View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Flinders Academic Commons

Archived at the Flinders Academic Commons:

http://dspace.flinders.edu.au/dspace/

This is the publisher's copyrighted version of this article.

The original can be found at:
http://www.jrpit.acs.org.au/jrpit/JRPITVolumes/JRPIT39/JRPIT39.1.35.pdf

© 2007 Journal of Research and Practice in Information Technology

Published version of the paper reproduced here in accordance with the copyright policy of the publisher. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from Journal of Research and Practice in Information Technology.

Incremental Association Mining using a Closed-Set Lattice

Aaron Ceglar and John F. Roddick

School of Informatics and Engineering
Flinders University of South Australia
PO Box 2100, Adelaide, South Australia 5001
Email: {aaron.ceglar, john.roddick}@infoeng.flinders.edu.au

The use of closed-set algorithms to generate condensed accurate representations of a dataset's frequent itemsets has been well documented. This paper presents a novel approach to incremental association mining in which the maintenance of the set of frequent itemsets is based upon the evolution of a closed-set lattice. This approach also creates a closed-set representation of the increment dataset, providing the user with insight to the increment's effect upon the maintained lattice and provides an effective means of incorporating windowing functionality.

Additional Keywords: Association mining, Incremental mining, Closed-set lattice
ACM Classification: I.5.2 (Pattern analysis)

1. INTRODUCTION

Incremental association mining research concerns the maintenance of the set of frequent itemsets, F, in an evolving dataset. Given F_{D^i}, the set of frequent itemsets generated from the evolving dataset D^i, where i signifies an evolutionary step of D, the incorporation of an increment dataset δ (such that $|\delta| \ll |D^i|$) will affect the degree of element presence in the combined dataset $D^1 = D^0 + \delta$. The naive computation of F_{D^i} involves re-mining D^i using a classic association mining algorithm, however this results in process replication as a significant part of the knowledge produced by the mining of D^i is already available in $F_{D^{i-1}}$. Hence incremental mining attempts to facilitate the inclusion of δ into F_{D^i} by using currently available information in $F_{D^{i-1}}$ and D^{i-1}. This evolution results in the alteration of participant states, summarised in Table 1, where the participants are the elements in D and the itemsets in F.

State	Description
Static	No relative change in participant presence.
Strengthened	The presence of the participant increases.
Weakened	The presence of the participant decreases.
Emergent	An infrequent participant becomes frequent.
Declined	A frequent participant becomes infrequent.

Table 1: Alteration of Participant state

Copyright© 2007, Australian Computer Society Inc. General permission to republish, but not for profit, all or part of this material is granted, provided that the JRPIT copyright notice is given and that reference is made to the publication, to its date of issue, and to the fact that reprinting privileges were granted by permission of the Australian Computer Society Inc.

Manuscript received: 19 October 2005
Communicating Editor: John Yearwood

This paper presents a novel technique that advances the state of increment association rule mining. The algorithm, Maintained Closed-set Lattice, MCL, like previous approaches, uses currently available information. Unlike previous algorithms however it uses the concept of the *closure of the Gauloise connection* (Ganter and Wille, 1999), or closed-sets, to base the incremental mining upon a condensed representation lattice from which F can be inferred.

This approach shows promise in both process optimisation and in facilitating the user interpretation of results. The processing is optimised, especially in highly correlated datasets, as increment datasets are applied to a smaller maintained lattice, reducing the search space. The maintenance of a closed-set lattice also aids user interpretation due to its reduced size. The other significant contribution of this algorithm is the production of an increment lattice I during the mining process. This provides the user with insight to the increment's effect upon the maintained lattice L and also provides an effective means of incorporating *windowing* functionality (Cheung *et al*, 1997), or the removal of previous increments, without further mining.

The rest of this paper is organised as follows: Section 2 discusses previous incremental mining algorithms. Section 3 presents the concept of closed-sets and discusses the CHARM algorithm that is incorporated within MCL. Section 4 presents MCL and Section 5 concludes the paper.

2. INCREMENTAL MINING ALGORITHMS

Cheung *et al* (1996) proposed the first incremental association mining algorithm, FUP (Fast UPdate), which defines a set of incremental inclusion rules to incorporate incremental functionality within the Apriori algorithm (Agrawal *et al*, 1993). This was subsequently extended in FUP2 (Cheung *et al*, 1997) to handle increment removal and to use statistical sampling to decide when to incorporate increment datasets. Like Apriori, these algorithms require k scans of D, where k represents the length of the largest frequent itemset F_i. Ayan *et al* (1999) propose UWEP (Update with Early Pruning) that provides further optimisation by scanning D at most once and the increment dataset δ exactly once. UWEP incorporates *tidlists* and look-ahead strategies to improve pruning, however it only promotes candidate itemsets if they are frequent in both D and δ, resulting in the possible removal of valid itemsets.

Other incremental extensions to the classic association mining algorithms have been proposed, for example Partition (Savasere *et al*, 1995) and FP-Growth (Han and Pei, 2000) and also to the notion of negative borders (Toivonen, 1996). An incremental Partition extension is proposed by Omiecinski and Savasere (1998) and a fundamentally equivalent foundation is used as the basis for Lee *et al*'s sliding window filtering algorithm, SWF, (Lee *et al*, 2001) and its proposed extension FI_SWF (Chang and Yang, 2003). Two incremental extensions to FP-Growth have been proposed, both of which avoid rebuilding the FP-tree structure when relative frequencies change. This is achieved through the development of a modified tree structure CATS-Tree (Cheung and Zaïane, 2003) and the development of an associated Pattern Repository structure from which FP-tree can be derived by Relue *et al* (2001). Negative borders are used in ULI (Thomas *et al*, 1997) that requires at most one scan of D and k scans of δ. Ganti *et al* (2001) extend ULI by using tidlists and introducing data stream monitoring to determine when to update. They also present a generic model, GEMM, which provides a general framework through which incremental algorithms can be extended to incorporate *windowing*.

More recently, incremental extensions (Veloso *et al*, 2001; 2002) have been proposed for maximal frequent set algorithms, MFS, in which only the largest frequent itemsets are maintained (Bayardo Jr, 1998). Although this results in process optimisation, the derived rules are approximations only as the actual support of smaller frequent itemsets is unknown. PELICAN,

(Veloso *et al*, 2001), extends MaxEclat, (Zaki, 2000), using the concept of prefix-based equivalence relations. Subsequent work by the same team proposed ZIGZAG, (Veloso *et al*, 2002), that extends GenMax, (Gouda and Zaki, 2001), using naive backtracking for searching and introducing two novel quality heuristics. ZIGZAG was further extended in WAVE (Veloso *et al*, 2002) to incorporate estimation techniques and trend analysis to efficiently maintain an approximate data model, which although further improving processing time comes at an additional cost to accuracy. Although not based upon the principle of MFS, MAAP (Zhou and Ezeife, 2001) is similar in its approximate inference of small frequent itemsets. The algorithm uses an Apriori based framework whereby given the high-level F_D, it is able to compute the equivalent high-level F_δ, and also infer some of the lower-level F_δ.

In contrast, MCL presents a novel closed-set approach to incremental association mining that presents a significantly smaller footprint than previous classic incremental algorithms, while still allowing the derivation of accurate support for all valid itemsets, unlike MFS approaches. This is especially true in dense dataset environments due to the increased compression of closed-set lattice representations. Furthermore MCL presents novel insight into the effect of the increment (δ) upon the lattice by generating a closed-set increment lattice as part of the update process, and which is also used to facilitate the subsequent removal of appended increments.

3. CLOSED SETS

Closed-set mining algorithms produce a condensed representation of the frequent itemsets within a dataset from which the complete set of F can be accurately derived. The theoretical foundation of closed-sets is based upon the closure of the Gauloise connection (Ganter and Wille, 1999) in which a closed pattern is the largest pattern common to a set of objects within a dataset. Non-closed patterns therefore have the same presence (support (σ)) as their closures, but are a subset thereof. Therefore the closure of itemset i, denoted $c(i)$ is the smallest closed pattern containing i.

Given that D is comprised of elements X and the objects O within which they participate, the identification of the closure of i, $c(i)$ is based upon the collaboration of two functions $t(i)$ and $g(o)$, where $t(i)$ returns the set of objects in which i participates and $g(o)$ identifies the set of elements common to all objects $o|o \in O$. The closure of i is found through $g(t(i))$, whereby the set of objects in which i participates is identified $(t(i))$ and then the set of elements common to this set of objects is derived. From this $g(t(i))=c(i)$, as the elements that always occur with i are identified, hence $c(i) \supseteq i$ and $\sigma(c(i))=\sigma(i)$. For example, an itemset $\{xy\}$ is not closed if every transaction containing $\{xy\}$ also contain z.

Closed set algorithms (Pasquier *et al*, 1999; Pei *et al*, 2000; Zaki and Hsiao, 2002) identify the closed itemsets, L, within D using different techniques. The additional constraints incorporated within these association mining algorithms significantly reduce the search space, especially in highly correlated datasets, improving efficiency. Once L is generated the derivation of F and the subsequent association rules is simple. However since L implies F, the generation and presentation of closed rules can facilitate user interpretation, where closed rules (Pasquier *et al*, 1999) are a reduced set of association rules derived from L, of the form $i_1 \Rightarrow i_2 - i_1 \mid i_1 \subset i_2 \wedge i_1, i_2 \in L$ the confidence of which is available, $(\sigma(i_2)/\ \sigma(i_1))$.

The closure properties adapted within MCL were first developed by Zaki and Hsiao (2002), in CHARM, a closed-set algorithm based upon the concept of equivalence classes (Zaki *et al*, 1997), in which two itemsets belong to the same k-class if they share a common prefix of $|k|$. The closure properties, Figure 1, are based upon the relationship between two itemsets tidlists, or objects within which the itemsets exist, within an equivalence class. Properties 1 and 2 result in equivalence class

1: *If* $t(i_1) = t(i_2)$ *then* $c(i_1) = c(i_2) = c(i_1 \cup i_2)$, *implies that all occurrences of* i_1 *can be replaced with* $i_1 \cup i_2$ *and* i_2 *can be removed from further consideration as its closure* $c(i_2) = c(i_1)$.

2: *If* $t(i_1) \subset t(i_2)$ *then* $c(i_1) \neq c(i_2)$ *but* $c(i_1) = c(i_1 \cup i_2)$ *implies similar replacement, however as* $c(i_2) \neq c(i_1)$, i_2 *cannot be removed from further consideration.*

3: *If* $t(i_1) \supset t(i_2)$ *then* $c(i_1) \neq c(i_2)$ *but* $c(i_2) = c(i_1 \cup i_2)$, *the inverse of Property 2 in which* i_2 *is replaced by* $i_1 \cup i_2$ *instead of* i_1.

4: *If* $t(i_1) \neq t(i_2)$ *then* $c(i_1) \neq c(i_2) \neq c(i_1 \cup i_2)$, *indicates that both* i_1 *and* i_2 *lead to different closures and hence no replacement can occur.*

Figure 1: CHARM closure properties

reduction and hence facilitate closed-set convergence, whilst 3 and 4 result in new equivalence class information that generally requires additional processing.

An equivalence class is comprised of a set of frequent itemset members that share a common *k*-prefix. New *k*+1 equivalence classes are derived from existing classes by merging each member with all members that lexicographically occur after it. If the merged pair results in a frequent itemset i, the closure properties are applied to control i's influence upon the generation of the closed-set equivalence class lattice. Each class c is then appended to L if it is not subsumed by a class c', such that $c \subset c'$ and $t(c)=t(c')$.

4. MAINTAINED CLOSED-SET LATTICE ALGORITHM

MCL provides a novel and efficient incremental association mining method through the maintenance of a closed-set lattice L, from which the set of frequent itemsets F, can be easily derived. The maintenance of L^i uses an increment closed-set lattice I that is derived from the increment dataset δ in the presence of D^{i-1} and L^{i-1}. I is then appended or removed from L^{i-1} resulting in L^i, an evolution of the maintained closed-set lattice. By assuming that an increment dataset must be appended to the maintained lattice before it can be removed, the removal process can be optimised by using the previously derived I, alleviating the need to remine δ. Furthermore the generation of I during the append process provides the user with an effective insight to δ's effect upon L.

The algorithm assumes an initial L^0 and D^0. The internal representation of the increment dataset δ, referred to as d (see Section 4.1), and D^i are vertically organised tidlists and the subsumption table, used as part of closed-set validation takes the tuple form $\{tidlist, Array[L_i]\}$. The lattice structures L and I are prefix trees, of node form $\{itemset, tidlist\}$ that use lexicographic ordering to maintain consistency during evolution. The output from both the append and remove process is an updated L^i and D^i, the append process also produces I and d to facilitate user interpretation and subsequent increment removal.

The following discussion is divided into four sections. The first three discuss the main steps in the append process: *generate*, *merge* and *strip*, while the last section discusses the removal process. The *generate* stage prepares the required data structures including the increment lattice, I, while *merge* and *strip* perform the update, focusing respectively upon the update of existent and emergent closed-sets. Figure 2 presents L^0 and D^0, from Zaki and Hsiao (2002), which is used as the running example.

Object	*Elements*
1	a,c,t,w
2	c,d,w
3	a,c,t,w
4	a,c,d,w
5	a,c,d,t,w
6	c,d,t

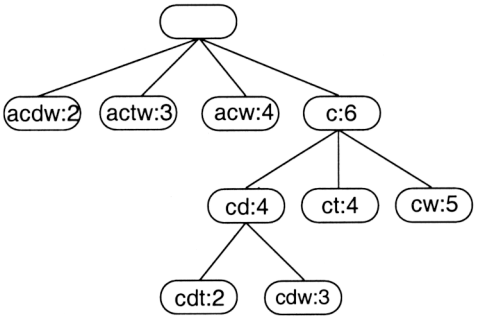

Figure 2: Dataset and resulting closed set lattice (Zaki and Hsaio, 2002)

4.1 Generate

This stage prepares for the subsequent merging of L and I through the preparation of data structures and the subsequent generation of I and X. Data structure preparation requires a scan of the increment dataset δ which is used to update D $\forall$ $e \in \delta$, generating D^1 from D^0. During this traversal the data structures d and ext are also populated. Given $\sigma(D_e^0) > minsup$ and $\sigma(D_e^1) > minsup$ then all objects containing $e \in \delta$ are appended to d, resulting in the list of δ objects which already exist in L. If $\sigma(D_e^0) < minsup$ and $\sigma(D_e^1) > minsup$ then e is an emergent element and its context within D^1, namely D_e^1, is appended to ext, resulting in the list of emergent elements in D^1 that may extend L^0.

The subsequent closed mining of d and ext, using the closure principles identified by Zaki and Hsiao (2002) in CHARM, see Section 3, result in the generation of the lexicographic tree's I and X respectively. While the mining of ext incorporates quality heuristics (eg. support), this does not apply to the mining of d, as all itemsets founded upon current frequent elements, those existant within L^0, must be reported in I to accurately update all itemsets in L. Once constructed I and X, presented in Figure 3 with δ, contain all the information required to accurately update L.

Pseudo 1 Generate

```
Update
 1: for all d_e ∈ d do
 2:    D_e^1 = D_e^0 + d_e
 3:    if σ(D_e^1) > minsup then
 4:       di.append(d_e)
 5:       if σ(D_e^0) < minsup then
 6:          ext.append(d_e)
 7:       end if
 8:    end if
 9: end for
10: I = closeMine(di, 1)
11: X = closeMine(ext, minsup)
```

4.2 Merge

The incorporation of I and X within L requires a scan of L for which all pertinent $x \in X$ and $i \in I$ are applied to $l \in L$. Given that $|\delta| \ll |D^i|$ and the set of elements $e \in E$ is common to both D and δ, relatively few new itemsets are appended to L during any given increment. The result of this is that an increment's effect upon L will often be internal to the lattice as most frequent itemsets are already

Object	Elements
7	a,c,d,t,z,x
8	a,t,x
9	c,d,z,x

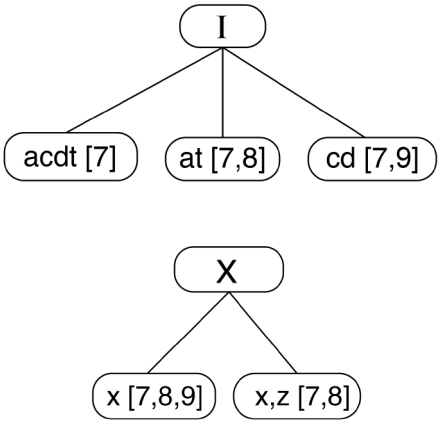

Figure 3: Increment dataset and derived *I* and *X*

represented. However, although less common, lattice boundary extension does occur through emergent itemsets and superset extension. Therefore each l must be tested against X for emergent supersets and against E to update its tidlist and to discover any emergent subsets. This process is optimised by considering only relevant X and E for each l by dynamically reducing these structures during processing.

Each $l \in L^0$ is checked (preorder traversal) against X and I, resulting in the construction of a new closed lattice L^1. The construction of this new lattice was found to significantly reduce update complexity in comparison to updating the original lattice. The set of emergent itemsets (X) applied to l is the subset of X that was found valid for l_{parent}, this set is denoted $X_{\bar{l}}$ and therefore $X_{root} = X$ (given that $l = L_{root}$ then $X_l = X$). For example, given $X=\{\{x\},\{xz\}\}$ and the resultant processing of $l=\{a\}$ finds that $X_a = \{x\}$, then this subset of X is the extension set passed to the children of a. Implementing the downward closure principle upon this process, X_l quickly reduces as the traversal deepens. Hence each l is merged with $X_{\bar{l}}$ and if frequent and not subsumed (closed set pruning), it is appended to L^1, X_l and R (discussed in Section 4.3).

The increment lattice is pruned during processing, see Section 4.3, so that the pertinent $i \in I$ for the current l are the first nodes encountered using a preorder traversal. Thus I is traversed until i lexicographically exceeds l at which point no subsequent i are pertinent to l.

The inclusion of further search space reduction depends upon the effect of I and X upon l and the relationship between i and l. For each pertinent i, if $i \supseteq l$ then L is updated with $i.tidlist$ and i's descendents are removed from the search space as the tidlists (all supersets of i) are subsets of

(a) Partial *L* **(b) Partial *I***

Figure 4: Partial *L* and *I*

i.tidlist and cannot further affect *l*. If $i \subset l$ then transcend to *i*'s children, with no action taken upon *l*. If *i* is neither a superset, a subset nor equal to *l*, then its subtree is not processed. For example, given the partial lattices in Figure 4, $l = \{ad\}$ is first compared against $i = \{a\}$, since $\{a\}$ is a subset of $\{ad\}$ no update occurs. *i* then transcends to $\{adt\}$, which being a superset of $\{ad\}$ results in an update of *l*, but any descendants of *i* are not processed. Comparison then proceeds to $i = \{adw\}$ also a superset therefore resulting in the update of *l*. Given that the next *i* lexicographically exceeds *l* then the update of $l = \{ad\}$ is complete. Since $\{ad\}$ incurred a valid update, its descendents are processed (namely $\{adw\}$). *i* is reset to L_{root} or $i = \{a\}$, as $i = \{a\}$ is a subset of $\{adw\}$ no update is undertaken and the descendants of $\{a\}$ are processed. $i = \{adt\}$ is unrelated to $l = \{adw\}$, therefore it and its descendants are ignored. *i* becomes equal to $\{adw\}$ which is equivalent to the current *l*, and therefore updated.

The inclusion of these search constraints optimises the update (strengthening and weakening) of existing $l \in L^0$. If there is no change to *l*, then no superset of *l* will be modified by *X* or *I* due to the Downward Closure Principle (DCP) (Agrawal *et al*, 1993). Hence the subtree of *l* is appended to L^1, without further comparison against *I* or *X*. However before any *l* is appended to L^1 a final check against *minsup* is made to identify declined itemsets, where if *l* has declined then *l* and its subtree (supersets) are eliminated from consideration. Pseudo 2 presents the *merge* process, including the *update* method that details the traversal of pertinent *I* for *l*.

Pseudo 2 Merge

```
merge(l,X)
 1: for all l' | l' ∈ l_child do
 2:    if l_parent == L_root then strip(l')
 3:    modified = l'.update(l',I)
 4:    if σ(l')> minsup then
 5:       if modified then
 6:          if !subsumed(l') then
 7:             L^1.append(l')
 8:             X' = extCheck(l',X)
 9:          end if
10:          merge(l', X')
11:       else
12:          L^1.appendTree(l',minsup)
13:       end if
14:    end if
15: end for
```

```
update(l,i)
 1: for all i' | i' ∈ i_child do
 2:    if i'.exceeds(l) then break
 3:    mod = false
 4:    if i' ⊇ l then
 5:       l.updateTid(i')
 6:       mod = true
 7:    end if
 8:    if i' ⊂ l then mod = update(l, i')
 9: end for
10: return mod
```

4.3 Strip

Strip is instigated from *merge* when a level-1 node is encountered during the traversal of L^0. It provides two functions, the reduction (stripping) of *I* to eliminate unwanted elements and the discovery of all remaining emergent itemsets. Prior to instigation a copy of *I* is made, denoted *R*, which provides the ability to both report the increments effect upon L^0 and to facilitate the removal process (Section 4.4). Therefore any emergent itemsets appended to L^1 are also inserted into R.

Due to lexicographic ordering, once all $l \in L^0$ that begin with a particular element *e* have been merged with *I* and *X*, all subsequent *l* must exceed *e*. Since *e* will take no further part in the update of L^0 it is stripped from all $i \in I$, allowing the progressive reduction of *I*.

Strip occurs before the *merge* of level-1 L^0 nodes, removing all elements from I that lexicographically precede the first element of the current l. Once the preceding elements have been removed, the modified itemsets, if not subsumed, are re-inserted into I. For example, given a level-1 $l = \{d\}$ and $I = \{\{ac\},\{cd\},\{dw\}\}$, then after stripping $I = \{\{d\},\{dw\}\}$. This dynamically reduces I and facilitates the update of L^0 by ensuring that the relevant increment closed-sets are the first encountered during *merge*, See Pseudo 3:Strip #1- #6.

The discovery of emergent closed-sets is undertaken in conjunction with stripping due to the focus upon the relevant structures. All candidate emergent set information is represented within I and X, and by removing their discovery from the traversal of L significant process duplication is avoided. The emergent closed-sets comprised of frequent elements in D^0 are represented in I, while emergent elements are represented in X.

The candidate emergent sets in which the stripped elements, S, participate are identified by deriving the set of itemsets from the closed-sets in I in which an $s \in S$ participates. For example, given $I = \{\{ac\},\{awx\},\{cd\},\{dw\}\}$, and $S = \{a\}$ then the set of candidate emergent itemsets $C = \{\{ac\},\{aw\},\{ax\},\{awx\}\}$, See Pseudo 3:Strip #2. The potential candidate emergent itemsets p are generated through a preorder traversal of I, incorporating DCP. If $p \notin C$, its support in D^1 is discovered and it is appended to C, irrespective of whether $\sigma(p) > minsup$. The representation of all p in C irrespective of support optimises processing as duplicate p can be quickly removed without calculating support.

The resulting C is then appended to L^1 and R where each $c \in C$ is frequent and not subsumed. Furthermore, if appended to L^1, c is then checked against the emergent elements X in like manner to l, see Section 4.2, to possibly generate further emergent supersets, See Pseudo 3:generateEmergent. The identification of the emergent closed-sets, although algorithmically complex consumes relatively little process time due to the small number of candidate sets generated and the pruning techniques incorporated.

After L^0 has been updated through the merge process the remaining emergent closed-sets are discovered by applying the same process to what remains of I. Finally the emergent elements, $x \in X$, are appended to L^1 and R if they are not subsumed.

Pseudo 3 Strip

```
strip(l)
 1: for all i | i ∈ I ∧ i_1. < l_1 do
 2:     C = genCandidateEmergent(i,l)
 3:     i.removeFromParent()
 4:     i.removePreceding(l)
 5:     if !subsumed(i) then
 6:         I.insert(i)
 7:     end if
 8:     generateEmergent(C)
 9: end for
```

```
generateEmergent(C)
 1: for all c | c ∈ C do
 2:     if σ(c) > minsup & !subsumed(c) then
 3:         L^1.append(c)
 4:         R.append(c)
 5:         extCheck(l',X)
 6:     end if
 7: end for
```

4.4 Removal

The creation of the increment lattice, I, facilitates the removal of previous increments from L by alleviating the need to re-mine the increment dataset, significantly reducing the removal process through the reuse of information. The maintained dataset D is first updated by removing the

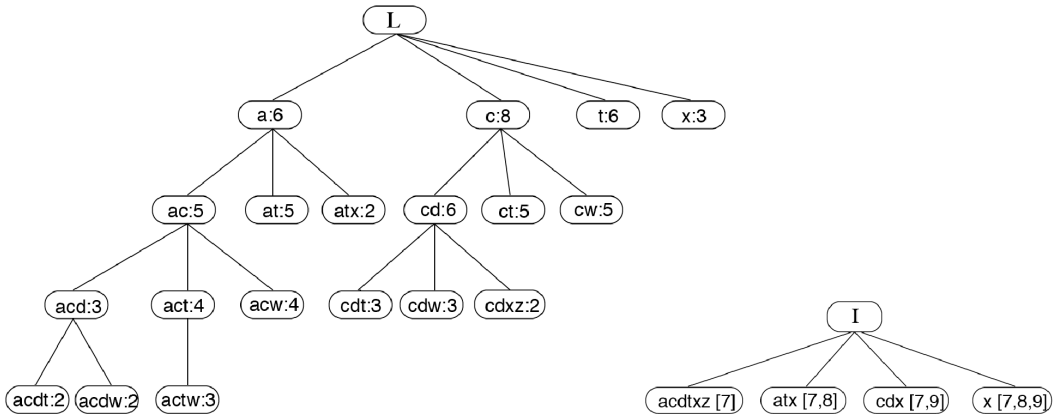

Figure 5: Resultant *L* and *I*

increment dataset, δ, from it. The subsequent updating of the maintained closed-set lattice, L, is similar to the merge stage of append, in that it is based upon the traversal of L and uses the consistent ordering to reduce the search space.

For each $l \in L$,I is traversed until i lexicographically exceeds l or $i \supset l$. If $i \supset l$ then $i_{tidlist}$ is removed from $l_{tidlist}$, which may result in the subsequent declination and removal of l and its supersets from L. If l remains frequent, its presence within the subsumption table is altered to reflect its new tidlist and if l subsumes its parent, the parent is replaced by l in L.

5. RESULTS AND CONCLUSION

This paper presents MCL, a novel and efficient algorithm that maintains a closed-set lattice in an evolving dataset environment. The maintained lattice L and increment lattice I resulting from this process are presented in Figure 5.

Some experimental results are presented in Figure 6, which compares processing time and lattice size between a naive algorithm and MCL as dataset density increases. These graphs are based

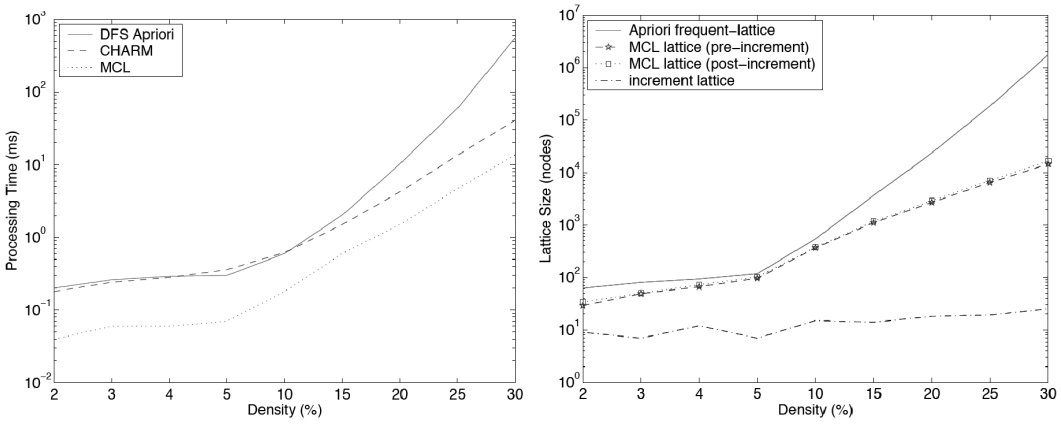

Figure 6: Experimental results (log_{10})

upon results from an artificial dataset, $|D^0| = 35K$ and $|\delta| = 2K$, in which density is manipulated by adjusting the number of elements from which the objects are generated. Figure 6(a) illustrates the efficiency of MCL over the naive re-mining of D^1, using Apriori, and also presents the time taken to construct the initial lattice L^0 using CHARM. Figure 6(b) illustrates, from the same result set, the relative lattice size reduction of the maintained and increment lattices, L and I, over the regular (non-closed) lattice as density increases.

The contributions of MCL to incremental association mining are twofold. First, through the use of closed-sets a condensed representative lattice is maintained that facilitates efficient update, especially for dense dataset environments, through reduced processing. Second, the creation of a closed-set increment lattice allows insight to the increment's effect upon the maintained lattice and reduces the processing required for subsequent increment removal.

The results to date support the theory of closed incremental mining and its contribution to data mining, especially within dense datasets environments. Further testing, against other incremental association mining algorithms, is underway to discover the extent of these contributions.

REFERENCES

AGRAWAL, R., IMIELINSKI, T. and SWAMI, A. (1993): Mining association rules between sets of items in large databases. In *ACM SIGMOD International Conference on the Management of Data*, Eds. BUNEMAN, P. and JAJODIA, S. ACM Press, Washington DC, USA, 207–216.

AYAN, N.F., TANSEL, A.U. and ARKUN, E. (1999): An efficient algorithm to update large itemsets with early pruning. In *5th International Conference on Knowledge Discovery and Data Mining (SIGKDD'99)*. ACM Press, San Diego, CA USA, 287–291.

BAYARDO, Jr, R. (1998): Efficiently mining long patterns from databases. In *ACM SIGMOD International Conference on the Management of Data, SIGMOD'98*. ACM Press, Seattle, WA, USA, 85–93.

CHANG, C.-H. and YANG, S.-H. (2003): Enhancing SWF for incremental association mining by itemset maintenance. In *7th Pacific-Asia Conference on Knowledge Discovery and Data Mining, PAKDD 2003*, Eds. WHANG, K.-Y., JEON, J., SHIM, K. and SRIVASTAVA, J., LNAI, Springer, Seoul, Korea, 2637: 301–312.

CHEUNG, D., HAN, J., NG, V. and WONG, C. (1996): Maintenance of discovered association rules in large databases: an incremental updating technique. In *12th International Conference on Data Engineering (ICDE'96)*, Ed. SU, S.Y.W., IEEE Computer Society, New Orleans, Louisiana, USA, 106–114.

CHEUNG, D. W.-L., LEE, S. D. and KAO, B. (1997): A general incremental technique for maintaining discovered association rules. In *5th International Conference On Database Systems For Advanced Applications*. Melbourne, Australia, 185–194.

CHEUNG, W. and ZAÏANE, O.R. (2003): Incremental mining of frequent patterns without candidate generation or support constraint. In *7th International database Engineering and Applications Symposium (IDEAS'03)*. Hong Kong, China.

GANTER, G. and WILLE, R. (1999): Formal concept analysis: Mathematical foundations. Springer.

GANTI, V., GEHRKE, J. and RAMAKRISHNAN, R. (2000):. Demon: Mining and monitoring evolving data. *IEEE Transactions on Knowledge and Data Engineering* 13(1): 50–63.

GOUDA, K. and ZAKI, M.J. (2001): Efficiently mining maximal frequent itemsets. In *IEEE International Conference on Data Mining*. IEEE Press, San Jose.

HAN, J. and PEI, J. (2000): Mining frequent patterns by pattern growth: Methodology and implications. *SIGKDD Explorations,* 2(2): 14–20.

LEE, C.-H., LIN, C.-R. and CHEN, M.-S. (2001): Sliding-window filtering: An efficient algorithm for incremental mining. In *10th International Conference on Information and Knowledge Management*, CIKM 2001. ACM, Atlanta, Georgia, USA, 263–270.

OMIECINSKI, E. and SAVASERE, A. (1998): Efficient mining of association rules in large dynamic databases. In *16th British National Conference on Databases (BNCOD'98)*. Cardiff, Wales, UK, 49–63.

PASQUIER, N., BASTIDE, Y., TAOUIL, R. and LAKHAL, L. (1999): Discovering frequent closed itemsets for association rules. In *7th International Conference on Database Theory (ICDT99)*. Springer, Jerusalem, Israel, 398–416.

PEI, J., MAO, R., HU, K. and ZHU, H. (2000): Towards data mining benchmarking: a test bed for performance study of frequent pattern mining. In *ACM SIGMOD International Conference on the Management of Data (SIGMOD 2000)*, Eds. CHEN, W., NAUGHTON, J. and BERNSTEIN, P.A., ACM Press, Dallas, TX, USA, 592.

RELUE, R., WU, X. and HUANG, H. (2001): Efficient runtime generation of association rules. In *10th ACM International Conference on Information and Knowledge Management*. Atlanta, Georgia, USA, 466–473.

SAVASERE, A., OMIECINSKI, E. and NAVATHE, S. (1995): An efficient algorithm for mining association rules in large databases. In *21st International Conference on Very Large Data Bases, VLDB'95*, Eds. DAYAL, U., GRAY, P.M.D. and NISHIO, S., Morgan Kaufmann, Zurich, Switzerland, 432–444.

THOMAS, S., BODAGALA, S., ALSABTI, K. and RANKA, S. (1997): An efficient algorithm for the incremental updation of association rules. In *3rd International Conference on Knowledge Discovery and Data Mining (KDD 97)*. ACM Press, New Port Beach, CA, USA, 263–266.

TOIVONEN, H. (1996): Sampling large databases for association rules. In *22nd International Conference on Very Large Data Bases, VLDB'96*, Eds. VIJAYARAMAN, T., BUCHMANN, P., MOHAN, C. and SARDA, N., Morgan Kaufmann, Mumbai (Bombay), India, 134–141.

VELOSO, A., POSSAS, B., MEIRA Jr, W. and DE CARVALHO, M.B. (2001): Knowledge management in association rule mining. In *Integrating Data Mining and Knowledge Management, held in conjunction with the 2001 IEE International Conference on Data Mining (ICDM)*. San Jose, CA, USA.

VELOSO, A.A., MEIRA Jr, W., DE CARVALHO, M.B., POSSAS, B., PARTHASARATHY, S. and ZAKI, M. (2002): Mining frequent itemsets in evolving databases. In *2nd SIAM International Conference on Data Mining (SDM'02)*, Eds. GROSSMAN, R.L., HAN, J., KUMAR, V., MANNILA, H. and MOTWANI, R., SIAM, Arlington, VA, USA.

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

ZAKI, M.J. and HSIAO, C.-J. (2002): CHARM: An efficient algorithm for closed itemset mining. In *2nd SIAM International Conference on Data Mining (SDM'02)*, Eds. GROSSMAN, R.L., HAN, J., KUMAR, V., MANNILA, H. and MOTWANI, R., SIAM, Arlington, VA, USA, 457–473.

ZAKI, M.J., PARTHASARATHY, S., OGIHARA, M. and LI, W. (1997): New algorithms for fast discovery of association rules. In *3rd International Conference on Knowledge Discovery and Data Mining (KDD-97)*. AAAI Press, Newport Beach, CA, USA, 283–286.

ZHOU, Z. and EZEIFE, C.I. (2001): A low-scan incremental rule maintenence method. In *14th Canadian Conference on Artificial Intelligence (AI'2001)*, Eds. STROULIA, E. and MATWIN, S., Springer, Ottawa, Canada.

BIOGRAPHICAL NOTES

Dr Aaron Ceglar earned his PhD from Flinders University in 2005 in the field of interactive knowledge discovery. He has a number of conference and journal publications in the fields of knowledge discovery, visualisation and collaborative architectures. He is currently employed as a research fellow at Flinders University.

Aaron Ceglar

Professor John Roddick is currently the SACITT Chair in Information Technology and Associate Head (Research) in the School of Informatics and Engineering at Flinders University. He joined Flinders in 2000 after 10 years at the University of South Australia and five years at the University of Tasmania. This followed 10 years experience in the computing industry as (progressively) a programmer, analyst, project leader and consultant. Professor Roddick has published over 100 papers in a number of areas of computing but specialises in the fields of data mining and knowledge discovery – specifically in temporal and spatial data mining and as applied to medical and health data, and conceptual modelling – specifically in enhanced database systems semantics such as schema evolution and temporal and spatial systems design and use. He holds a PhD from La Trobe University, an MSc from Deakin University and a BSc(Eng)(Hons) from Imperial College, London.

John Roddick