In-Close2, a high performance formal concept miner

ANDREWS, Simon <http://orcid.org/0000-0003-2094-7456>

Published version

ANDREWS, Simon (2011). In-Close2, a high performance formal concept miner. In: ANDREWS, Simon, POLOVINA, Simon, HILL, Richard and AKHGAR, Babak, (eds.) Conceptual Structures for Discovering Knowledge : 19th International Conference on Conceptual Structures, ICCS 2011, Derby, UK, July 25-29, 2011. Proceedings. Lecture notes in computer science (6828). Springer, 50-62.

In-Close2, a High Performance Formal Concept Miner

Simon Andrews

Conceptual Structures Research Group
Communication and Computing Research Centre
Faculty of Arts, Computing, Engineering and Sciences
Sheffield Hallam University, Sheffield, UK
s.andrews@shu.ac.uk

Abstract. This paper presents a program, called **In-Close2**, that is a high performance realisation of the Close-by-One (CbO) algorithm. The design of In-Close2 is discussed and some new optimisation and data preprocessing techniques are presented. The performance of In-Close2 is favourably compared with another contemporary CbO variant called FCbO. An application of In-Close2 is given, using minimum support to reduce the size and complexity of a large formal context. Based on this application, an analysis of gene expression data is presented. In-Close2 can be downloaded from Sourceforge [1].

1 Introduction

The emergence of Formal Concept Analysis (FCA) as a data analysis technology [2, 7, 15] has increased the need for algorithms that compute formal concepts quickly. When Kuznetsov specified, in Close-by-One (CbO) [11], how repeated computations of concepts could be detected using their natural canonicity, it was no longer necessary to exhaustively search through previously generated concepts to determine the uniqueness of a newly generated one. Algorithms have been developed based on CbO, such as In-Close [1], FCbO [9] and FCbO's predecessor [8], which significantly outperform older algorithms that relied on searching [12].

In a recent, albeit small, international competition between concept computing algorithms [14], FCbO took first place and In-Close took second. In-Close2, the program described in this paper, develops the previous implementation of In-Close by adding breadth searching to allow elements of an intent to be inherited, and by implementing some new optimisation and data preprocessing techniques.

2 The In-Close2 Design

In-Close2 has been designed with the fast back-tracking canonicity test of In-Close [1] combined with a dual depth-first and breadth-first approach, used in

[1] http://sourceforge.net/projects/inclose/

algorithms such as FCbO [9], to provide attribute inheritance. In-Close2 incrementally closes parent concepts whilst noting new child extents along the way. The attributes collected during closure are then passed down to each child extent, so that during the closure of a child concept these attributes do not need to be tested for inclusion.

2.1 Structure of In-Close2

In In-Close2 a formal context is represented by a Boolean matrix, I, with m rows, representing a set of objects $\{0, 1, ..., m-1\}$, and n columns, representing a set of attributes $\{0, 1, ..., n-1\}$. For an object i and an attribute j, $I[i][j] = true$ says that object i has attribute j.

A formal concept is represented by an extent, $A[r]$ (an ordered list of objects), and an intent, $B[r]$ (an ordered list of attributes), where r is the concept number (index). For example, if $B[r] = (3, 5, 7)$, $B[r][2] = 7$. For the purposes of the following pseudocode, $A[r]$ and $B[r]$ will be treated as sets, where convenient. Thus, $B[r] \cup \{j\}$ appends attribute j to $B[r]$.

In the algorithm, there is a current attribute, j, the index of the parent concept, r, and a global index of the *candidate* new concept, r_{new}. The candidate concept is instantiated if it is canonical.

There are two procedures, `InCloseII(`r`,`y`)` and `IsCanonical(`r`,`r_{new}`,`j`)`, where y is a starting attribute. The supremum is the concept with index 0 and is initialised as $A[0] = (0, 1, ..., m-1)$, $B[0] = \emptyset$. Initially, $r_{new} = 1$ and the invocation of `InCloseII` is `InCloseII(0,0)`.

The pseudocode is presented below, with a line-by-line explanation.

2.2 Explanation of main procedure, `InCloseII`

The procedure iterates across the context from y, forming intersections between the current extent and the next attribute extent. When the current extent is found in an attribute extent, that attribute is added to the current intent. When a different intersection results, its canonicity is tested to determine if it is a new extent. If it is new, it is added to the children of the current concept for closing later. The current intent is inherited by the children by passing it down though the recursion.

Lines 2 and 3 - Initially there are no children of the current concept.

Line 4 - Iterate over the context, starting at attribute y.

Line 5 - If the next attribute, j, is not an inherited attribute then...

Lines 6 to 9 - ...form a new extent by intersecting the current extent with the attribute extent of j.

Line 10 - If the new extent is the same as the current extent then...

Line 11 - ...add attribute j to the current intent.

Line 13 - Else if the new extent is canonical then...

Line 14 - ...add the starting attribute of the child concept,...

Line 15 - ...add the index number of the child concept,...

```
InCloseII(r, y)
1  begin
2      jchildren ← ∅;
3      rchildren ← ∅;
4      for j ← y upto n − 1 do
5          if j ∉ B[r] then
6              A[r_new] ← ∅;
7              foreach i in A[r] do
8                  if I[i][j] then
9                      A[r_new] ← A[r_new] ∪ {i};
10             if A[r_new] = A[r] then
11                 B[r] ← B[r] ∪ {j};
12             else
13                 if IsCanonical(r, r_new, j) then
14                     jchildren ← jchildren ∪ {j};
15                     rchildren ← rchildren ∪ {r_new};
16                     B[r_new] ← B[r] ∪ {j};
17                     r_new ← r_new + 1;
18     for k ← 0 upto |jchildren| − 1 do
19         InCloseII(rchildren[k], jchildren[k] + 1);
20 end
```

Line 16 - ...inherit the current intent and...

Line 17 - ...increment the concept index.

Line 18 - Iterate across the children...

Line 19 - ...closing each by passing the concept number and next attribute to InCloseII.

2.3 Explanation of procedure `IsCanonical`

The procedure searches backwards in the context for the new extent, skipping attributes that are part of the current intent.

Line 2 - Starting at one less than the current attribute, j, iterate backwards across the context.

Line 3 - If the next attribute, k, is not part of the current intent then...

Lines 4 and 5 - ...intersect the new extent with the attribute extent of k.

Line 6 - If the extent is found, stop searching and return $false$ (the new extent is not canonical).

Line 7 - If this line is reached, the new extent has not been found, so return *true* (the new extent is canonical).

IsCanonical(r, r_{new}, j)

```
   Result: Returns false if A[r_new] is found, true if not found
1  begin
2      for k ← j − 1 downto 0 do
3          if k ∉ B[r] then
4              for h ← 0 upto |A[r_new]| − 1 do
5                  if not I[A[r_new][h]][k] then break;
6          if h = |A[r_new]| then return false;
7      return true;
8  end
```

2.4 The effect of attribute inheritance on performance

Table 1 compares the number of intersections performed by In-Close2 (lines 6-17 of `InCloseII`, above), with and without attribute inheritance, using three well-known public data sets from UCI [4]. An 'extended' formal context of the UCI Adult data set is used that includes scaled continuous attributes from the data and was created using a context creation tool called FcaBedrock [3][2]. The context for the UCI Internet Ads data set used here also includes scaled continuous attributes, created using the same tool. Hence both the extended Adult and the Internet Ads contexts contain more attributes and concepts than is typically found in other publications. The times, in seconds, are also given. The experiments were carried out using a standard Windows PC with an Intel E4600 2.39GHz processor with 32KB of level one data cache and 3GB of RAM. Note that the times in Table 1 are taken from a version of In-Close2 that incorporates the data preprocessing and optimisation techniques presented later in this paper.

Table 1. Number of intersections performed and time taken by In-Close2, with and without attribute inheritance.

UCI Dataset	Mushroom	Adult	Internet Ads
$\|G\| \times \|M\|$	$8,124 \times 125$	$32,561 \times 124$	$3,279 \times 1,565$
#Concepts	226,921	1,388,468	16,570
no inheritance: #intersections	2,729,388	4,644,001	1,953,155
time	0.824	3.323	0.345
inheritance: #intersections	1,193,779	2,919,135	1,925,734
time	0.568	2.632	0.323

[2] http://sourceforge.net/projects/fcabedrock/

3 Data Preprocessing and Optimisation Techniques for Efficient use of Cache Memory

3.1 Physical Column Sorting

The well-known technique of sorting context columns in ascending order of support is implemented in In-Close2. The typical approach is to sort pointers to the columns, rather than the columns themselves, as this takes less time. However, in In-Close2, the columns are physically sorted to make more efficient use of cache memory. If data is contiguous in RAM, cache lines will be filled with data that are more likely to be used when carrying out column intersections in `InCloseII` and when finding an already closed extent in `IsCanonical`. This significantly reduces level one data cache misses, particularly when large contexts are being processed. The overhead of physically sorting the context is outweighed by the saving in memory loads. A comparison between logical and physical column sorting is given in Table 2. The figures for level one data cache misses (L1-DCM) were measured using Intel's V-Tune profiling tool.

Table 2. L1 data cache misses and time taken by In-Close2, comparing logical and physical column sorting.

UCI Dataset	Mushroom	Adult	Internet Ads
$\|G\| \times \|M\|$	$8,124 \times 125$	$32,561 \times 124$	$3,279 \times 1,565$
#Concepts	226,921	1,388,468	16,570
logical sort: L1-DCM	67,300,000	617,300,000	47,000,000
time	0.922	4.563	0.362
physical sort: L1-DCM	33,900,000	252,000,000	32,100,000
time	0.743	3.104	0.341

3.2 Reducing Row Hamming Distance

The Hamming distance between two strings of equal length is the number of positions at which the corresponding symbols are different [6]. In bit-strings this is the number of positions in the bit-strings at which 0s and 1s differ. In In-Close2, after physical column sorting, the *rows* are then also physically sorted to reduce the Hamming distance between consecutive rows. By treating rows as unsigned binary integers, sorting them numerically minimises the number of bits that change from row to row (see Figure 1). This increased uniformity in the data significantly reduces level one data cache misses (see Table 3).

3.3 Using a Bit-Array

The well-known technique of storing the context as a bit-array, rather than an array of bool (byte) data, is implemented in In-Close2. The usual reason for this

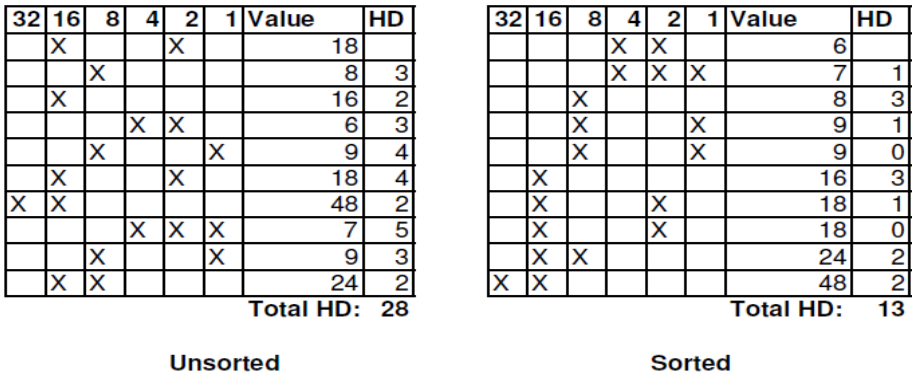

Fig. 1. Row sorting reduces Hamming distance (HD)

Table 3. L1 data cache misses and time taken by In-Close2, with and without reduced Hamming distance.

UCI Dataset	Mushroom	Adult	Internet Ads
$\|G\| \times \|M\|$	$8,124 \times 125$	$32,561 \times 124$	$3,279 \times 1,565$
#Concepts	226,921	1,388,468	16,570
HD not reduced: L1-DCM	33,900,000	252,000,000	32,100,000
time	0.743	3.104	0.341
HD reduced: L1-DCM	10,200,000	50,400,000	21,900,000
time	0.568	2.632	0.323

is that it allows larger contexts to be stored in RAM. However, in the implementation of In-Close2, because the array is sorted physically there is a further improvement in the efficiency of the cache. Although there is an overhead in the extra code required to access individual bits, once contexts have become too large to comfortably fit into cache memory in one-byte data form, this is outweighed by the efficiency gained in the use of the cache. Using two versions of In-Close2, one implementing the context as a bool-array and the other implementing the context as a bit-array, the point at which this occurs is clearly visible in Figure 2. Using a context density of 1% and 200 attributes, the number of objects was increased from 5000 to 25000. With fewer than 10,000 objects, the bool-array implementation was quicker. With more than 10,000 objects, the bit-array implementation was quicker.

4 In-Close2 Performance Evaluation

A number of experiments were carried out to compare In-Close2 with FCbO. An implementation of FCbO was supplied by an author of FCbO, with the understanding that it was a highly optimised version, but with only some details

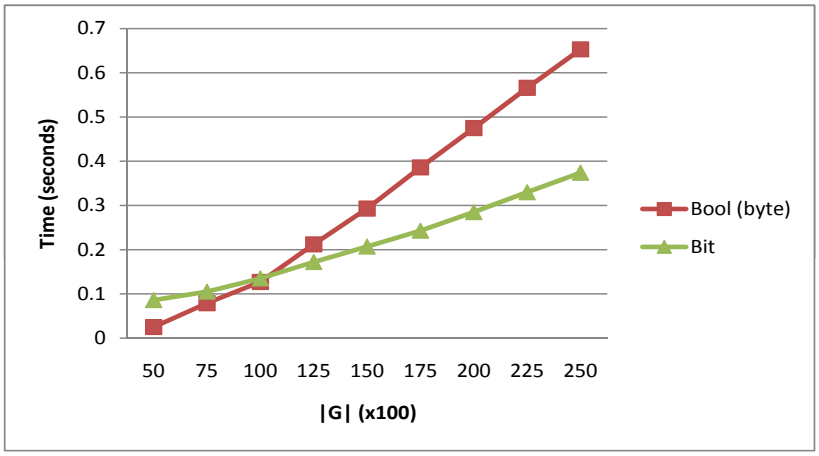

Fig. 2. Comparison of performance between bool and bit-array context data

about the optimisations used. In testing, it was a faster version of FCbO than the one in the competition at ICCS 2010 [14].

Another promising program, called AddExtent, was also tested in the experiments. This program is an attribute-incremental implementation of an algorithm called AddIntent [16] and was supplied by an author of AddIntent.

Experiments were carried out using the three well known public data sets already mentioned in this paper, three artificial data sets and several series of random data sets. The public data sets allowed the programs to be compared under real conditions, the artificial data sets were a means of simulating real conditions but with larger data sizes and the random data sets allowed controlled comparison over key variables: number of attributes, context density and number of objects.

The experiments were carried out using a standard Windows PC with an Intel E4600 2.39GHz processor with 32KB of level one data cache memory and 3GB of RAM. The times for the programs are total times to include data preprocessing. The results are given below.

4.1 Public data set experiments

The results of the public data set experiments are given in Table 4. There was no clear winner; each of the programs performing best with one of the data sets. The largest of the contexts was that of the Adult data set, and here In-Close2

significantly outperformed the other two. The strong performance of In-Close2 with regard to large context size is borne out by the results of the artificial and random data set experiments that follow.

Table 4. UCI data set results (timings in seconds).

	Mushroom	Adult	Internet Ads
$\|G\| \times \|M\|$	$8,124 \times 125$	$32,561 \times 124$	$3,279 \times 1,565$
Density	17.36%	11.29%	0.97%
#Concepts	226,921	1,388,469	16,570
AddExtent	5.787	72.080	0.324
FCbO	0.508	5.687	0.812
In-Close2	0.568	2.365	0.328

4.2 Artificial data set experiments

The following artificial data sets were used:

M7X10G120K - a data set based on simulating many-valued attributes. The scaling of many-valued attributes is simulated by creating 'blocks' in the context containing disjoint columns. There are 7 blocks, each containing 10 disjoint columns.

M10X30G120K - a similar data set, but with a context containing 10 blocks, each with 30 disjoint columns.

T10I4D100K - an artificial data set from the FIMI data set repository [5].

In these experiments, only times for FCbO and In-Close2 are given (Table 5), because times for AddExtent were very large. For each of the three artificial data sets, In-Close2 significantly outperformed FCbO.

Table 5. Artificial data set results (timings in seconds).

	M7X10G120K	M10X30G120K	T10I4D100K
$\|G\| \times \|M\|$	$120,000 \times 70$	$120,000 \times 300$	$100,000 \times 1,000$
Density	10.00%	3.33%	1.01%
#Concepts	1,166,343	4,570,498	2,347,376
FCbO	4.281	28.812	40.765
In-Close2	2.233	20.648	24.277

4.3 Random data set experiments

Three series of random data experiments were carried out to compare the performance of In-Close2 and FCbO, testing the affect of changes in the number of attributes, context density, and number of objects:

Attributes series - with 1% density and 10,000 objects, the number of attributes was varied between 400 and 1,800 (Figure 3). In-Close2 significantly outperformed FCbO, increasingly so as the number of attributes increased.

Density series - with 200 attributes and 10,000 objects, the density of 1s in the context was varied between 1 and 9% (Figure 4). The performance was simliar, although In-Close2 was slightly faster with densities below 7% and FCbO outperformed In-Close with densities greater than 7%, increasingly so as the density increased.

Objects series - with 1% density and 200 attributes, the number of objects was varied between 100,000 and 500,000 (Figure 5). In-Close2 significantly outperformed FCbO, consistantly so as the number of objects increased.

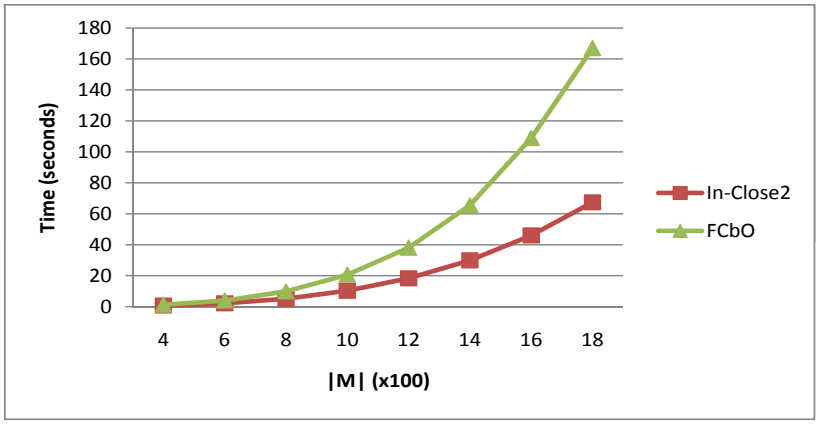

Fig. 3. Comparison of performance with varying number of attributes

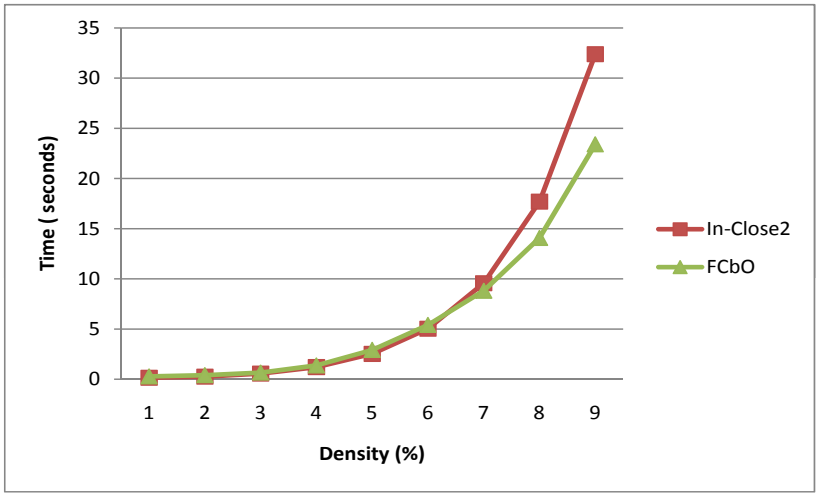

Fig. 4. Comparison of performance with varying context density

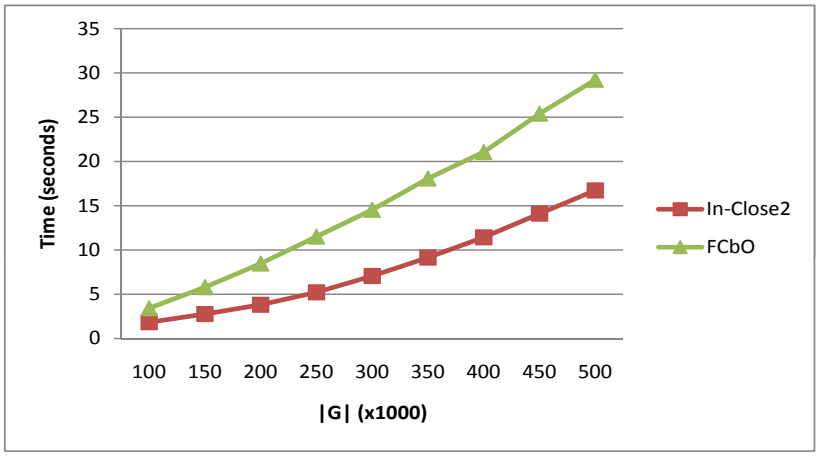

Fig. 5. Comparison of performance with varying number of objects

5 Application of Context Reduction by In-Close2 in Gene Co-expression Analysis

A feature of In-Close2 is to use the well-known idea of minimum support to reduce the size and complexity of formal contexts. A minimum support for both objects and attributes can be specified so that only concepts that satisfy the support will be mined. In-Close2 then outputs a reduced context excluding all objects and attributes that are not part of concepts that satisfy the support. This allows a readable concept lattice to be produced from a large and complex context.

This technique was applied to a data set produced in collaboration with Herriot-Watt University, Edinburgh, Scotland, UK, from the EMAGE Edinburgh Mouse Atlas Project database. The dataset consisted of mouse gene expression data for 6,838 genes in 2,335 mouse tissues (coded as so-called EMAP numbers). There were seven levels of strength of gene expression in the tissues, ranging from 'not detected' to 'strong'. By interpreting a gene as a formal object and a combination of tissue with level of expression as a formal attribute, this data was converted into a formal context using the context creator tool, FcaBedrock. In the context created, In-Close2 detected 208,378 concepts, each representing a gene co-expression. By specifying a minimum support of 14 for genes and 18 for tissue/levels (through a process of trial and error), 13 concepts were detected that satisfied the support, and the context became reduced to 24 objects and 14 attributes. The reduced context is shown in Figure 6.

If a technique such as fault tolerance [13] was applied to this context, so that, for example, the 'missing' relation *(Tgfbi, EMAP:8146-strong)* was assumed to exist, it could be argued that the context should be approximated to a single concept of gene co-expression; all 24 genes being strongly expressed in all 14 tissues. Alternatively, the 'missing' relations could be investigated by examining the original data, where it was found, for example, that there was no record for gene *Tgfbi* in tissue *EMAP:8146* (perhaps indicating that such an experiment has not yet been carried out). On the other hand, a record for gene *Dnajc18* in tissue *EMAP:8349* did exist, stating a 'moderate' level of detection; a good enough indication, perhaps, that this gene/tissue pair should be part of this concept of co-expression.

It also became interesting to discover what bones are missing from the co-expression and why. There are bones in the skull for which there were no data recorded for particular genes, but when an image of the embryo from the relevant experiment was subsequently examined, it was clear that these bones *did* have expression in them. Thus, another possible use for this FCA technique is in inferring the most likely level of expression for a gene-tissue pair when data is missing.

It was also noted that the tissues are mostly bones in the skull and that they are all from the same development stage of the mouse embryo (*Theiler Stage* 23). Further biological investigation is now required to determine the significance of this co-expression; what is happening in the development of the mouse skull at Theiler Stage 23?

Gene Co-Exp	EMAP:8385-strong	EMAP:8339-strong	EMAP:7371-strong	EMAP:7204-strong	EMAP:8360-strong	EMAP:8359-strong	EMAP:8146-strong	EMAP:7847-strong	EMAP:8389-strong	EMAP:7364-strong	EMAP:7363-strong	EMAP:7749-strong	EMAP:8371-strong	EMAP:8394-strong
Mapk8ip2	×	×	×	×	×	×	×	×	×	×	×	×	×	×
Tgfbi	×	×	×	×	×	×		×	×	×	×	×	×	×
Zcchc6	×	×	×	×	×	×	×	×	×	×	×	×	×	×
Brpf3	×	×	×	×	×	×	×	×	×	×	×	×	×	×
Caly	×	×	×	×	×	×	×	×	×	×	×	×	×	×
Bcl9l	×	×	×	×	×	×	×	×	×	×	×	×	×	×
Zc3h18	×	×	×	×	×	×	×	×	×	×	×	×	×	×
Dnajc18	×	×	×	×	×	×	×	×	×	×	×	×	×	
H2-T22	×	×	×	×	×	×	×	×	×	×	×	×	×	×
Colec12	×	×	×	×	×	×		×	×	×	×	×	×	×
Wwp2	×	×	×	×	×	×	×		×	×	×	×	×	×
Emp3	×	×	×	×	×	×	×	×	×	×	×	×	×	×
Tcam1	×	×	×	×	×	×	×	×	×	×	×	×	×	×
Papss2	×	×	×	×	×	×	×	×	×	×	×		×	×
Ubxn10	×	×	×	×	×	×	×	×	×	×	×	×	×	×
Cytl1	×	×	×	×	×	×	×	×	×	×	×	×	×	×
BC024814	×	×	×	×	×	×	×	×	×	×	×	×	×	×
Plekhb1	×	×	×	×	×	×	×	×	×	×	×	×	×	×
Apitd1	×	×	×	×	×	×	×	×	×	×	×	×	×	×
Cebpz	×	×	×	×	×	×	×	×	×	×	×	×	×	×
1110017D15Rik	×	×	×	×	×	×	×	×	×	×	×	×	×	×
Copb1	×	×	×	×	×	×	×	×		×	×	×	×	×
Unc5cl	×	×	×	×	×	×	×	×	×	×	×	×	×	×
Haus4	×	×	×	×	×	×	×	×	×	×	×	×	×	×

Fig. 6. Reduced context of gene expression in mouse tissues

6 Conclusion

In-Close2 has been shown to outperform FCbO and AddExtent, apart from where the formal context was a combination of dense ($> 7\%$) *and* random. In such cases, FCbO was the best performer. This is probably because, although FCbO closes a concept before testing its canonicity, it does it very quickly by using bitwise operations on 32 bit data when performing closure (i.e., 32 comparisons of context table cells are performed at a time). In-Close2 tests each cell individually in its backtracking canonicty test, which avoids having to close a concept before testing its canonicty, but can be slower if the context is random and dense. This is probably because it is more likely that several cells will have common crosses tested before the comparison fails. In real and artificial data sets, density seems to have less of a detrimenatal effect on In-Close2, probably becuase of the natural predominance of patterns over random noise, reducing the number of 'near misses' when testing canonicity. Further investigation would be required to confirm this hypothesis.

Optimisations have been shown in this paper that significantly improve the performance of In-Close2, by making better use of cache memory. These optimisations are quite general and could be applied to many algorithms that operate on Boolean data. It is not clear from the authors of FCbO the extent to which similar techniques have been used in their implementation of FCbO, although they stated that their implementation was "highly tuned".

The usefulness of a high-performance concept miner has been shown by the analysis of a large, complex, context of gene expression data using the technique of context reduction though minimum support. A large co-expression of genes in the skull bones of a mouse embryo has been discovered. Uses for FCA in focusing further investigation and inferring missing data have been shown.

Acknowledgments The help is acknowledged of Kenneth McLeod and Albert Berger of the School of Mathematical and Computer Sciences, Herriot-Watt University, Edinburgh, Scotland, UK, in the preparation and analysis of the mouse gene expression data.

This work is part of the CUBIST project ("Combining and Uniting Business Intelligence with Semantic Technologies"), funded by the European Commission's 7th Framework Programme of ICT, under topic 4.3: Intelligent Information Management.

References

1. S. Andrews. In-close, a fast algorithm for computing formal concepts. In S. Rudolph, F. Dau, and S. O. Kuznetsov, editors, *ICCS 2009*, volume 483 of *CEUR WS*. http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-483/, 2009.
2. S. Andrews and C. Orphanides. Analysis of large data sets using formal concept lattices. In Kryszkiewicz and Obiedkov [10], pages 104–115.

3. S. Andrews and C. Orphanides. Fcabedrock, a formal context creator. In M. Croitoru, S. Ferre, and D. Lukose, editors, *ICCS 2010*, volume 6208/2010 of *LNCS*. Springer, 2010.
4. A. Frank and A. Asuncion. UCI machine learning repository: http://archive.ics.uci.edu/ml, 2010.
5. B. Goethals. Frequent itemset implementations (fimi) repository: http://fimi.cs.helsinki.fi/, 2010.
6. R. W. Hamming. Error detecting and error correcting codes. *Bell System Technical Journal*, 29(2):147–160, 1950.
7. M. Kaytoue, S. Duplessis, S. O. Kuznetsov, and A. Napoli. Two fca-based methods for mining gene expression data. In S. Ferre and S. Rudolph, editors, *ICFCA 2009*, volume 5548 of *LNAI*. Springer, 2009.
8. P. Krajca, J. Outrata, and V. Vychodil. Parallel recursive algorithm for fca. In R. Belohavlek and S. O. Kuznetsov, editors, *CLA 2008*, 2008.
9. P. Krajca, V. Vychodil, and J. Outrata. Advances in algorithms based on cbo. In Kryszkiewicz and Obiedkov [10], pages 325–337.
10. M. Kryszkiewicz and S. Obiedkov, editors. *7th International Conference on Concept Lattices and Their Applications, CLA 2010, Seville*. University of Sevilla, 2010.
11. S. O. Kuznetsov. Learning of simple conceptual graphs from positive and negative examples. In J. M. Zytkow and J. Rauch, editors, *PKDD'99*, volume 1704 of *Lecture Notes in Computer Science*, pages 384–391. Springer, 1999.
12. S.O. Kuznetsov and S.A. Obiedkov. Comparing performance of algorithms for generating concept lattices. *Journal of Experimental and Theoretical Artificial Intelligence*, 14:189–216, 2002.
13. R. G. Pensa and J-F. Boulicaut. Towards fault-tolerant formal concept analysis. In S Bandini and S Manzoni, editors, *Advances in Artificial Intelligence, 9th Congress of the Italian Association for Artificial Intelligence*, volume 3673 of *Lecture Notes in Computer Science*. Springer-Verlag Berlin Heidelberg, 2005.
14. U. Priss. Fca algorithms: http://www.upriss.org.uk/fca/fcaalgorithms.html, 2009.
15. T. Tanabata, K. Sawase, H. Nobuhara, and B. Bede. Interactive data mining for image databases based on fca. *Journal of Advanced Computational Intelligence and Intelligent Informatics*, 14(3):303–308, 2010.
16. D. van der Merwe, S. A. Obiedkov, and D. G. Kourie. Addintent: A new incremental algorithm for constructing concept lattices. In Peter W. Eklund, editor, *ICFCA 2004*, volume 2961 of *Lecture Notes in Computer Science*, pages 372–385. Springer, 2004.