# Scalable Multi-Relational Association Mining

Amanda Clare*
Department of Computer Science,
University of Wales Aberystwyth,
Aberystwyth, SY23 3DB, UK
afc@aber.ac.uk

Hugh E. Williams    Nicholas Lester
School of Computer Science and Information Technology,
RMIT University, GPO Box 2476V,
Melbourne, Australia 3001.
{hugh,nml}@cs.rmit.edu.au

## Abstract

*We propose the new* RADAR *technique for multi-relational data mining. This permits the mining of very large collections and provides a new technique for discovering multi-relational associations. Results show that* RADAR *is reliable and scalable for mining a large yeast homology collection, and that it does not have the main-memory scalability constraints of the Farmer and Warmr tools.*

## 1. Introduction

Large collections of multi-relational data present significant new challenges to data mining. These challenges are reflected in the annual KDD Cup competition, which involved relational datasets in 2001 and 2002, and network mining in 2003. The July 2003 edition of the ACM SIGKDD Explorations is devoted to position papers outlining the current frontiers in multi-relational data mining. Similar problems exist in bioinformatics databases — such as those at MIPS[1] — that provide integrated data on a genome-wide scale for whole organisms, with multiple cross references to other databases.

The vast majority of association mining algorithms are designed for single table, propositional datasets. We propose a novel technique for multi-relational association mining that permits efficient and scalable discovery of relationships. To our knowledge, the only existing multi-relational association mining algorithms are upgrades of Apriori [1] and, with the field in its infancy, there is much scope for improving the scalability of these solutions. Our technique uses an *inverted index*, a largely disk-based search structure that is used to support querying in all practical Information Retrieval systems and web search engines.

*This work carried out at and supported by the School of Computer Science and Information Technology at RMIT University.

[1]See: `http://mips.gsf.de/`

## 2. Inverted Indexes

An inverted index is a well-known structure used in all practical text retrieval systems [8]. It consists of an in-memory (or partially in-memory) search structure that stores the *vocabulary* of searchable terms, and on-disk *postings* that store, for each term, the location of that term in the collection. In practice, the vocabulary is typically the words that occur in the collection [8].

Using the notation of Zobel and Moffat [10], each term $t$ has postings $< f_{d,t}, d >$, where $f_{d,t}$ is the frequency $f$ of term $t$ in document $d$. Consider an example for the term “mining” that occurs in four documents:

$$< 2, 11 > \ < 1, 19 > \ < 1, 72 > \ < 2, 107 >$$

This *postings list* shows that the word “mining” occurs twice in document 11, once in document 19, once in document 72, and twice in document 107. The documents themselves are ordinally numbered, and a *mapping table* associates each document number to its location on disk. Despite its simplicity, this inverted index structure is sufficient to support the popular ranked query mode that is used by most search engine users.

The organisation, compression, and processing of postings lists is crucial to retrieval system performance. Compression is important for three reasons: first, a compressed representation requires less storage space than an uncompressed one; second, a retrieval system is faster when compression is used, since the cost of transferring compressed lists and decompressing them is typically much less than the cost of transferring uncompressed data; and, last, caching is improved because more lists fit into main-memory than when uncompressed lists are used. Scholer et al. [7] recently showed that compression of postings lists more than halves query evaluation times than when no compression is used.

## 3. Multi-Relational Association Mining

The first mining technique to find associations in multi-table relational data was Warmr [4]. Warmr is a first-order upgrade of Apriori, with the additional introduction of a user-defined language bias to restrict the search space. Blockeel et al. [2] have been investigating enhancements — such as query packs — to the underlying Prolog compiler to address efficiency issues. They have also implemented techniques to allow the user to limit the amount of data required to be loaded into main-memory. With Warmr, the user has the full power of the Prolog programming language for specifying the data and background knowledge.

PolyFARM [3] was based on the ideas of Warmr and written for distribution on a Beowulf cluster by partitioning the data to be counted. Unfortunately, although the size of the database is reduced by partitioning, the size of the candidate associations held in main-memory can grow impractically large.

Nijssen and Kok's Farmer [6] is a new multi-relational mining technique, with a running time that is an order of magnitude improvement over Warmr; indeed, on small data sets, Farmer can be astonishingly fast. However, they still require that all data is available in main-memory — a still significant problem for large datasets — and the main-memory use increases steadily throughout each search.

## 4. RADAR

We propose RADAR, the Relational Association Datamining AlgoRithm[2]. RADAR is the first multi-relational association mining algorithm that uses compressed inverted indexing techniques to provide a scalable solution for mining large databases.

Our aim is to count all *frequent associations* in a database. We use the language of first order logic to represent the associations. A frequent association is a conjunction or set of atoms that occurs with at least the minimum support frequency in the database [4]. For example, "a chardonnay wine that is made by an Australian grower" is represented by the association:

$$\mathrm{wine}(W) \wedge \mathrm{chardonnay}(W) \wedge \mathrm{grower}(W, G) \wedge \mathrm{australian}(G)$$

Inspired by the Eclat algorithm [9], we propose to mine these frequent associations by flattening the database, building an inverted index of the flattened database, and repeatedly joining postings lists.

In a multi-table relational database, we must decide which field in which table is our main key or notion of *transaction*, that is, what we are counting. For example, in a database representing wines, retailers, and growers, we must decide if we are interested in counting the number of Australian growers that make chardonnay wines, or the number of chardonnay wines that are made by Australian growers. We refer to this field as the COUNTKEY, so as to distinguish it from the common notion of a database key field.

[2]The RADAR software and sample databases are available from http://www.aber.ac.uk/compsci/Research/bio/dss/radar/

**Figure 1** Five tables representing molecules by atoms and bonds.

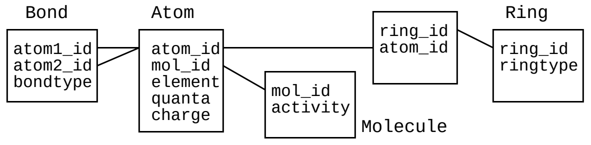

**Figure 2** Example of the two-column flattened database with keys. For example, line 2 describes a double bond in mol 12 between atoms 10 and 11.

| Keys (Arguments) | Attributes (Predicate Symbols) |
|---|---|
| m12 | inactive |
| m12, a10, a11 | bond_double |
| m12, a10 | elem_carbon, quanta_27, charge_medium |
| m12, a11, a12 | bond_double |
| m12, a11, a13 | bond_single |
| m12, a11 | elem_carbon, quanta_22, charge_medium |
| m12, r47, a10 | ring_benzene |
| m12, r47, a11 | ring_benzene |
| m12, r47, a12 | ring_benzene |

To prepare for indexing, the database is flattened into a single table with a two-column format. The first column stores the database keys (which represent the arguments to the predicates), and the second column stores the database items, that is, descriptive attributes (which represent the predicate names). We refer to these as *keys* and *predicate symbols* respectively. Each row of the flattened database can hold multiple keys and multiple predicate symbols.

The attributes in a simple multi-table relational database describing molecules represented by bonds and atoms are shown in Figure 1. Selected flattened rows from this database are shown in Figure 2. Flattening can be made more or less explicit depending on the application requirements.

Keys are used to form the arguments to the predicates. For example, if grower$(Wine, Grower)$ is to be a possible atom in associations, then any row in the flattened database that contains an instance of the grower term in the second column must always have both Wine and Grower keys listed in the first column of that row.

To create an inverted index for the flattened database, we number each row sequentially and use these numbers as the document numbers. All terms within a row are indexed, that

**Figure 3** A section of the inverted index of the flattened database from Figure 2. For compactness, the postings list show only document numbers; we have omitted $f_{d,t}$.

| Term ($t$) | Postings list ($d_1 \ldots d_{f,t}$) |
|---|---|
| inactive | 1 |
| bond_double | 2,4 |
| bond_single | 5 |
| elem_carbon | 3,6 |
| ring_benzene | 7,8,9 |
| quanta_22 | 6 |
| m12 | 1,2,3,4,5,6,7,8,9 |
| a10 | 2,3,7 |
| a11 | 2,4,5,6,8 |

is, both keys and attributes. A section of the inverted index for Figure 2 is shown in Figure 3.

To mine the data, the user provides the flattened database and a *language bias* (the set of factors that influence and direct the search). In our case, this is a list of the COUNTKEYS, a list of all the predicates for use in associations, the types and modes of their arguments, and other constraints. Associations are then generated depth-first.

All arguments to the predicates in an association are variables that can be satisfied by particular database keys. To count how frequently an association appears in the database — with respect to the COUNTKEY — we need to test whether, for each possible COUNTKEY, there is a set of keys that satisfy this relationship. This means that when we have multi-relational data we cannot simply intersect postings lists for predicates that appear within the same association because we are seeking to identify predicates that share the correct set of keys that hold the relationships between the predicates. The algorithm for counting associations using our compressed inverted index is shown in Figure 4.

**Figure 4:** Algorithm for counting an association

```
function countassoc($association$)
    $v_1..v_n$ ← fetch postings lists for each predicate $p_1..p_n$ in $association$
    foreach $countkey$ in $countkeys$ do
        $ck\_postings$ ← fetch postings list for $countkey$
        join $ck\_postings$ with each appropriate $v_i$
        if all $v_i$ are non-empty then
            if other args exist then
                if $doargs(1, v_1..v_n, association)$ then
                    $total$++
            else
                $total$++
    return $total$

function doargs($argnum, v_1..v_n, association$)
    $docs$ ← find shortest docs list amongst appropriate predicates
    foreach $doc$ in $docs$ do
        $key$ ← key of appropriate type for argnum from $doc$
        $k\_postings$ ← fetch postings list for $key$
        join $k\_postings$ with each appropriate $v_i$
        if all $v_i$ are non-empty then
            if other args exist then
                if $doargs(argnum + 1, v_1..v_n, association)$ then
                    return true
            else
                return true
    return false
```

## 5. Results

We present results of using RADAR, Warmr, and Farmer. All measurements were carried on a 1.66 GHz AMD Athlon-based workstation running Linux with 2 GB of main-memory. We used two small collections — for which RADAR is not optimised, but that are well-known and well-suited to the other schemes — and a large collection that illustrates the scalability of RADAR. MUTA is a well-known mutagenesis dataset [5], consisting of descriptions of molecules, including their atoms, bonds, and ring structures. KDD2002 is the collection used in Task 2 of the KDD 2002 Cup competition[3], that describes yeast proteins and their interactions. YEASTHOM is a large collection[4] that describes homologous relationships between yeast genes and proteins in the SwissProt database.

[3]See: http://www.biostat.wisc.edu/∼craven/kddcup/

[4]http://www.aber.ac.uk/compsci/Research/bio/dss/yeastdata/

We compared RADAR to Warmr (version ACE 1.2.6) and Farmer (2003). A fair, direct comparison is not straightforward as each algorithm has its own distinct properties. In particular, Farmer does not allow a limit on the length of the association, but only on the maximum use of each individual predicate. This means that we cannot stop Farmer from finding more, longer associations than the other algorithms.

Table 1 shows the results of our experiments. The results for MUTA and KDD2002 illustrate the general properties of the schemes: RADAR uses 34 Mb of main-memory for both collections, while the memory use of the other schemes varies significantly with the number of discovered associations (from 25 to 119 Mb for Warmr, and from 387 to 11 Mb for Farmer). Constant memory use comes at a price for small collections: RADAR is two to three times slower than the other schemes on the MUTA task, and unacceptably slow on the KDD2002 task compared to the fast Farmer.

The results for YEASTHOM illustrate the advantages of RADAR, and the disadvantages of the other approaches. RADAR is highly scalable: despite the almost thousand-fold increase in data size from KDD2002 to YEASTHOM, main-memory use only increases from 34 Mb to 56 Mb. Farmer — which is impressive on small datasets — is unsuitable for this task: main-memory use increases steadily throughout the lifetime of the task, since it holds the database and as-

| Data | Algorithm | Data size | | Maximum Memory | Time | Associations |
|---|---|---|---|---|---|---|
| | | Original | Compiled | Use (Mb) | | Found |
| MUTA | Warmr | 823 kb | 1,292 kb | 25 | 7.8 mins | 2,756 |
| | Farmer | 823 kb | — | 387 | 10.9 mins | 95,715 |
| | RADAR | 596 kb | 526 kb | 34 | 25.0 mins | 12,530 |
| KDD2002 | Warmr | 1,407 kb | 1,556 kb | 119 | 31.1 mins | 7,523 |
| | Farmer | 1,407 kb | — | 11 | 0.1 mins | 20,359 |
| | RADAR | 1,023 kb | 418 kb | 34 | 361.0 mins | 9,130 |
| YEASTHOM | Warmr | 841 Mb | 880 Mb | 800 | *25 days* | 7,712* |
| | Farmer | 1,465 Mb | — | 1,254 | 18 days | 698,974 |
| | RADAR | 1,565 Mb | 163 Mb | 56 | *25 days* | 34,782* |

**Table 1. Experiments on the MUTA, KDD2002 and YEASTHOM collections. For MUTA, support = 20 molecules (10.6%), max. assoc. length = 3 predicates (excluding $\mathrm{mol}(X)$). Farmer continued to find associations to length 11. For KDD2002, support = 20 ORFs (0.84%), max. assoc. length = 3 predicates (excluding $\mathrm{orf}(\mathrm{X})$). Farmer continued to find associations to length 8. For YEASTHOM, support = 20 ORFs (0.31%), max. assoc. length = 3 predicates (excluding $\mathrm{orf}(X)$). Italicised figures indicate that the algorithm was still running. Farmer continued to find associations to length 7 but stopped before completion due to main memory exhaustion. Warmr's maximum memory use was set to 800 Mb.**

sociations in memory. Indeed, after 18 days, main-memory was exhausted. Warmr processes associations in *packs* that group together common subparts for faster counting. This means that no results are given until a whole level is complete. For the YEASTHOM collection, associations of length two were produced after about six hours, and then the system gave no further output for several weeks.

RADAR is structured — similarly to Farmer — as an *anytime* algorithm that produces continuous output. Further, RADAR can be seeded with an association, so that the application can be restarted at any time. This aspect is useful for large-scale mining problems that run for weeks.

## 6. Conclusion

Large multi-relational collections are the next frontier for data mining. In this paper we have shown how compressed inverted indexes used in text retrieval systems can be adapted for multi-relational data mining. Our technique, RADAR, is both scalable and reliable on large amounts of data. It produces output continuously, with the option of stopping and resuming the mining process later. For small datasets — for which RADAR is not designed — the Warmr and Farmer techniques should be used in preference.

## Acknowledgements

This work was supported by the Australian Research Council.

## References

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *20th International Conference on Very Large Databases (VLDB 94)*, 1994.

[2] H. Blockeel et al. Improving the efficiency of Inductive Logic Programming through the use of query packs. *Journal of Artificial Intelligence Research*, 16:135–166, 2002.

[3] A. Clare and R. D. King. Data mining the yeast genome in a lazy functional language. In *Practical Aspects of Declarative Languages (PADL'03)*, 2003.

[4] L. Dehaspe. *Frequent Pattern Discovery in First Order Logic*. PhD thesis, Department of Computer Science, Katholieke Universiteit Leuven, 1998.

[5] R. King, S. Muggleton, A. Srinivasan, and M. Sternberg. Structure-activity relationships derived by machine learning. *Proc. Nat. Acad. Sci. USA*, 93:438–442, 1996.

[6] S. Nijssen and J. N. Kok. Efficient frequent query discovery in FARMER. In *13th International Conference on Inductive Logic Programming (ILP 2003)*, 2003.

[7] F. Scholer, H. E. Williams, J. Yiannis, and J. Zobel. Compression of inverted indexes for fast query evaluation. In K. Järvelin, M. Beaulieu, R. Baeza-Yates, and S. H. Myaeng, editors, *Proc. ACM-SIGIR International Conference on Research and Development in Information Retrieval*, pages 222–229, Tampere, Finland, 2002.

[8] I. Witten, A. Moffat, and T. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kaufmann Publishers, Los Altos, CA 94022, USA, second edition, 1999.

[9] M. J. Zaki. Scalable algorithms for association mining. *IEEE Transactions on Knowledge and Data Engineering*, 12(3):372–390, 2000.

[10] J. Zobel and A. Moffat. Exploring the similarity space. *ACM SIGIR Forum*, 32(1):18–34, 1998.