In-Close, a Fast Algorithm for Computing Formal Concepts

Simon Andrews

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 an algorithm, called **In-Close**, that uses *incremental closure* and *matrix searching* to quickly compute all formal concepts in a formal context. **In-Close** is based, conceptually, on a well known algorithm called *Close-By-One*.
The serial version of a recently published algorithm (**Krajca**, 2008) was shown to be in the order of 100 times faster than several well-known algorithms, and timings of other algorithms in reviews suggest that none of them are faster than **Krajca**. This paper compares **In-Close** to **Krajca**, discussing computational methods, data requirements and memory considerations. From experiments using several public data sets and random data, this paper shows that **In-Close** is in the order of 20 times faster than **Krajca**. **In-Close** is small, straightforward, requires no matrix pre-processing and is simple to implement.

1 Introduction

Tables are often used to represent incidence data in the form of binary relations. Columns in the table represent attributes or items and rows represent objects, instances or transactions. A presence of the value 1 ($true$) in row i, column j, says that object i has attribute j. A presence of the value 0 ($false$) in row i, column j, says that object i does not have attribute j. This form of data is the basis for Formal Concept Analysis (FCA) [9, 20, 18], where it is referred to as the *formal context*, and is important in data mining [11]. Data analysis applications are numerous, in fields such as marketing, census analysis and bioinformatics.

A formal description of formal concepts begins with a set of objects G and a set of attributes M. A binary relation $I \subseteq G \times M$ is called the *formal context*. If $i \in G$ and $j \in M$ then iIj says that object i has attribute j. For a set of objects $A \subseteq G$, a derivation operator $'$ is defined to obtain the set of attributes common to the objects in A as follows:

$$A' := \{\ j \in M \mid \forall i \in A : iIj\ \}.$$

Similarly, for a set of attributes $B \subseteq M$, the operator is defined to obtain the set of objects common to the attributes in B as follows:

$$B' := \{\ i \in G \mid \forall j \in B : iIj\ \}.$$

(A, B) is a formal concept iff $A' = B$ and $B' = A$. A is called the *extent* of the formal concept and B is called the *intent*.

The problem associated with the computation of all formal concepts in a formal context is that every combination of columns (or rows) can generate a concept. Only a proportion of these will be closed and the number of formal concepts grows exponentially with the size of the matrix [6]. In addition, many different combinations of columns may share the same rows, leading to the same result being generated many times. This leads to a large number of computations and the searching of large numbers of results for repeats, even for medium-sized data sets. (For the purposes of this paper, 200 attributes and 10,000 objects is an example of a medium-sized data set.) This has led to a variety of interesting approaches to computing formal concepts. This paper presents an algorithm, called **In-Close**, based conceptually on Kuznetsov's *Close-By-One* algorithm [13], as a contender for being the fastest method to date.

2 Computing Formal Concepts

In computing formal concepts, a Boolean matrix is often used to represent the formal context, where rows and columns are, computationally, interchangeable. Thus, in what follows, the same will apply if 'A', 'extent', 'object' and 'row' are substituted for 'B', 'intent', 'attribute' and 'column', and vice-versa.

Reviews of algorithms that take a variety of approaches to generating formal concepts may be found in [2, 14, 21]. Many of these algorithms perform additional tasks, such as constructing the concept lattice (Hasse diagram), and not all were designed with fast performance as a requirement, nor with the intention to be applied to the size of data sets suggested here.

Other work has drawn from different areas where formal concepts are important: in data mining [21], where formal concepts may be referred to as *closed frequent itemsets* and in Boolean factor analysis where they are *optimal factors* [3]. There is also a strong correlation with algorithms determining the rectangular groups used to simplify Boolean expressions in Karnaugh maps [22]. Elsewhere, algorithms have referred to formal concepts as *maximal rectangles* [4], although this term has also been used when referring to the largest contiguous empty rectangles in a Boolean matrix [7].

Fundamental to the performance of algorithms to generate formal concepts is how they deal with the complexity of the computation. As mentioned in section 1, many values of B can generate the same value of A, and only the largest (closed) value of B is part of a formal concept. Taking the matrix in Figure 1, $A = \{4, 5\}$ will be generated for three values of B: $\{2\}$, $\{2, 4\}$ and $\{4\}$ where $\{2, 4\}$ is the closed B. As As are generated, a common method is to search previous results and discard repeats. A key to good peformance is how efficiently previous results are searched. Lindig's algorithm [15], and others like it, use a search tree, or *trie*, to quicky find repeated results. Others use a hash function where the cardinality

of results is used to divide them into groups, thus narrowing the search [10]. Berry's algorithm [4] avoids searching by pruning the matrix before generating concepts from it - eliminating columns that are strict subsets of other columns and pairing those that are equal.

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 \end{bmatrix}$$

Fig. 1. A Boolean matrix.

Another way of dealing with complexity is to artificially restrict the result space. Some algorithms discard results that have less than a user-specified size of A, thereby significantly reducing the computation [21]. This size is sometimes given as a proportion of $|G|$ and is referred to in data mining as the *minimum support* required by an itemset for it to be of significance. It is argued that formal concepts with small extents may be of less interest, or that they may be the result of small errors in the data set, and not generating them is therefore sensible. It is acknowledged that this is not always the case and that useful results may be lost by 'over-pruning' the computation [17]. Similar approaches to reducing computation have been explored that discard results with small $|B|$ or small $|B \times A|$, or use some other constraint ('powers of 2' rectangles in Karnaugh maps, for example).

In all cases, concepts need to be closed by the algorithm at some point. One approach is to take each new extent and, from it, determine the corresponding intent. This is the approach used by **Krajca** but not by **In-Close**, which uses an incremental approach to closure, as will be seen later.

2.1 The Lexicographical Order of Concepts

Ganter showed how an *order* of concepts could be used to circumvent the need to explicitly search for repeated results [8]. In mathematics, combinations have a lexicographical order, or *cannon*, where $\{1, 2, 3\}$ comes before $\{1, 2, 4\}$, for example, and $\{1, 2, 6\}$ comes before $\{1, 3\}$. If concepts are generated in this order for A, or for B, it is possible to determine if a concept is new by examining its cannonicity. Ganter's algorithm and subsequent algorithms, such as Kuznetsov's *Close-By-One* [13], maintain a *current object*. The concept next generated is new (cannonical) if its extent contains no object preceeding the current object. For example, if the current object is 2 and the extent then generated is $\{1, 2, 4\}$, the corresponding concept is not cannonical and can be discarded.

One way of implementing this approach is to use a recursive function to generate combinations of objects or attributes in lexicographical order. Both **Krajca** and **In-Close** use such a function to generate attribute combinations, and this will be the approach described in what follows. A loop is used recursively to generate the combinations as follows:

for $j \leftarrow y$ **upto** $n - 1$

where $n = |G|$. For the first iteration of the loop, $y = 0$. Recursive iterations of the loop are carried out, each time setting the initial value, y, to $j + 1$.

The general method is to add attributes, one at a time, to a current intent. As each new attribute is added, the corresponding extent is computed. This is achieved by intersecting the current extent, A, with the *attribute extent*, $\{j\}'$, of the new attribute. i.e. $newA = A \cap \{j\}'$. The new extent can be one of three outcomes:

1. The empty set.
2. A non-empty set smaller than A.
3. The same set, i.e. $A \cap \{j\}' = A$.

Krajca and **In-Close** deal with these outcomes in different ways.

2.2 The Krajca Method

A recent algorithm by Krajca, Outrata and Vychodil [12, 16, 19], referred to here as **Krajca**, was shown to be particularly efficient. They have presented a serial and a parallel version of their algorithms but only the serial version (and results from it) is considered in this paper. Three real data sets were used, by them, in a comparative experiment with some well known algorithms (already mentioned here) that are representative of several mainstream approaches. On average, **Krajca** was 139 times faster than Ganter's algorithm [8], 406 times faster than Lindig's algorithm [15] and 106 times faster than Berry's algorithm [4]. An inspection of timings in the algorithm reviews cited at the start of this section suggests that none of the algorithms reviewed are faster than the serial version of **Krajca** (bearing in mind the caveats mentioned).

The process **Krajca** uses is as follows: when an attribute is added to B, **Krajca** computes closure of the intent by iterating across all the rows in $A \cap \{j\}'$, identifying all truthfully associated columns. The algorithm then utilises the lexicographical order of attribute generation to quickly determine the newness of the intent. It compares the newly closed intent with the unclosed version, and, if the closed version agrees with the unclosed version, up to the current attribute, the result must be new. The presence of an earlier attribute in the closed version indicates that the concept must have already been generated. In this case, the recursion is skipped.

Krajca makes efficient use of outcome 3. After closure, if the next attribute to be included is already in the intent, the recursion can be skipped. This can significantly reduce the number of repeated concepts.

However, although **Krajca** is efficient in several ways, completing closure of repeats requires a significant amount of computation, particularly if there are many rows to iterate across. **Krajca**'s way of determining outcome 1 is rather inefficient, too. At the start of the process of closure, **Krajca** sets B to M and outcome 1 is then deemed to have occured if B is still equal to M, after the closure. There is also an issue arising from the fact that $B = M$ is also true when there is an object with all the attributes. The test in the algorithm should probably be 'if $A = \emptyset$', but, because **Krajca** represents A and B as Boolean lists, this would take more time.

2.3 The In-Close Method

In-Close uses the lexicographic approach for implicitly searching, but avoids the overheads of computing repeated closures. Instead, it completes closure incrementally, and only once per concept, as it iterates across the attributes, using outcome 3 as an indicator to add the new attribute to the current intent and prune the recursion at that point. Iteration across the attributes continues, adding a new attribute and pruning its branch each time outcome 3 occurs, until the concept is closed. Figure 2 shows this happening for a concept with intent $\{0, 1, 3, ...\}$: (i) Attribute 1 joins 0 and the branch of recursion is pruned. (ii) Attribute 2 is not part of B, so either the branch is taken (if A is not empty), or the branch is pruned (if A is empty). (iii) Attribute 3 joins 0 and 1 and the branch of recursion is pruned. (iv) On to the next attribute.

Outcome 1 is quickly dealt with by testing the size of the new extent immediately after the intersection is made. If the new extent is empty, the recursion is skipped.

Outcome 2 requires A to be searched for in previous results. This is done implicitly by examining the Boolean matrix, paying attention to its cannonicity. The method is to attempt to match the rows in A with those in any previous column, *excluding columns in* B (Figure 3). If a match is found, then A will have already been generated. In this case, the recursion is skipped. If A is new, the next branch of recursion is taken to begin closure of the new concept.

Put formally, if B is the current intent, j is the new attribute and A is the resulting extent, A is not cannonical if

$$\exists k \in M - B \mid k < j \wedge \forall i \in A : iIk.$$

In **In-Close**, the closures are implicit, in that we can only say that all concepts with intents beginning with attribute j are closed once the algorithm has completed that breadth of computation (and moved on to intents beginning with $j+1$). The benefit of this is that concepts are closed only once per concept and that the test of cannonicity requires iteration over a relatively small portion of the matrix. Thus, closure and searching are carried out efficiently, with comparatively few redundant operations.

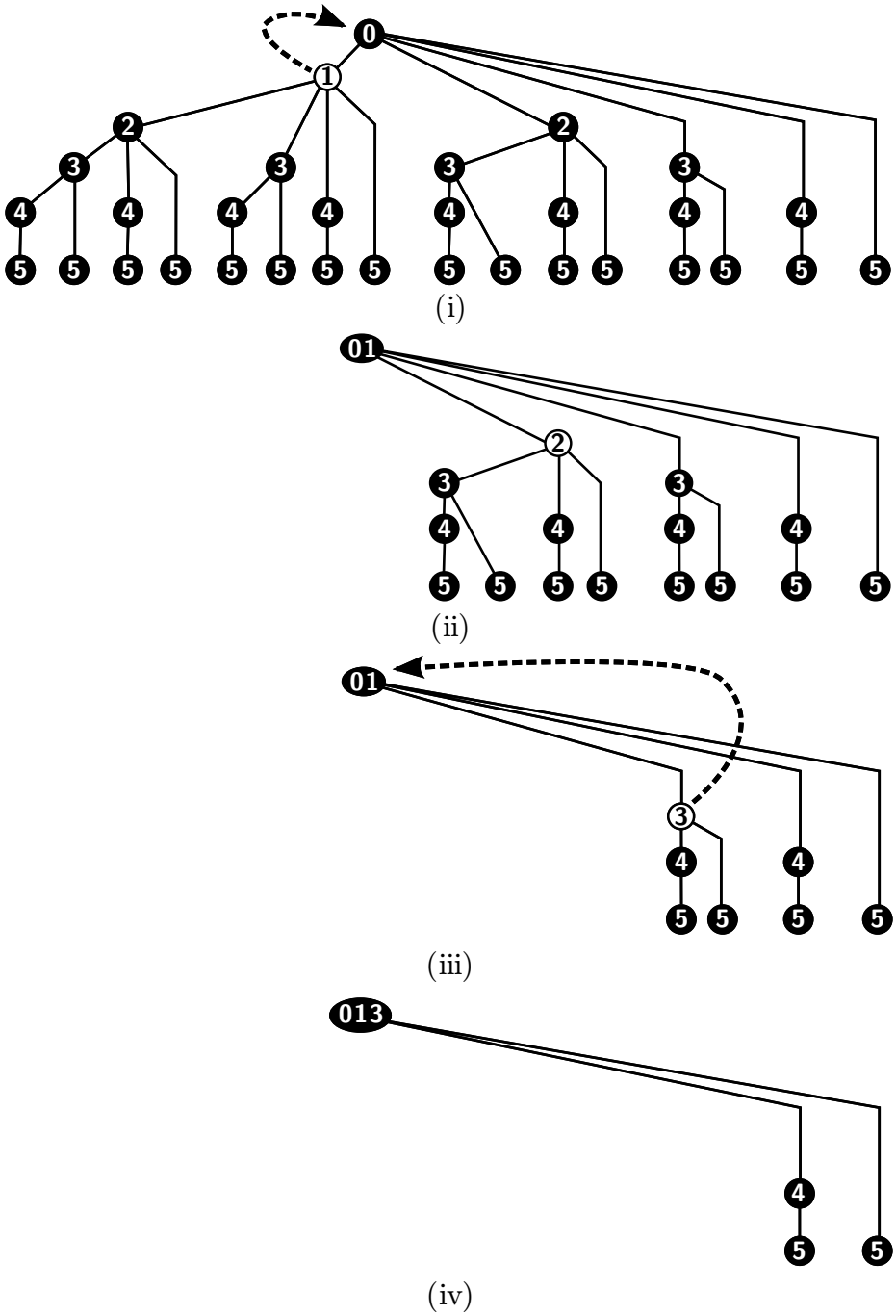

Fig. 2. Incrementally closing intent $B = \{0, 1, 3, ...\}$.

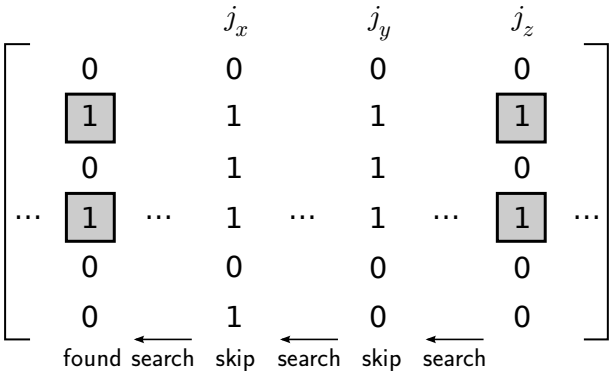

Fig. 3. Checking cannonicity by searching for rows $A = \{1, 3\}$, skipping columns $B = \{j_x, j_y, j_z\}$.

3 The In-Close Algorithm

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 of objects and attributes, respectively, 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 current concept being closed, r, and a global index of the candidate new concept, r_{new}.

There are two procedures, `InClose(`r`,`y`)` and `IsCannonical(`r`,`y`)`, where y is a starting column. `InClose(`r`,`y`)` means 'incrementally close concept r, beginning at attribute y'.

The supremum is the concept with index 0 and is initialised as $A[0] = (0, 1, ..., m-1)$ and $B[0] = \emptyset$. Initially, $r_{new} = 0$ and the invocation of `InClose` is `InClose(0,0)`.

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

3.1 Explanation of `InClose`

Line 2 - Begin a new concept.
Line 3 - Iterate over attributes, starting at attribute y.
Lines 4 to 7 - Form an extent, $A[r_{new}]$, by computing $A[r] \cap \{j\}'$.
Line 8 - If the extent is empty (outcome 1), skip the recursion and move on to

```
InClose(r, y)
1  begin
2      r_new ← r_new + 1;
3      for j ← y upto n − 1 do
4          A[r_new] ← ∅;
5          foreach i in A[r] do
6              if I[i][j] then
7                  A[r_new] ← A[r_new] ∪ {i};
8          if |A[r_new]| > 0 then
9              if |A[r_new]| = |A[r]| then
10                 B[r] ← B[r] ∪ {j};
11             else
12                 if IsCannonical(r, j − 1) then
13                     B[r_new] ← B[r] ∪ {j};
14                     InClose(r_new, j + 1);
15 end
```

the next attribute. Note that a value > 0 can be used if concepts with extents below a certain size are not of interest (the notion of *minimum support* described in section 2).
Lines 9 and 10 - If the extent is unchanged (outcome 2) then add the attribute to $B[r]$ (incremental closure), skip the recursion and move on to the next attribute.
Lines 11 and 12 - Otherwise, the extent must be a smaller, non-empty intersection (outcome 3), so call `IsCannonical` to see if it has already been generated.
Lines 13 and 14 - If the extent is not found, the concept is cannonical, so initialise $B[r_{new}]$ and call `InClose` to begin its closure. Otherwise, if the extent is found, skip the recursion and move on to the next attribute.

3.2 Explanation of `IsCannonical`

The procedure searches for $A[r_{new}]$ in the blocks of columns in-between those in $B[r]$ (Figure 2).
Line 2 - Take each column in $B[r]$ (in reverse order) and
Line 3 - iterate down to that column from a starting column.
Lines 4 and 5 - In each column, try to match the extent. i.e. is $A[r_{new}] \subseteq \{j\}'$?
Line 6 - If the extent is found, stop searching and return $false$ (it is not cannonical).
Line 7 - Skip the column just iterated down to, ready to iterate the next block of columns.
Lines 8 - 11 - Finally (if not already found), search for the extent in the block of columns down to 0.

IsCannonical(r, y)

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

4 Experimental Evaluation

In-Close and the serial version of **Krajca** were implemented in Microsoft C. The algorithm designs (pseudocode) were at a similar level of abstraction. The implementations were carried out using similar constructs, without further optimisation and without altering the logic of the designs. Thus, as far as possible, the experiments compared 'like with like'. Two groups of experiments were carried out to compare their performance. One group of experiments was carried out using four publicly available data sets from the UCI Machine Learning Repository [1] and one group using several sets of random data. The UCI data sets were chosen on the basis of suitability in terms of size (thousands of objects, hundreds of attributes) and data type (predominantly Boolean or multivariate). Two of the sets chosen (*Anonymous Microsoft Web Data* and *Mushroom*) were used in the experiments of **Krajca** [12] and so would provide a corroborative facet to the comparison. Multivariate data was converted into corresponding Boolean representations, i.e. an attribute with x values was converted into x Boolean attributes. Continuous data were not used.

The experiments were carried out on a standard Windows-XP PC using an Intel E4600 2.40GHz processor with 3.00 GB of RAM. The formal concepts generated were stored in RAM and subsequently output to a `.text` file as comma separated, 16 bit, attribute and object values in the form $j_0, j_1, ..., j_{|B|-1}, \backslash t$ $i_0, i_1, ..., i_{|A|-1}, \backslash n$. File output was not timed.

4.1 Pre-processing

For the sake of efficiency, in addition to the Boolean matrix, **Krajca** uses an array of ordered lists of objects to represent the input data. This is generated

from the matrix as a pre-processing task. **Krajca** also requires an initial closure, $(\emptyset', \emptyset'')$, to be computed, to provide a starting point. As this is most sensibly done at the same time the array of ordered lists is populated, **Krajca**'s pre-processing has been included in the timings that follow, although it contributes only very slightly.

In-Close requires no pre-processing.

4.2 Public data set experiments

Table 1 lists the timings achieved using the four UCI data sets. The first section of the table gives the name of the data set, the corresponding matrix size and the density of *true* values in the matrix. Any missing values in a data set are taken into account in the calculation of the matrix density. The second section of the table gives the number of formal concepts generated and the size of the resulting `.text` file. The third section gives the time, in seconds, taken by **Krajca** and **In-Close** to generate the concepts and a comparison of their times.

The names of the data sets used are *Anonymous Microsoft Web Data*, *Mushroom*, *Adult* (also known as *Census Income*) and *Internet Advertisements*. Details of the data sets and how they were used is given in the following subsections.

Table 1. UCI data set results (timings in seconds).

Data set	Anon MS-Web	Mushroom	Adult	Internet Ads
$\lvert G\rvert \times \lvert M\rvert$	32711×294	8124×125	48842×96	3279×1555
Density	1.03%	17.36%	8.24%	0.77%
#Concepts	128,380	226,920	68,872	7,680
File KB	9,342	95,499	34,482	735
Krajca	57.72	27.86	14.48	2.30
In-Close	1.42	2.11	1.05	0.23
× faster	40.64	13.20	13.79	10.00

***Anonymous Microsoft Web Data* Data Set** Contains logs of web site areas visited by anonymous users of www.microsoft.com. 32711 logs (objects in the experiment) were sampled, covering 294 areas of the site (attributes in the experiment). Areas are numbered 1000 - 1297, with numbers 1047, 1285, 1286 and 1296 not used. Care was taken in the experiment to exclude these numbers during data acquisition. Instance data is recorded in the data set in the form *user: areas visited*, so there are no missing values.

***Mushroom* Data Set** Contains records from *The Audobon Society Field Guide to North American Mushrooms (1981)*. It describes 8124 mushrooms (4208 ed-

ible, 3916 poisonous) in terms of physical characteristics. There are 22 characteristics recorded as multivariate attributes. This is the list in the form *attribute name*(number of values): *cap-shape*(6), *cap-surface*(4), *cap-color*(10), *bruises?*(2), *odor*(9), *gill-attachment*(4), *gill-spacing*(3), *gill-size*(2), *gill-color*(12), *stalk-shape*(2), *stalk-root*(6), *stalk-surface-above-ring*(4), *stalk-surface-below-ring*(4), *stalk-color-above-ring*(9), *stalk-color-below-ring*(9), *veil-type*(2), *veil-color*(4), *ring-number*(3), *ring-type*(8), *spore-print-color*(9), *population*(6), *habitat*(7). There are 2480 missing values, all for the stalk-root attributes, treated as 0 in the experiment (i.e. a separate Boolean attribute for 'missing' was not included). Thus, there were 125 Boolean attributes in the experiment.

***Adult* Data Set** Contains 48842 instances of 1994 US census data, extracted from the US census bureau database. There are 14 attributes as follows, with multivariate attributes given the number of values in parenthesis: *age*(continuous), *workclass*(8), *fnlwgt*(continuous), *education*(16), *education-num*(continuous), *marital-status*(7), *occupation*(14), *relationship*(6), *race*(5), *sex*(2), *capital-gain*(continuous), *capital-loss*(continuous), *hours-per-week*(continuous), *native-country*(40). Continuous data were not used. Thus, there were 96 Boolean attributes in the experiment. There are 4262 missing values in the data used, treated as 0 in the experiment.

***Internet Advertisements* Data Set** Contains data concerning features on internet pages. There are 1558 attributes: three continuous, 1555 Boolean. The continuous attributes were not used in the experiment. There are 15 missing values in the Boolean data, all for the first Boolean attribute, treated as 0 in the experiment.

4.3 Random Data Experiments

Three experiments were carried out to compare the performance of **In-Close** and **Krajca** while varying $|M|$, $|G|$ and matrix density. Table 2 gives the results for variable $|M|$, Table 3 gives the results for variable $|G|$ and Table 4 gives the results for variable density. In each case the timings and number of concepts stated are an average over a series of sets of random numbers.

Table 2. Variable $|M|$ results (timings in seconds). Density = 2%, $|G|$ = 10,000.

$\|M\|$	200	400	600	800	1000	1500
#Concepts	29,696	111,460	266,091	502,032	832,862	2,096,413
Krajca	1.52	11.68	41.70	106.44	225.89	862.10
In-Close	0.07	0.50	1.69	4.06	8.11	29.49
× faster	21.71	23.36	24.67	26.22	27.85	29.23

Table 3. Variable $|G|$ results (timings in seconds). Density = 5%, $|M| = 200$.

$\|G\|$	10,000	20,000	30,000	40,000	50,000
#Concepts	465,016	1,164,325	1,839,593	2,503,508	3,219,911
Krajca	28.87	187.78	420.05	731.94	1178.41
In-Close	0.99	2.55	4.22	5.94	7.77
× faster	29.16	73.64	99.53	123.22	151.66

Table 4. Variable density results (timings in seconds). $|M| = 100$, $|G| = 2{,}000$

Density	5%	10%	15%	20%
#Concepts	10,623	111,324	683,595	4,043,027
Krajca	0.13	1.71	12.95	88.60
In-Close	0.011	0.155	1.17	8.02
× faster	11.82	11.03	11.07	11.04

5 Conclusion

In-Close significantly outperformed the serial version **Krajca** in all experiments. Published results [12], comparing the serial version of **Krajca** with several well known algorithms, showed **Krajca** to be the fastest, by about 100 times. **In-Close** is, therefore, a contender to be the fastest serial algorithm, to date, for generating all formal concepts in a formal context. Furthermore, the timings here indicate that **In-Close** performs as well, if not better, than the *parallel* version of **Krajca**. For example, in comparable conditions, **Krajca** was timed at 11.466 seconds to compute the *Anonymous MS Web* data set, running on 8 CPUs, compared with 1.42 seconds for **In-Close**.

The variable $|G|$ results (Table 3) highlight a vulnerability of the type of closure used by **Krajca**, where the time taken is exponential to $|A \cap \{j\}'|$. There are significant overheads if closure requires iteration across all $i \in A \cap \{j\}'$.

6 Discussion

Data Sets Problems concerning the comparison of performance using the same data sets have been highlighted by Kuznetsov [14]. For example, the UCI *Mushroom* data set is quoted in this paper as having 125 Boolean attributes whereas elsewhere it has been quoted as having 119 attributes. This is because the 22 multivariate attributes in the data set can be simplified. For example, there are four binary attributes that might be represented as four Boolean attributes (rather than the eight used here). Kuznetsov has suggested that some well recognised data sets could be used, with clearly defined multivariate scalings. One way of doing this would be to implement a Boolean data set repository. As well as providing standardised public data sets, it could be a resource for providing random data sets with varying properties.

Memory Considerations On 32-bit hardware, large Boolean matrices are difficult to store. Because of this, a bit-array version of **In-Close** is proposed. **Krajca** and **In-Close** generate concepts in exponential memory, athough **Krajca** has the advantage that both A and B are static data and can therefore be linearised. In **In-Close**, only A is static, although some savings in B can be made with the use of linked lists, without significant loss of performance. However, a version of **In-Close** is being developed that uses a tree data structure to store concepts in polynomial memory and which produces the corresponding concept lattice.

References

1. Asuncion, A., Newman, D. J.: UCI Machine Learning Repository [http://www.ics.uci.edu/∼mlearn/MLRepository.html]. Irvine, CA: University of California, School of Information and Computer Science (2007).
2. Arevalo, G., Berry, A., Huchard, M., Perrot, G., Sigayret, A.: Performances of Galois Sub-hierarchy-building algorithms. In: Kuznetsov, S.O., Schmidt, S. (eds), *ICFCA 2007*, LNAI 4390, pp. 166180, 2007. Springer-Verlag, Berlin, Heidelberg (2007).
3. Belohlavek, R.: Optimal decompositions of matrices with grades, In: *Intelligent Systems, 2008. IS '08. 4th International IEEE Conference*, vol. 2, pp.15-2-15-7, 6-8 (Sept. 2008).
4. Berry, A., Bordat, J-P., Sigayret, A.: A Local Approach to Concept Generation. In: *Annals of Mathematics and Artifical Intelligence*, Vol. 49, pp. 117-136 (2007).
5. Boulicaut, J-F., Besson, J.: Actionability and Formal Concepts: A Data Mining Perspective. In: Medina, R., Obiedkov, S. (eds.), *Int. Conf. on Formal Concept Analysis*, Lecture Notes on Artifical Intelligence series, pp. 14-31, 2008, Springer-Verlag, Berlin / Heidelberg (2008).
6. Carpineto, C., Romano, G.: Concept Data Analysis: Theory and Application. Wiley (2004).
7. Edmonds, J., Gryz, J., Liang, D., Miller, R.J.: Mining for Empty Rectangles in Large Data Sets. In: *Database Theory - ICDT 2001*, Lecture Notes in Computer Science series, Vol. 1973/2001, pp. 174-188. Springer Berlin / Heidelberg (2001).
8. Ganter, B.: Two Basic Algorithms in Concept Analysis, Technical Report *FB4-Preprint No. 831*, TH Marstadt (1984).
9. Ganter, B., Wille, R: Formal Concept Analysis: mathematical foundations, Springer, Heidelberg 1999.
10. Godin, R., Missaoui, R., Alaoui, H.: Incremental Concept Formation Algorithms Based on Galois Lattices. In: *Computational Intelligence*, Vol. 11(2), pp 246-267, Blackwell Synergy (1995).
11. Han, J., Kamber, M.: Data Mining: Concepts and Techniques. Morgan Kaufman (2001).
12. Krajca, P., Outrata, J., Vychodil, V.: Parallel Recursive Algorithm for FCA. In: Belohlavek, R., Kuznetsov, S.O. (eds.), *Proceeding of the Sixth International Conference on Concept Lattices and their Applications*, pp. 71-82, Palacky University, Olomouc (2008).
13. Kuznetsov, S.O.: Learning of Simple Conceptual Graphs from Positive and Negative Examples. In: *Proceedings of the Third European Conference on Principles of*

Data Mining and Knowledge Discovery, Lecture Notes In Computer Science, Vol. 1704, pp. 384 - 391. Springer-Verlag, London (1999).
14. Kuznetsov, S.O., Obiedkov, S.A.: Comparing Performance of Algorithms for Generating Concept Lattices. In: *Journal of Experimental and Theoretical Artificial Intelligence*, Vol. 14, pp. 189-216 (2002).
15. Lindig, C.: Fast concept analysis. In: *Working with Conceptual Stuctures: Contributions to ICCS 2000*, pp. 152-161, Shaker-Verlag, Aachen (2000).
16. Outrata, J., Vychodil, V.: Fast algorithm for computing maximal rectangles from object-attribute relational data (submitted).
17. Pensa, R.G., Boulicaut, J-F.: Towards Fault-Tolerant Formal Concept Analysis. In: Bandini, S., Manzoni, S. (eds.), *AI*IA 2005: Advances in Artificial Intelligence*, Lecture Notes in Computer Science series, Vol. 3673, pp. 212-223, Springer-Verlag, Berlin / Heidelberg (2005).
18. Priss, U.: Formal Concept Analysis in Information Science. In: Cronin, B. (ed.), *Annual Review of Information Science and Technology*. Vol 40, 2006, p. 521-543 (2006).
19. Vychodil, V.: A new algorithm for computing formal concepts. In: Trappl, R. (ed.), *Cybernetics and Systems 2008*, Proc. 19th EMSCSR, 2008, pp. 15-21, (2008).
20. Wille, R.: Formal Concept Analysis as Mathematical Theory of Concepts and Concept Hierarchies. In: Ganter, B., Stumme, G., Wille, R. (eds.), *Formal Concept Analysis: Foundations and Applications*, pages 1-33. Springer-Verlag, Germany (2005).
21. Zaki, M.J., Hsiao, C-J.: Efficient Algorithms for Mining Closed Itemsets and Their Lattice Structure. In: *IEEE Transactions on Knowledge and Data Mining*, Vol. 17, No. 4, IEE Computer Soc. (April 2005).
22. Zaki, M.J., Ramakrishnan, N.: Reasoning about sets using redescription mining. In: *Proceedings of the Eleventh ACM SIGKDD international Conference on Knowledge Discovery in Data Mining*, Chicago, Illinois, USA, August 21 - 24, 2005. KDD '05. pp. 364-373. ACM, New York, NY (2005).