returns a list of the k strongest dependencies encountered while systematically and exhaustively exploring an exponential search space. However, in the next section we develop domain independent search heuristics that allow MSDD to prune the search space (i.e. we give a specification for the suggestively named REMOVE-PRUNABLE routine that is called in step 3.*c* of the algorithm), resulting in a highly efficient search for the k strongest dependencies in multiple streams of categorical data.

Algorithm 1 MSDD

MSDD($\mathcal{S}, w_p, w_s, \delta, f, k$)

1. *best* = ()
2. *open* = ($\{\} \Rightarrow \{\}$)
3. while not EMPTY(*open*) do
   a. *node* = NEXT-NODE(*open*)
   b. *children* = SYSTEMATIC-EXPAND(*node*, w_p, w_s)
   c. *children* = REMOVE-PRUNABLE(*children*)
   d. add *children* to *open*
   e. for *child* in *children* do
      i. if LENGTH(*best*) $< k$ or $\exists n \in$ *best* s.t. $f(child, \mathcal{S}, \delta) > f(n, \mathcal{S}, \delta)$ then add *child* to *best*
      ii. if LENGTH(*best*) $> k$ then remove from *best* the node with the lowest f value
4. return *best*

3 Search Heuristics

The result of MSDD's search for structure is a list of the k strongest dependencies found in the data, where dependency strength is measured by the user-supplied evaluation function f. The G statistic computed for 2x2 contingency tables is a statistical measure of non-independence, and is therefore an ideal candidate for f (Wickens 1989). Consider the contingency table be-

low that describes the frequency of co-occurrence of precursor x and successor y:

	y	$\overline{y}$
x	n_1	n_2
$\overline{x}$	n_3	n_4

The cell counts are obtained by counting occurrences of x and y in the streams; they indicate the number of times that x was followed by y (n_1), that x was not followed by y (n_2), that x did not precede y (n_3), and that neither x nor y occurred (n_4). The first row margin, $r_1 = n_1 + n_2$, is the number of times that x occurred, and the first column margin, $c_1 = n_1 + n_3$, is the number of times that y occurred. The table total, $T = \sum_{i=1}^{4} n_i$, is the total number of time steps over which events were counted.

G is computed for the table above as follows:

$$G = 2 \sum_{i=1}^{4} n_i log(n_i / \hat{n}_i)$$

$\hat{n}_i$ is the expected value of n_i under the assumption of independence, and is computed from the row margins and the table totals. For example, $\hat{n}_1 = p(x \wedge y)T = p(x)p(y)T = (r_1/T)(c_1/T)T = r_1 c_1/T$. Values of G near zero indicate that x and y are independent, whereas large values of G suggest non-independence. G is very similar to cross-entropy computed for the two discrete probability distributions (n_1/T, n_2/T, n_3/T, n_4/T) and ($\hat{n}_1/T$, $\hat{n}_2/T$, $\hat{n}_3/T$, $\hat{n}_4/T$). In fact, it is easy to show that for the case just described, the cross-entropy is given by $G/2T$. We prefer G to cross-entropy because G scores can be referred to a χ^2 distribution with one degree of freedom to determine the probability of making an error in rejecting the null hypothesis of independence. That is, G can be used to perform tests of statistical significance on individual dependencies.

Because MSDD returns a list of the k strongest dependencies, if we can derive an upper bound on the value of the evaluation function f for all of the descendants of a given node, then we can use that bound to prune the search. Suppose the function $fmax(N)$ returns a value such that no descendant of N can have an f value greater than $fmax(N)$. If at some point during the search we remove a node N from the open list for expansion, and $fmax(N)$ is less than the f value of all k nodes in the current list of best nodes, then we can prune N. There is no need to generate N's children because none of the descendants of N can have an f value higher than any of the current best nodes; none of N's descendants can be one of the k best nodes that will be returned by the search. (This type of pruning actually takes place at the time that children are generated, in line 3.c of Algorithm 1, to keep the size of the open list as small as possible.) The use of an optimistic bounding function is similar to the idea behind the $\hat{h}$ function in A* search. That is, if a goal node is found whose cost is less than underestimates of the total cost-to-goal ($g + \hat{h}$) of all other nodes currently under consideration, then that goal node must be optimal. Pruning based on optimistic estimates of the value of the descendants of a node has been used infrequently in rule induction algorithms, with ITRULE (Smyth & Goodman 1992) and OPUS (Webb 1996) being notable exceptions.

It is possible to derive a bound on the value of G for all of the descendants of any given node in MSDD's search space, allowing MSDD to perform optimistic pruning as described above. Recall that the children of a node are generated by adding a term either to the node's precursor or to its successor, yielding a more specific dependency. The result is that the more specific multitoken (the child's precursor or successor) may not match on some of the time steps on which the less specific multitoken in the parent matches. Suppose the child's precursor, x', is more specific than the parent's precursor, x, and consider time t on which an occurrence of x is followed by an occurrence of the successor. If x' does not match at time t because it is more specific than x, then what was an n_1 entry in the parent's contingency table becomes an n_3 entry in the child's. Following similar reasoning, it is easy to see how n_2 entries in the parent may become n_4 entries in the child (see Figure 3a). Likewise, when the child's successor is more specific, n_1 and n_3 entries in the parent may become n_2 and n_4 entries in the child, respectively (see Figure 3b).

	y	$\overline{y}$
x'	$n_1 - \Delta_1$	$n_2 - \Delta_2$
$\overline{x'}$	$n_3 + \Delta_1$	$n_4 + \Delta_2$

(a) More specific precursor

	y'	$\overline{y'}$
x	$n_1 - \Delta_1$	$n_2 + \Delta_1$
$\overline{x}$	$n_3 - \Delta_2$	$n_4 + \Delta_2$

(b) More specific successor

Figure 3: Given a parent dependency $x \Rightarrow y$ and its contingency table (n_1, n_2, n_3, n_4), the figure shows how the mass of the table may be redistributed for (a) a child dependency $x' \Rightarrow y$ with a more specific precursor, and (b) a child dependency $x \Rightarrow y'$ with a more specific successor.

The order imposed on MSDD's search operators for

the purpose of implementing a systematic search allows us to reason about how the mass of a node's contingency table might be redistributed in descendants of that node, and therefore to establish bounds on G. Recall that dependencies are built by first constructing their precursors, and then their successors. Given a node N with a non-empty successor (case (b) of Figure 3) whose contingency table is given by (n_1, n_2, n_3, n_4), any descendant, D, of N will have the same precursor and a more specific successor. In addition, D's contingency table will be of the form $(n_1 - \Delta_1, n_2 + \Delta_1, n_3 - \Delta_2, n_4 + \Delta_2)$. It can be shown that G is maximized for that table in one of two cases: $\Delta_1 = n_1$ and $\Delta_2 = 0$, or $\Delta_1 = 0$ and $\Delta_2 = n_3$. Therefore, for nodes with non-empty successors:

$$Gmax(n_1, n_2, n_3, n_4) =$$

$$max \left(\begin{array}{l} G(n_1, n_2, 0, n_3 + n_4) \\ G(0, n_1 + n_2, n_3, n_4) \end{array} \right)$$

Nodes with empty successors are more complicated because both the precursor and the successor of descendant nodes can be more specific than the original node. Entries in the contingency table can first migrate out of row one (case (a) of Figure 3), and then migrate out of column one (case (b) of Figure 3). It can be shown in this case that (the proof is omitted due to lack of space):

$$Gmax(n_1, n_2, n_3, n_4) =$$

$$max \left(\begin{array}{l} (1) \\ \quad \text{if } n_1 \leq n_2 + n_3 + n_4 \\ \quad\quad G(n_1, 0, 0, n_2 + n_3 + n_4) \\ \quad \text{else} \\ \quad\quad G(\frac{n_1+n_2+n_3+n_4}{2}, 0, 0, \frac{n_1+n_2+n_3+n_4}{2}) \\ \\ (2) \\ \quad \text{if } n_1 \geq abs(n_2 - n_3) \\ \quad\quad G(0, \frac{n_1+n_2+n_3}{2}, \frac{n_1+n_2+n_3}{2}, n_4) \\ \quad \text{else if } n_2 > n_3 \\ \quad\quad G(0, n_2, n_1 + n_3, n_4) \\ \quad \text{else} \\ \quad\quad G(0, n_1 + n_2, n_3, n_4) \end{array} \right)$$

Therefore, we can compute an upper bound on the value of G for all of the descendants of a node, whether that node has an empty or a non-empty successor. In the next section we describe the results of using G and $Gmax$ as outlined above to search for the strongest dependencies in several datasets.

4 Empirical Results

To evaluate MSDD's ability to find the k best rules in exponential spaces of dependencies, we ran the algorithm on several datasets. In all cases we used G as the measure of dependency strength, $Gmax$ as derived in Section 3 to prune nodes that cannot lead to "best" nodes, and breadth first search (i.e. the open list was treated as a queue in steps 3.a and 3.d of Algorithm 1).

To demonstrate the flexibility of the multitoken representation, we first applied MSDD to datasets taken from the UC Irvine machine learning repository. Those datasets are typically associated with algorithms that learn *classification rules*; rules that predict the value of a single domain variable by forming combinations of the values of the other $n - 1$ variables. However, MSDD can search for more complex structure in those datasets, finding rules that combine subsets of domain variables on *both* sides of the rules. Running MSDD with $w_p = w_s = 1$ and $\delta = 0$ eliminates the temporal component of dependencies, allowing the algorithm to find structure in a set of temporally independent vectors of categorical values (e.g. most of the datasets in the UC Irvine repository).

The results of running MSDD on the chess end-game dataset are summarized in Table 1 and Figure 4. The chess end-game dataset contained 500 instances of king/knight vs. king/rook chess end-games, where each instance contained 16 features that are typically used to predict whether a game is safe or lost for black (the king/knight pair). Table 1 presents the top $k = 10$ rules found by MSDD. Note that only two of the top ten rules are classification rules that predict a single value, the status of the game. Of the other eight rules, three specify values for multiple features on their right-hand-sides (including the two most highly ranked rules), and four do not include game status in either their left- or right-hand-sides (including the most highly ranked rule). Consider the rule with the highest G score, which says that when the distance from the black king to the white rook is two squares, then the distance from the black king to the black knight is one square and the distance from the white rook to the black knight is one square. This rule captures structure in the domain. The program used to generate instances for this dataset creates end-games in which the black king and the white rook are on the same row or column, separated by the black knight which keeps the king out of check. Therefore, if the king and rook are two squares apart (`bk-to-rk 2`), the knight must be in the intervening square, one square away from the king (`bk-to-kn 1`) and one square away from the rook (`rk-to-kn 1`).

The space of dependencies for the chess dataset with $w_p = w_s = 1$ and $\delta = 0$ contains more than 10^{17} elements. The number of nodes expanded by MSDD to find the $k = 10$ strongest dependencies in that space is summarized in Figure 4. The left-hand graph shows the number of nodes on the open list at each

Rule	Contingency Table	G
1. (bk-to-rk 2) ⇒ (bk-to-kn 1) (rk-to-kn 1)	(154 0 0 364)	617.4911
2. (bk-to-kn 3) ⇒ (game lost) (bk-to-rk 3)	(99 0 12 389)	421.57306
3. (bk-to-kn 3) ⇒ (game lost)	(99 0 17 384)	400.94736
4. (bk-to-rk 3) ⇒ (game safe)	(0 99 384 17)	400.94736
5. (bk-to-rk 2) ⇒ (game safe) (rk-to-kn 1)	(149 5 53 293)	334.18567
6. (game safe) (bk-to-wk 3) ⇒ (bk-to-kn 3)	(0 333 99 68)	271.89417
7. (rk-to-kn 1) ⇒ (bk-to-rk 3)	(113 154 233 0)	253.67134
8. (rk-to-kn 1) ⇒ (bk-to-rk 2)	(154 113 0 233)	253.67134
9. (game safe) (wk-to-kn 3) ⇒ (bk-to-kn 3)	(0 319 99 82)	248.29706
10. (bk-square open) ⇒ (kn-square open)	(279 0 104 117)	238.4587

Table 1: The top ten rules found by MSDD in the chess end-game dataset.

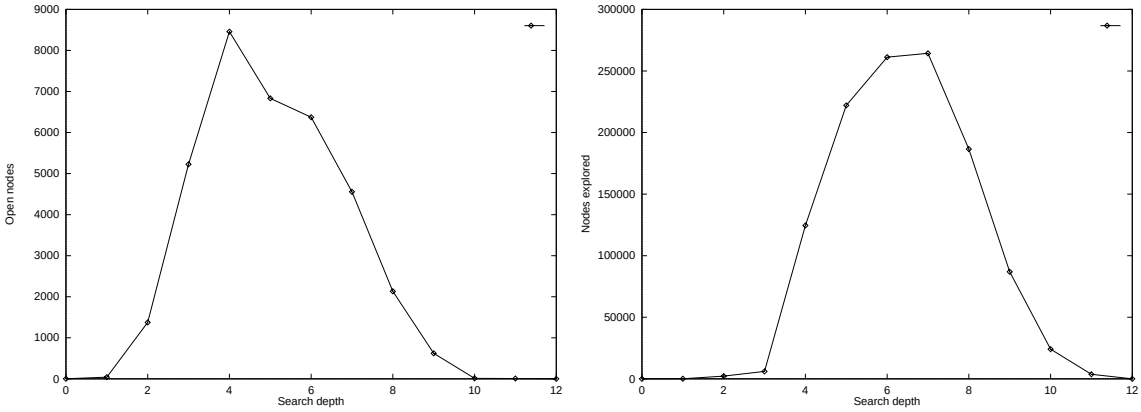

Figure 4: The number of nodes on the open list (the left-most graph) and the number of nodes expanded (the right-most graph) at each level of the search on the chess end-game dataset.

depth of the search, and the right-hand graph shows the number of nodes explored at each depth of the search. Clearly, MSDD explored a tiny fraction of the search space, yet is was able to find the ten best rules. Note that many more nodes are explored at any given depth than survive the pruning process to become open nodes at the next level. Pruning on $Gmax$ is highly effective. The number of nodes on the open list rises rapidly until depth four is reached, and thereafter declines sharply. Most of the ten best rules were found by MSDD at depth three, boosting the minimum G value in the "best" list and therefore making pruning more effective. The number of nodes explored at each level shows similar behavior, growing less rapidly and then sharply declining after depth four.

We applied MSDD to a number of other datasets from the machine learning repository, including the solar flares, auto imports, promoter and voting datasets. In more than one case, *all* of the most highly ranked rules contained values for multiple attributes on both sides of the rules. For example, the following is the most highly ranked rule in the auto imports dataset (numerical values were discretized into three bins: `high, med and low`):

```
(curb-weight low) (engine-type
overhead-cam) ⇒
(wheel-base low) (num-of-cylinders
four) (engine-size low) (bore low)
```

In another experiment, MSDD searched for the top 25 rules in 2000 time steps of the multivariate time series of chest volume (`cv`), heart rate (`hr`) and oxygen saturation (`os`) taken from a patient suffering from sleep apnea (Weigend & Gershenfeld 1993). MSDD found rules that used values in the streams over the last three time steps ($w_p = 3$) to predict values over the next two time steps ($w_s = 2, \delta = 1$). The continuous values in each stream were discretized into ten equal bins. An example of the rules returned by MSDD is shown below:

```
(9 os 0) (9 os 1) (9 os 2) ⇒ (4 cv 1)
(6 hr 1)
```

This rule states that when oxygen saturation holds steady at 9 for three consecutive time steps, then two time steps later chest volume will be 4 and heart rate

will be 6.

MSDD's ability to find strong dependencies between arbitrary patterns of token values is valuable in the automated discovery of structure in multivariate time series. First, it is easy to construct streams for which the dependencies $x \Rightarrow y_1$, $x \Rightarrow y_2$, ..., $x \Rightarrow y_k$ are all weak, and yet the dependency $x \Rightarrow y_1, \ldots, y_k$ has a very high G value. Algorithms that search for rules with restricted right-hand-sides may be unable to find such structure. Second, the fact that MSDD's rules have unrestricted right-hand-sides means that human users of the algorithm do not need to provide a set of target concepts. MSDD finds structure in the data, regardless of whether an accurate set of target concepts exists.

5 Related Work

The research reported in this paper grew out of the work of Howe and Cohen on finding dependencies between events in execution traces (a single stream of data) generated by the Phoenix planner (Howe & Cohen 1995). Dependencies between planner failures, failure recovery actions, and subsequent failures were combined with a weak model of the planner to automate analysis and debugging of recovery mechanisms. MSDD extends the dependency detection portion of that work by considering a significantly more expressive space of dependencies and by providing efficient mechanisms for finding the k strongest dependencies within that space.

Several systematic search algorithms have appeared in the literature (Oates, Gregory, & Cohen 1994; Riddle, Segal, & Etzioni 1994; Rymon 1992; Schlimmer 1993; Webb 1996), all of them variations on the basic idea of imposing an order on search operators, and applying only those operators at a node that are higher in the order than all other operators that have been applied on the path from the root to the node. Some of these cut off the search at an arbitrary depth to limit the size of the search space (e.g. (Riddle, Segal, & Etzioni 1994; Schlimmer 1993)). In contrast, MSDD returns a list of the k strongest dependencies that is equivalent to the list that would be returned by an algorithm that exhaustively searched the space of all possible dependencies. Our use of optimistic bounds on the value of the node evaluation function for pruning systematic search spaces is similar to the OPUS algorithm (Webb 1996), which in turn is a generalization of the same idea as applied to non-systematic search in the ITRULE induction algorithm (Smyth & Goodman 1992). MSDD and ITRULE return the k best rules, whereas OPUS returns a single goal node or the single node with the highest value.

Our approach to rule induction from databases differs from others, including all of those cited above that perform systematic search, in that it does not require the user to specify a set of target concepts to serve as rule right-hand-sides. Most existing rule induction methods return rules that use the values of one or more domain variables (e.g. attribute values), appropriately combined, to characterize one of a small number of pre-specified target concepts (e.g. class labels). Often, such algorithms must be run multiple times to learn rules for multiple concepts, once for each concept (Schlimmer 1993; Webb 1996), losing the benefit of pruning information generated during previous runs. The ITRULE algorithm is somewhat more general in that it simultaneously searches for rules whose right-hand-sides can specify the value of any single domain variable, not just the one containing the class label. In contrast, MSDD explores the space of dependencies between pairs of arbitrary patterns of values, looking for structure in the data regardless of where it exists. That is a considerable advantage when there is no *a priori* knowledge concerning probable relationships that exist in the streams, or when the "target concepts" themselves involve complex combinations of domain variables that may be difficult for a human to accurately express.

6 Conclusions and Future Directions

In this paper we formulated the process of finding structure in multiple streams of categorical data as systematic search over a space of dependencies. The MSDD algorithm performs an efficient search over such spaces to find the k strongest dependencies. The search is pruned by comparing an overestimate of the value of descendants of a node to the values of the current list of best nodes, pruning if the overestimate is lower than all of those values. We developed an upper bound on the value of G, a statisical measure of dependency strength, for the descendants of a node. MSDD successfully found the strongest dependencies in several of the datasets in the UC Irvine machine learning repository. The set of rules discovered in the chess endgame dataset were presented, along with an analysis of the amount of search required. In addition, sample rules were presented for the auto imports dataset and a mutivariate medical time series.

We will extend this research in several directions, including work on the basic MSDD algorithm and applications. In terms of the algorithm, we are currently implementing and testing a distributed version, called D-MSDD, which parallelizes the search for structure on multiple machines distributed across a network (Oates, Schmill, & Cohen 1996). The structure of the search makes it possible to non-redundantly expand nodes and to make pruning decisions based only on information that is available within individual nodes (with

one minor caveat). Therefore, nodes on the open list can be divided among different processes running on the same or different machines, requiring a minimum of communication to keep the load appropriately balanced and to maintain a collective list of the k best nodes found. We are also working on an incremental version (I-MSDD) of the basic algorithm. Although the algorithm as implemented is very efficient, recently published work of Webb (Webb 1996) describes an interesting technique for dynamically restructuring systematic search spaces so that most of the space is placed beneath nodes that have a high probability of being pruned. We will investigate extending that technique to apply to the space of dependencies between multitokens searched by MSDD. In terms of applications, work is proceeding in two areas. First, we are using MSDD to find structure in the interactions of an artificial agent with its environment for the purpose of learning planning operators (Oates & Cohen 1996). Precursor multitokens encode state/action pairs, and successor multitokens encode state changes. Strong dependencies capture state changes that the agent can reliably bring about. Second, we are using MSDD to learn how current and past states of computer networks are related to future states for the purpose of acquiring rules that will allow network managers to predict and avoid problems in their networks before they arise (Oates 1995).

Acknowledgements

This research was supported by ARPA/Rome Laboratory under contract numbers F30602-91-C-0076 and F30602-93-0100, and by a National Defense Science and Engineering Graduate Fellowship. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes not withstanding any copyright notation hereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements either expressed or implied, of the Advanced Research Projects Agency, Rome Laboratory or the U.S. Government.

References

Howe, A. E., and Cohen, P. R. 1995. Understanding planner behavior. *Artificial Intelligence* 76(1–2):125–166.

Oates, T., and Cohen, P. R. 1996. Searching for planning operators with context-dependent and probabilistic effects. To appear in *Proceedings of the Thirteenth National Conference on Artificial Intelligence.*

Oates, T.; Gregory, D. E.; and Cohen, P. R. 1994. Detecting complex dependencies in categorical data. In *Preliminary Papers of the Fifth International Workshop on Artificial Intelligence and Statistics*, 417–423. Does not contain work on incremental algorithm reported in book version.

Oates, T.; Schmill, M. D.; and Cohen, P. R. 1996. Parallel and distributed search for structure in multivariate time series. Technical Report 96-23, University of Massachusetts at Amherst, Computer Science Department.

Oates, T. 1995. Fault identification in computer networks: A review and a new approach. Technical Report 95-113, University of Massachusetts at Amherst, Computer Science Department.

Riddle, P.; Segal, R.; and Etzioni, O. 1994. Representation design and brute-force induction in a boeing manufacturing domain. *Applied Artificial Intelligence* 8:125–147.

Rymon, R. 1992. Search through systematic set enumeration. In *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning.*

Schlimmer, J. C. 1993. Efficiently inducing determinations: A complete and systematic search algorithm that uses optimal pruning. In *Proceedings of the Tenth International Conference on Machine Learning*, 284–290.

Smyth, P., and Goodman, R. M. 1992. An information theoretic approach to rule induction from databases. *IEEE Transactions on Knowledge and Data Engineering* 4(4):301–316.

Webb, G. I. 1996. OPUS: An efficient admissible algorithm for unordered search. *Journal of Artificial Intelligence Research* 3:45–83.

Weigend, A., and Gershenfeld, N. 1993. *Time Series Prediction: Forecasting the Future and Understanding the Past.* Addison-Wesley.

Wickens, T. D. 1989. *Multiway Contingency Tables Analysis for the Social Sciences.* Lawrence Erlbaum Associates.