Parallel and Distributed Search for Structure in Multivariate Time Series

Tim Oates, Matthew D. Schmill and Paul R. Cohen

Computer Science Department, LGRC
University of Massachusetts
Box 34610
Amherst, MA 01003-4610
{oates,schmill,cohen}@cs.umass.edu

Abstract. Efficient data mining algorithms are crucial for effective knowledge discovery. We present the Multi-Stream Dependency Detection (MSDD) data mining algorithm that performs a systematic search for structure in multivariate time series of categorical data. The systematicity of MSDD's search makes implementation of both parallel and distributed versions straightforward. Distributing the search for structure over multiple processors or networked machines makes mining of large numbers of databases or very large databases feasible. We present results showing that MSDD efficiently finds complex structure in multivariate time series, and that the distributed version finds the same structure in approximately $1/n$ of the time required by MSDD, where n is the number of machines across which the search is distributed.

1 Introduction

Knowledge discovery in databases (KDD) is an iterative process in which data is repeatedly transformed and analyzed to reveal hidden structure. The analysis portion of KDD, the actual search for structure in data, is called data mining. Efficient data mining algorithms are necessary when the number of databases to be mined is large, when the amount of data in a given database is large, or when many iterations of the transform/analyze cycle are required. The ease with which vast quantities of electronically available information can be generated and stored gives rise to the former two conditions. Parallel and distributed data mining algorithms can quickly mine large amounts of data by taking full advantage of existing hardware, both multiple processors on one machine and multiple machines on a network. Multi-Stream Dependency Detection (MSDD) is an easily parallelized data mining algorithm that performs an efficient systematic search for complex structure in multivariate time series of categorical data.

MSDD finds *dependencies* between patterns of values that occur in multivariate time series of categorical data. We call each univariate time series a *stream* of data. Dependencies are unexpectedly frequent or infrequent co-occurrences of patterns in the streams. MSDD finds the k strongest dependencies in a set of

streams by performing a *systematic* search over the space of all possible dependencies. Systematic search expands the children of search nodes in a manner that ensures that no node can ever be generated more than once [9–12, 14]. Because non-redundant expansion is achieved without access to large, rapidly changing data structures, such as lists of open and closed nodes, the search space can be divided between multiple processes on multiple machines. Only a small amount of inter-process communication is required to keep the list of the k strongest dependencies globally consistent.

Because MSDD returns a list of the k strongest dependencies, rather than all of the dependencies that it encounters during the search, it is possible to use upper bounds on the values of a node's descendants to prune. The expressiveness of MSDD's rule representation allows the algorithm to find complex structure in data, but also leads to an exponential search space, making effective pruning essential. We use the G statistic as a measure of dependency strength, and develop optimistic bounds on the value of G for the descendants of a node to prune.

The remainder of the paper is organized as follows: Section 2 discusses systematic search in detail. Section 3 presents the MSDD algorithm, defines the space of dependencies that the algorithm explores, and develops domain independent pruning techniques. Section 4 shows how the systematicity of MSDD's search can be exploited to develop parallel and distributed versions of the algorithm. Section 5 explores the ability of the core algorithm to find interesting and complex structure in multivariate time series, and compares the speed of the centralized (MSDD) and distributed (D-MSDD) versions of the algorithm. Section 6 reviews related work, and Section 7 concludes.

2 Systematic Search

MSDD's search for the k strongest dependencies in a set of streams is *systematic*, leading to search efficiency and parallelizability. Systematic search non-redundantly enumerates the elements of search spaces for which the value or semantics of any given node are independent of the path from the root to that node. Webb calls such search spaces *unordered* [14]. Consider the space of disjunctive concepts over the set of literals $\{A, B, C\}$. Given a root node containing the empty disjunct, *false*, and a set of search operators that add a single literal to a node's concept, a *non*-systematic elaboration of the search space contains (among other redundancies) six variants of a single concept – $A \vee B \vee C$, $A \vee C \vee B$, $B \vee A \vee C$, $B \vee C \vee A$, $C \vee A \vee B$ and $C \vee B \vee A$. Each variant is semantically the same as the other five, yet syntactically distinct. In the space of disjunctive concepts, the semantics of any node's concept is unaffected by the path taken from the root to that node. For example, the two paths below yield semantically identical leaf nodes:

$$false \rightarrow A \rightarrow A \vee B \rightarrow A \vee B \vee C$$
$$false \rightarrow C \rightarrow C \vee B \rightarrow C \vee B \vee A$$

Clearly, naive expansion of nodes in unordered search spaces leads to redundant generation and wasted computation.

Systematic search of unordered spaces generates no more than one syntactic form of each semantically distinct concept. That 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. Let op_A, op_B and op_C be the operators that add the literals A, B and C respectively to a node's concept. If we order those operators so that $op_A < op_B < op_C$, then the corresponding space of disjunctive concepts can be enumerated systematically so that each semantically distinct concept appears exactly once. For example, the concept A is obtained by applying operator op_A to the root node. Because $op_B > op_A$ and $op_C > op_A$, both op_B and op_C can be applied to the concept A, generating the child concepts $A \vee B$ and $A \vee C$. In contrast, the concept C, which is obtained by applying op_C to the root node, has no children. Because all other operators (op_A and op_B) are lower in the ordering than op_C, none will be applied and no children will be generated.

The fact that unordered search spaces can be explored without redundant node generation through systematic search is the key to parallelizing MSDD. Given any search node in the tree, the only information required to simultaneously generate that node's children and avoid redundant node generation is the operator ordering (e.g. $op_A < op_B < op_C$). For example, each of the subtrees rooted at the three children of the root node, A, B and C, could be expanded by systematic search algorithms running on different machines. The machine expanding node B would generate its children by applying all operators higher in the ordering than op_B, yielding the single child $B \vee C$ through the application of operator op_C. Because no operators are higher in the ordering than op_C, the node $B \vee C$ has no children, and the subtree rooted at B has been completely explored. Not only was no communication with the search algorithms running on the other machines required to expand that subtree, there was no need to know that they even existed or that other portions of the search space were being explored.

3 The MSDD Algorithm

MSDD accepts as input a set of streams that are used to define the space of dependencies the algorithm will search and to evaluate the strength of dependencies. The set of m input streams is denoted $\mathcal{S} = \{s_1, \ldots, s_m\}$, and the i^{th} stream is composed of categorical values, called *tokens*, taken from the set $\mathcal{V}_i$. 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.

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. Precursors can span at most w_p time steps, and successors can span at most w_s time steps. Both w_p and w_s are parameters of the MSDD algorithm.

Patterns of tokens (precursors and successors) are represented 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, they are called *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 $T_s = \{(v, s, d) | 1 \leq s \leq m, v \in \mathcal{V}_s, 0 \leq d < w_s\}$.

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 ($\{\} \Rightarrow \{\}$). 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.

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. The use of an optimistic bounding function is similar to the idea behind the h function in A* search. That is, if a goal node is found whose cost is less than underestimates of the cost-to-goal 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 [13], OPUS [14] and PROGOL [7] being notable exceptions.

In practice, we use the G statistic computed for 2x2 contingency tables to measure dependency strength, and we have derived bounds on the value of G for the descendants of a node, making it an ideal candidate for f. The interested reader is referred to [8] for more details.

[1] The definition of a multitoken given here is an extension of the one given in previous descriptions of the algorithm [9].

4 Parallel and Distributed MSDD

In the same way MSDD guarantees non-redundant generation of search nodes, MSDD guarantees that distinct nodes at the same depth in the search tree are parent to completely disjoint sets of children. The result is a search space that can be trivially partitioned into computationally independent subsets, and consequently MSDD is an algorithm well suited for parallel and distributed implemetation. We begin by discussing a parallel implementation of MSDD

The easy partitioning of MSDD's search space allows us to treat any intermediate search node as a root of a new search tree. Consider the goal of "basic" MSDD; search for elaborations on the completely general rule $\{\} \rightarrow \{\}$ that maximize the evaluation function f. A more general, parallelized approach is to search for elaborations on *an arbitrary rule* that maximize f. In this way, we treat each node as an "island", independent of anything else MSDD has learned, spawning a new thread to perform the search as if the node were the root.

An efficient parallel implementation of MSDD is possible because the search at any given node does not require access to previously elaborated search tree. The threads of P-MSDD need only non-exclusive read access to the time series and exclusive write access for insertions into the queue of k best nodes. Using a semaphore to provide exclusive writes to the k best list, the computation of MSDD can be effectively balanced over many processing elements.

4.1 Distributed MSDD

The implementation of parallel MSDD can be translated easily to an efficient distributed algorithm. This algorithm, D-MSDD, makes use of a client-server TCP tools to perform D-MSDD's search over a network of cooperating systems.

The D-MSDD algorithm begins with the *server*. The server is responsible for initiating the search, mediating communication, and declaring the search finished. Any number of *client* machines may contact the server to declare themselves as eligible for aiding in the search. This declaration process is called *registration*, where the server takes note of each client machine, issuing it a unique identifier for future communication. Once a desirable number of clients have registered, the server is ready to initiate the search process.

The distributed search proceeds on each participant machine according to a local *agenda*. Each machine's agenda is an independent partition of the unexplored MSDD search space. As with P-MSDD, the only shared strucutres are the list of k best dependencies and the dataset itself. Each machine participating in a D-MSDD search maintains local copies of these structures, keeping them synchronized through network message passing. We simulate the accessing of shared data by sending *best* messages to describe a candidate node for the k best list. We emulate the load balancing that goes on on a parallel machine by passing *node* messages that transfer nodes from an overloaded machine's agenda to a machine with a lighter agenda.

5 Empirical Results

In this section we compare the performance of MSDD and D-MSDD. For each of three datasets, the two algorithms found the $k = 20$ strongest dependencies. We ran D-MSDD on both two and three machines connected via a local area network. The datasets, which were all taken from the UC Irvine repository, included chess end-games, solar flares, and congressional voting records. The results are summarized below in Table 1. The table shows the number of nodes expanded, CPU cycles consumed, and the number of messages sent to keep the list of the k best dependencies consistent. When D-MSDD ran on two and three machines (the D-MSDD – 2 and D-MSDD – 3 cases respectively), the table contains the sum of the value over all machines participating in the search. Note that relatively few search nodes were required to find the 20 strongest dependencies in exponential spaces; pruning based on optimistic estimates of G is effective. Because the distributed search may be at different depths on different machines, the total number of nodes expanded may vary depending on when strong dependencies are found and used for subsequent pruning. However, in each case the total number of CPU cycles required to complete the search remains fairly constant, independent of the number of participating machines. Because load-balancing between the machines is fine grained, n machines can complete the search for structure roughly n times faster than one machine.

Dataset	Algorithm	Search Nodes	CPU Cycles	Messages
vote	MSDD	107,858	6,911,826	0
	D-MSDD – 2	124,234	7,915,858	7024
	D-MSDD – 3	115,375	7,963,435	6697
chess	MSDD	22,346	1,507,160	0
	D-MSDD – 2	27,073	1,573,309	1321
	D-MSDD – 3	31,955	1,793,743	2964
solar	MSDD	12,199	805,920	0
	D-MSDD – 2	13,544	906,188	457
	D-MSDD – 3	17,941	1,095,695	1,706

Table 1. Comparison of MSDD and D-MSDD on three dataset.

6 Related Work

Several systematic search algorithms have appeared in the literature [9–12, 14], 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.

Our use of optimistic bounds on the value of the node evaluation function for pruning systematic search spaces is similar to the OPUS algorithm [14], which in turn is a generalization of the same idea as applied to non-systematic search in the ITRULE induction algorithm [13]. MSDD and ITRULE return the k best rules, whereas OPUS returns a single goal node or the single node with the highest value.

Both parallel algorithms and consideration of data with a temporal component are rare in the KDD and data mining literature. Holsheimer and Kersten describe a system for inducing rules from large relational databases that performs a parallelized beam search over the space of possible rules and accesses the data through a parallel DBMS [5]. However, their system is limited to classification rules (a conjunct of literals predicting a single literal), and it can miss high quality rules due to the use of beam search. Aronis and Provost developed a parallel algorithm that builds new features from existing features in relational databases [2]. The newly constructed features are then passed to a standard (serial) inductive learning algorithm. While parallelism speeds the search for new features, it does not affect the speed with which rules using those features can be learned. Agrawal and Shafer [1] explore several parallel algorithms for mining association rules from very large databases, and Dehaspe and De Raedt [4] present a parallel implementation of the CLAUDIEN clausal discovery system. Berndt and Clifford describe a dynamic programming algorithm for finding recurring patterns in univariate time series [3], and Mannila *et al.* [6] developed an algorithm that finds frequently occurring episodes in event-based data (e.g. event logs generated by telecommunications networks).

7 Conclusion

In this paper we presented the MSDD data mining algorithm which performs a systematic search for structure in multivariate time series. MSDD discovers the k strongest dependencies between pairs of multitokens, arbitrary patterns of values that can span multiple streams and multiple time steps. MSDD prunes the search space with an upper bound on the value of the descendant of a given node, and we derived such a bound on the value of G. We recognized that systematic search over unordered spaces is easily parallelized, and developed D-MSDD, a distributed version of MSDD. MSDD is a powerful tool for discovering complex structure in very large databases due to the efficiency and expressiveness of the core algorithm and the ease with which the search for structure can be distributed over multiple machines on a network via D-MSDD.

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. We thank the anonymous reviews for helpful suggestions.

References

1. R. Agrawal and J. C. Shafer. Parallel mining of association rules: Design, implementation and experience. Technical Report RJ 10004, IBM, 1996.
2. John M. Aronis and Foster J. Provost. Efficiently constructing relational features from background knowledge for inductive machine learning. In *Working Notes of the Knowledge Discovery in Databases Workshop*, pages 347–358, 1994.
3. Donald J. Berndt and James Clifford. Using dynamic time warping to find patterns in time series. In *Working Notes of the Knowledge Discovery in Databases Workshop*, pages 359–370, 1994.
4. Luc Dehaspe and Luc De Raedt. Parallel inductive logic programming. In *Proceedings of the MLnet Familiarization Workshop on Statistics, Machine Learning and Knowledge Discovery in Databases*, 1995.
5. Marcel Holsheimer and Martin L. Kersten. Architectural support for data mining. In *Working Notes of the Knowledge Discovery in Databases Workshop*, pages 217–228, 1994.
6. Heikki Mannila, Hannu Toivonen, and A. Inkeri Verkamo. Discovering frequent episodes in sequences. In *Proceedings of the First International Conference on Knowledge Discovery and Data Mining*, pages 210–215, 1995.
7. S. Muggleton. Inverse entailment and progol. *New Generation Computing*, 13:245–286, 1995.
8. Tim Oates and Paul R. Cohen. Searching for structure in multiple streams of data. In *Proceedings of the Thirteenth International Conference on Machine Learning*, pages 346 – 354, 1996.
9. Tim Oates, Dawn E. Gregory, and Paul R. Cohen. Detecting complex dependencies in categorical data. In *Preliminary Papers of the Fifth International Workshop on Artificial Intelligence and Statistics*, pages 417–423, 1994.
10. Patricia Riddle, Richard Segal, and Oren Etzioni. Representation design and brute-force induction in a boeing manufacturing domain. *Applied Artificial Intelligence*, 8:125–147, 1994.
11. Ron Rymon. Search through systematic set enumeration. In *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning*, 1992.
12. Jeffrey C. Schlimmer. Efficiently inducing determinations: A complete and systematic search algorithm that uses optimal pruning. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 284–290, 1993.
13. Padhraic Smyth and Rodney M. Goodman. An information theoretic approach to rule induction from databases. *IEEE Transactions on Knowledge and Data Engineering*, 4(4):301–316, 1992.
14. Geoffrey I. Webb. OPUS: An efficient admissible algorithm for unordered search. *Journal of Artificial Intelligence Research*, 3:45–83, 1996.