A Fast Algorithm for Computing Hypergraph Transversals and its Application in Mining Emerging Patterns

James Bailey, Thomas Manoukian and Kotagiri Ramamohanarao
Department of Computer Science & Software Engineering
The University of Melbourne, Australia
{jbailey,tcm,rao}@cs.mu.oz.au

Abstract

Computing the minimal transversals of a hypergraph is an important problem in computer science that has significant applications in data mining. In this paper, we present a new algorithm for computing hypergraph transversals and highlight their close connection to an important class of patterns known as emerging patterns. We evaluate our technique on a number of large datasets and show that it outperforms previous approaches by a factor of 9-29 times.

Introduction Analysis of hypergraphs is an important problem in discrete mathematics which has many applications in computer science. Example areas range from minimal diagnosis and propositional circumscription [10], to learning boolean formulae [2], and boolean switching theory [4].

The hypergraph minimal transversal problem is particularly significant from a data mining perspective. Indeed, the algorithmic complexity of mining maximal frequent itemsets and minimal infrequent itemsets is closely linked to the complexity of computing minimal hypergraph transversals [6].

Minimal infrequent itemsets are minimal itemsets such that their frequency is less than α in the dataset being mined (where α is some threshold $\geq$ 1). They are interesting because of their connection to *contrast* or *emerging patterns*. Consider the transactions for class A: $\{\{b,d,g,j\},\{c,e,g,j\},\{c,f,h,j\},\{a,f,h,j\}\}$ and class B: $\{\{a,d,g,j\},\{a,d,g,i\},\{c,f,h,i\},\{a,e,g,j\}\}$. Suppose we are interested in finding minimal contrasts between them. The transactions in Class A contain the following contrast patterns: Transaction 1 $\{b\}$, Transaction 2 $\{ce,cg,cj\}$, Transaction 3 $\{cj,fj,hj\}$, Transaction 4 $\{af,ah,fj,hj\}$.

The relationship to hypergraphs is natural. These are defined by a set of vertices $V = \{v_1, v_2, ..., v_n\}$ and a set of edges E, where each edge is some subset of V. A *transversal* of a hypergraph is any set of vertices that contains at least one element of every edge. A *minimal transversal* is a transversal such that no proper subset is also a transversal.

Each transaction t in Class A can be thought of as inducing a hypergraph with respect to all the transactions in Class B. The vertex set corresponds to the elements of t. Hypergraph edges are individually defined by subtracting a transaction in the negative dataset from the vertex set. So, for $t = 3 = \{c,f,h,j\}$ in class A, we have a hypergraph with vertex set $\{c,f,h,j\}$ and edges: Hyperedge 1 $\{c,f,h\}$, Hyperedge 2 $\{c,f,h,j\}$, Hyperedge 3 $\{j\}$, Hyperedge 4 $\{c,f,h\}$. The three minimal transversals of this hypergraph are $\{cj,fj,hj\}$, these correspond precisely to the contrast patterns listed above for transaction 3 in class A.

Efficiently finding minimal hypergraph transversals (and thus mining minimal infrequent itemsets and emerging patterns) is a computationally difficult task. Hypergraphs with many vertices (high dataset dimensionality) and many edges (high dataset volume) can contain huge numbers of patterns - an exponential number in the worst case.

In this paper, we investigate the problem of efficiently finding minimal hypergraph transversals in the data mining context. Our main contributions are threefold:

- We identify the correspondence between computing minimal hypergraph transversals and data mining of emerging patterns. This connection has not, to our knowledge, been made elsewhere.
- We present a new algorithm for efficiently computing minimal transversals based on a guided partitioning heuristic.
- We discuss experiments performed on several large datasets that demonstrate our algorithm significantly outperforms previous approaches.

Hypergraphs The study of hypergraphs [1] and their related problems is a well established field of discrete mathematics. In this section we overview three known algorithms for computing minimal transversals. See [4, 5] for a formal presentation.

Work by Berge [1] proposed a cross-product based algorithm for solving the problem. The method involves iteratively finding partial minimal transversals of the input hypergraph until the entire minimal transversal set is computed. For each edge, e_i, we calculate the cross-product between e_i and the minimal transversal set calculated so far. The result of this cross product operation is then minimised.

The precise complexity of hypergraph transversal is an open problem. Work in [5] presented an algorithm that solves an equivalent problem, monotone Boolean duality and has the best known complexity. *Algorithm B* was presented to solve this problem in $\mathcal{O}(nm) + m^{\log m}$ time where n is the number of variables and m is the combined size of the input and output.

Work in [7] identified two primary drawbacks in the method of Berge; a large amount of memory is required to store partial transversals, no transversal is available for output until all have been found. The basis of their algorithm is to traverse, depth first, a tree of transversals. This feature means both issues are addressed.

Emerging Patterns *Emerging Patterns*, or EPs, were introduced in [3] as a means of contrasting disjoint sets of relational data. EPs are simply defined as itemsets whose frequency of occurrence, or support, differs substantially between two sets of data. They have been shown to be a powerful tool in the field of classification [8]. An important particular case of EPs, *Jumping Emerging Patterns*, or JEPs, refer to those itemsets that exist in one dataset and are totally absent from the other.

In the work that introduced them, the process of finding all EPs satisfying some support constraints relied on a function *Border-Diff*, which discovers unique minimal subsets that exist within one set when contrasted against a group of other sets.

The *Border-Diff* operator [3] is reminiscent of the algorithm of Berge, also using the strategy of generating partial results, in this case, partial EPs. No connection to hypergraphs was made in [3] though.

This operator has two input parameters: a reference set of itemsets, or negative instances (transactions), N and a single positive instance (transaction), p. The process initially involves finding all subsets of p that do not exist in at least one element of N. This is simply the set of differences *Diffs* $= \{p - n_i, \forall\, n_i \in N\}$. Taking these differences, the generation based on partial expansions of the cross-product follows in a similar manner as the Berge algorithm, with the set of differences taking the place of the set of edges found in a hypergraph.

The principal difference is a mechanism to avoid the generation of non-minimal itemsets. One observation lies in the fact that given a set of itemsets comprising the partial expansion up to some iteration j, PE_j, on processing the difference in iteration $j+1$, d_{j+1}, we need not consider those elements $pe \in PE_j$ for which $pe \cap d_{j+1} \neq \emptyset$.

An additional optimisation, further to the scheme of [1], centers on the initial set of differences. Due to the incremental nature of the process, The order in which one iterates through the differences can be significant. By processing them in increasing order of cardinality, the search space can be reduced.

It is also possible to compute minimal hypergraph transversals in a bottom-up, levelwise manner. This connection has been pointed out in [6, 9]. Work in [11] presents algorithms which are based on this idea. Here, itemsets are grown, one item at a time, until a transversal is found. A set enumeration tree can be used to guide the growth process. The major problem with this strategy is that minimal transversals do not possess the monotonic property that could allow the well-known a-priori optimisation to be used (cease growing an itemset once it isn't a transversal). Instead, they possess an anti-monotonic property (all supersets of a minimal transversal are themselves transversals). The amount of pruning that can be done when growing bottom-up is thus vastly reduced and consequently the method is not competitive with respect to the other schemes when the dimensionality becomes large.

A New Partitioning Algorithm While the methods of [3] and [7] both improve on the algorithm of Berge, all three are susceptible to the fact that when the number of items or vertices is large, potential worst-case exponential behaviour becomes a practical problem, the size of partial expansions grows very quickly and thus the task of keeping intermediate transversals minimal becomes a bottleneck.

Our system, is fundamentally designed to prevent situations in which prohibitively large (with respect cardinality) edges have their cross-product computed. We achieve this by a divide and conquer approach, which iterates through specific subsets of edges. The key is the manner in which we identify candidate subsets of edges such that cross-product expansion/minimisation is more tightly controlled. The process of partitioning is recursive.

Partitioning any edge set is based on the frequency of the vertices that comprise them. All partitions are induced by a particular vertex. Given some vertex, v a partition of edges is simply those which do not contain v. This first step ensures that the number of edges in any induced partition is smaller or equal to the set of edges from which it was projected. So by iterating through each vertex, we induce some partition. Our method of avoiding an expensive round of expansion/minimisation is to manufacture one of two kinds of partitions: i) those having many edges, where each has relatively low cardinality, or ii) those having few edges, when the cardinality of individual edges is high. This objective requires that edges in any partition must have their cardinality reduced. This is done by modifying each edge in any partition by projecting out only those vertices previously considered in prior iterations. The choice of a total vertex ordering is crucial, due to its effect on cardinality reduction of the edges.

Clearly, by selecting the least frequent vertex in the edges, we project a significant number of them (those not containing this vertex). By masking out those vertices that succeed the least frequent vertex from the partition, we generate the emptyset, a base example of our preferred partition i). Conversely when the last vertex is considered, no vertices remain to be masked out, however, the least number of edges are projected, a case of scenario ii). By forming such partitions, we have control over what we choose to input to a subroutine which generates minimal transversals. Here, we utilise a version of Berge with optimisations attributable to Dong and Li [3] as the subroutine in our algorithm. Our Partition algorithm utilises a global variable (*Min_Tvs*) to store the final set of minimal transversals. Determining whether to apply the algorithm recursively is based on whether the partition is still ‘too large’. To measure this, we use a simple metric based on i) the number of edges in a partition (*par_num_edges*) and ii) the average edge cardinality multiplied by the number of edges (*par_volume*). The function Add_Min_Can combines candidates (*Can_Tvs*) which are minimal for the partition (but possibly not globally), with the global set of transversals so far *Min_Tvs* . As all transversals are computed with respect some partition, the relevant vertices (*Par_Ver*) must be appended to each output of the *Optimised_Berge* subroutine. The algorithm is invoked by the top-level call Partition(edges,$\emptyset$) and *Min_Tvs* is initially $\emptyset$.

Algorithm 1 Partition(E, Par_Ver)

```
Input: A set of hypergraph edges, E
       A set of partitioned on vertices, Par_Ver
ver_universe = set of all vertices in hypergraph
Generate frequency table for all vertices
Order vertices in increasing frequency ⇒ [v_1, ..., v_k]
for v = v_1 to v_k do
  Partition = ∅
  ver_universe = ver_universe − v
  for all edge ∈ E do
    if v ∉ edge then
      Partition = Partition ∪ (edge − ver_universe)
    end if
  end for
  Par_Ver = Par_ver ∪ v
  if par_num_edges ≥ 2 AND par_volume ≥ 50 then
    Partition(Partition, Par_Ver)
  else
    Can_Tvs = Optimised_Berge(Partition)
    Add_Min_Can(Can_Tvs, Min_Tvs, Par_Ver)
  end if
  Par_Ver = Par_ver − v
end for
```

Experimental Results

We now experimentally study the algorithms discussed in the previous section. We examine three large datasets from the UCI Machine Learning Repository [1] - Waveform, Satimage and Splice. Statistics for these are shown in table 1. All experiments were carried out on a 500MHz Pentium III PC, with 512MB of memory, running Linux(Mandrake 8.2). All times are measured in seconds.

The overall outline of each experiment is as follows: A single (positive) transaction p is chosen from the first class within a dataset. The entire set of negative transactions corresponds to all the transactions in the other classes. m of these transactions are then randomly chosen to yield a smaller negative set N. Together, p and N induce a set of vertices and hypergraph edges, as explained earlier. The time taken to calculate the minimal transversals of this hypergraph is measured. This time is then averaged over k different choices for p, where k is 100. The procedure is then repeated for different choices of m. The results for all classes were very similar.

The second column of each table shows the average number of transactions (from classes other than the one being considered) from which hypergraph edges were generated. We ensured we were dealing with simple hypergraphs (ones with no non-minimal edges) by performing an initial minimisation step and thus the the value in parentheses is the actual number of edges used as input. The third column shows the average number of transversals generated per problem instance.

We considered the collection of algorithms previously mentioned. Aside from our algorithm (Partition), we chose *Border-Diff* [3] and two hypergraph transversal algorithms, *Algorithm B* of Fredman-Khachiyan [5] and Kavvadias-Stavropoulos [7]. We implemented *Border-Diff*, the Fredman-Khachiyan algorithm and a Level-Wise system. We were able to obtain an executable of the Kavvadias-Stavropoulos algorithm. Preliminary work (results excluded due to lack of space) showed that both the Fredman-Khachiyan algorithm and the Level-Wise approach were uncompetitive. For Fredman-Khachiyan this is due to the conservative manner in which it decomposes prime implicants, splitting on a single variable per recursive call. Its complexity (in terms of the number of recursive calls made) is a function of the size of the input and output and its cost on the modest Waveform dataset was prohibitive (~1600 times slower than Partition and ~92 times slower than

[1] www.ics.uci.edu/~mlearn/MLRepository.html

Dataset	#classes	#attrs	#vertices	#transactions
Waveform	3	21	105	5000
Satimage	6	36	180	6435
Splice	3	60	287	3175

Table 1. DATASET CHARACTERISTICS

CLASS	N (EDGES)	TVS/CALL	AVE TIME/CALL		RATIO
WAVEFORM DATASET					
			PART	B-DIFF	
0	3343(562)	4988	0.44	5.35	12.16
SATIMAGE DATASET					
CLASS	N (EDGES)	TVS/CALL	AVE TIME/CALL		RATIO
			PART	B-DIFF	
1	1000(163)	5128	0.40	10.31	25.78
	1250(186)	6021	0.49	14.12	28.82
SPLICE DATASET					
CLASS	N (EDGES)	TVS/CALL	AVE TIME/CALL		RATIO
			PART	B-DIFF	
EI	75(75)	56393	5.59	50.82	9.09
	100(99)	88560	8.95	108.41	12.11

Table 2. PARTITION AND BORDER-DIFF

CLASS	N (EDGES)	TVS/CALL	AVE TIME/CALL		RATIO
WAVEFORM DATASET					
			PART	KAV-STA	
0	3343(562)	4988	0.96	25.92	27.00
SATIMAGE DATASET					
CLASS	N (EDGES)	TVS/CALL	AVE TIME/CALL		RATIO
			PART	KAV-STA	
1	1000(163)	5128	0.48	11.44	23.83
	1250(186)	6021	0.58	15.98	27.55
SPLICE DATASET					
CLASS	N (EDGES)	TVS/CALL	AVE TIME/CALL		RATIO
			PART	KAV-STA	
EI	100(99)	88560	9.07	80.56	8.88
	125(124)	124739	13.08	180.13	13.77

Table 3. PARTITION AND KAVVADIAS-STAVROPOULOS

Kavvadias-Stavropoulos and *Border-Diff* on half the number of instances shown here). The SE-Tree, while competitive for Waveform, suffered from a performance blowout on Satimage (˜241 times slower than Partition). With this approach, the dimensionality of the initial positive instance determines the potential search space. When this increases, the search space that must be examined (without the assistance of the a-priori property) becomes prohibitive and the algorithm becomes impractical.

Table 2 shows the results of a single *Border-Diff* call averaged over 100 calls. The results in this table clearly indicate the strength of our partitioning mechanism for handling these challenging datasets. There are many instances in which the improvement is more than an order of magnitude. Next, evaluation of our partitioning algorithm against the Kavvadias-Stavropoulos algorithm is presented in table 3. The results are also very impressive. Our system is considerably faster in all the cases. The improvements against *Border-Diff* and the Kavvadias-Stavropoulos algorithm are proportional to the number of edges, an expected, yet important result. Clearly, our partitioning algorithm is uniformly superior on all problem instances considered.

Acknowledgements: This work was supported in part by an Expertise Grant from the Victorian Partnership for Advanced Computing. Thanks to Elias Stavropoulos for providing an executable for the algorithm in [7].

References

[1] C. Berge. *Hypergraphs, North Holland Mathematical Library*, volume 45. Elsevier, 1989.

[2] E. Boros, V. Gurvich, L. Khachiyan, and K. Makino. On the Complexity of Generating Maximal Frequent and Minimal Infrequent Sets. In *Proceedings of STACS 2002*, pages 133–141, 2002.

[3] G. Dong and J. Li. Efficient Mining of Emerging Patterns: Discovering Trends and Differnces. In *Proceedings of KDD'99*, pages 43–52, 1999.

[4] T. Eiter and G. Gottlob. Identifying the Minimal Transversals of a Hypergraph and Related Problems. *SIAM Journal on Computing*, 24(6):1278–1304, 1995.

[5] M. L. Fredman and L. Khachiyan. On the Complexity of Dualization of Monotone Disjunctive Normal Forms. *Journal of Algorithms*, 21(3):618–628, 1996.

[6] D. Gunopulos, R. Khardon, H. Mannila, and H. Toivonen. Data Mining, Hypergraph Transversals, and Machine Learning. In *Proceedings of PODS'97*, pages 209–216, 1997.

[7] D. Kavvadias and E. C. Stavropoulos. Evaluation of an Algorithm for the Transversal Hypergraph Problem. In *Proceedings of WAE'99*, pages 72–84, 1999.

[8] J. Li, G. Dong, and K. Ramamohanarao. Making use of the most Expressive Jumping Emerging Patterns for Classification. In *Proceedings of PAKDD'00*, pages 220–232, 2000.

[9] H. Mannila and H. Toivonen. Levelwise Search and Borders of Theories in Knowledge Discovery. *Data Mining and Knowledge Discovery*, 3(1):241–258, 1997.

[10] R. Reiter. A Theory of Diagnosis from First Principles. *Artificial Intelligence*, 32(1):57–95, 1987.

[11] R. Rymon. An SE-Tree-Based Prime Implicant Generation Algorithm. *Annals of Mathematics and Artificial Intelligence*, 11(1-4):351–366, 1994.