Algorithms for the Construction of Concept Lattices and Their Diagram Graphs

Sergei O. Kuznetsov[1] and Sergei A. Obiedkov[2]

[1]All-Russia Institute for Scientific and Technical Information (VINITI), Moscow, Russia
serge@viniti.ru
[2]Russian State University for the Humanities, Moscow, Russia
bs-obj@east.ru

Abstract. Several algorithms that generate the set of all formal concepts and graphs of line (Hasse) diagrams of concept lattices are considered. Some modifications of well-known algorithms are proposed. Algorithmic complexity of the algorithms is studied both theoretically (in the worst case) and experimentally. Conditions of preferable use of some algorithms are given in terms of density/sparsity of underlying formal contexts.

1 Introduction

Concept lattices proved to be a useful tool for machine learning and knowledge discovery in databases [3, 6, 9, 19, 22–24]. The problem of generating the set of all concepts and the diagram graph of the concept lattice is extensively studied in the literature [2-5, 7, 10, 11, 13, 16, 18–20]. It is known that the number of concepts can be exponential in the size of the input context (e.g., when the lattice is a Boolean one) and the problem of determining this number is #P-complete [15]. Therefore, from the standpoint of the worst-case complexity, an algorithm can be considered optimal if it generates the concept lattice in time and space linear in the number of all concepts (modulo a factor polynomial in the input size). On the other hand, "dense" contexts, which realize the worst case by bringing about exponential number of concepts, may occur not often in practice. Moreover, various implementation issues, such as dimension of a "typical" context, specificity of the operating system used, and so on, may be crucial for the practical evaluation of algorithms. In this article, we consider, both theoretically and experimentally, several algorithms that generate concept lattices for clearly specified data sets. In most cases, it was possible to improve the original versions of the algorithms. We present modifications of some algorithms and indicate conditions when some of them perform better than the others. Only a few known algorithms generating the concept set construct the graph of the line diagram. We modified some algorithms so that they can construct graphs of line diagrams.

The first comparative study of four algorithms constructing the concept set and the graph of the line diagram can be found in [13]. Descriptions of the algorithms are sometimes buggy and the description of the experimental tests lacks any information about data used for tests. The fact that the choice of an algorithm should depend on input data is not accounted for. Besides, only one of the algorithms considered in [13], namely that of Bordat [2], constructs the graph of the line diagram; thus, it is hard to compare its performance with that of the other algorithms.

A much more elaborate review can be found in [11] (where another algorithm is proposed). The authors of [11] consider algorithms that generate the graph of the line diagram. Algorithms that were not originally designed for this purpose are extended by the authors. Such extensions are not always efficient: for example, the time complexity of the version of the **Ganter** algorithm (called **Ganter-Allaoui**) dramatically increases with the growth of the context size. This drawback can be cancelled by the use of binary search in the list produced by the original **Ganter** algorithm. Tests were conducted only for contexts with small number of attributes per object as compared to the number of all attributes. Our experiments (we consider more algorithms) also show that the algorithm proposed in [11] performs better on such contexts than the others do [17]. However, for "dense" contexts, this algorithm performs worse than some other algorithms (details are found in [17]).

The paper is organized as follows. In Section 2, we give main definitions and an example. In Section 3, we give a survey of batch and incremental algorithms for constructing concept lattices and analyze their worst-case complexity. In Section 4, we consider results of experimental comparison of the algorithms.

2 Main Definitions

First, we introduce standard FCA notation [8], which will be used throughout the paper.

A *(formal) context* is a triple of sets (G, M, I), where G is called a set of objects, M is called a set of attributes, and $I \subseteq G \times M$. For $A \subseteq G$ and $B \subseteq M$: $A' = \{m \in M \mid \forall g{\in}A\ (gIm)\}$; $B' = \{g \in G \mid \forall m{\in}B\ (gIm)\}$. $''$ is a closure operator, i.e., it is monotone, extensive, and idempotent. A *(formal) concept* of a formal context (G, M, I) is a pair (A, B), where $A \subseteq G$, $B \subseteq M$, $A' = B$, and $B' = A$. The set A is called the (*formal*) *extent* and B the (*formal*) *intent* of the concept (A, B). For a context (G, M, I), a concept $X = (A, B)$ is *less general than or equal to* a concept $Y = (C, D)$ (or $X \leq Y$) if $A \subseteq C$ or, equivalently, $D \subseteq B$. Suppose that X and Y are concepts, $X \leq Y$, and there is no concept Z such that $Z \neq X$, $Z \neq Y$, $X \leq Z \leq Y$. Then X is called a *lower neighbor of* Y, and Y is called an *upper neighbor of* X. This relationship is denoted by $X \prec Y$. The set of all concepts of a formal context forms a complete lattice L [8]. The *graph of the line diagram* of a concept lattice (or simply a *diagram graph*) is the directed graph of the relation $\prec$. The *line diagram* is a plane embedding of a diagram graph where each concept vertex is always drawn above all its lower neighbors (thus, the arrows on the arcs become superfluous and can be omitted).

Example 1. Below we present a formal context with some elementary geometric figures and its line diagram. We shall sometimes omit parentheses and write, e.g., 12 instead of $\{1, 2\}$.

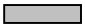
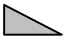
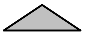
$G \backslash M$	a = 4 vertices	b = 3 vertices	c = has a right angle	d = all sides are equal
1	!		!	!
2 	!		!	
3 		!	!	
4 		!		!

Fig. 1. A formal context

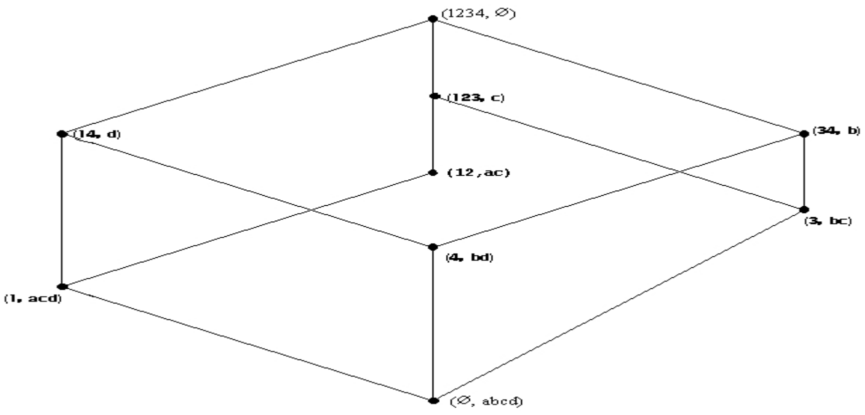


Fig. 2. The line diagram for the context from Fig. 1

Data structures that realize concept sets and diagram graphs of concept lattices are of great importance. Since their sizes can be exponentially large w.r.t. the input size, some their natural representations are not polynomially equivalent, as it is in the case of graphs. For example, the size of the incidence matrix of a diagram graph is quadratic w.r.t. the size of the incidence list of the diagram graph and thus cannot be reduced to the latter in time polynomial w.r.t. the input. Moreover, some important operations, such as find_concept, are performed for some representations (spanning trees [2, 10], ordered lists [7], CbO trees [16], 2-3 trees [1]) in polynomial time, but for some other representations (unordered lists) they can be performed only in exponential time. A representation of a concept lattice can be considered reasonable if its size cannot be exponentially compressed w.r.t. the input and allows the search for a particular concept in time polynomial in the input.

All the algorithms can be divided into two categories: incremental algorithms [3, 5, 11, 20], which, at the *i*th step, produce the concept set or the diagram graph for *i* first objects of the context, and batch ones, which build the concept set or its diagram graph for the whole context from scratch [2, 4, 7, 16, 18, 25]. Besides, any algorithm typically adheres to one of the two strategies: top–down (from the maximal extent to the minimal one) or bottom–up (from the minimal extent to the maximal one).

In many cases, we attempted to improve the efficiency of the original algorithms presented below. Only some of the original versions of the algorithms construct the diagram graph [2, 11, 18, 21]; it turned out that the other algorithms could be extended to construct the diagram graph within the same worst-case time complexity bounds. Some algorithms are given the name of their authors.

In the next section, we will discuss worst-case complexity bounds of the considered algorithms. Due to the possibility of the exponential output of the algorithms, it is reasonable to estimate their complexity not only in terms of the input and output size, but also in terms of (cumulative) delay. Recall that an algorithm for listing a family of combinatorial structures is said to have *polynomial delay* [14] if it executes at most polynomially many computation steps before either outputting each next structure or halting. Note that the worst-case complexity of an algorithm with polynomial delay is a linear function of the output size modulo some factor polynomial in the input size. A weaker notion of efficiency of listing algorithms was proposed in [12]. An algorithm is said to have a *cumulative delay d* if it is the case that at any point of time in any execution of the algorithm with any input *p* the total number of computation steps that have been executed is at most *d*(*p*) plus the product of *d*(*p*) and the number of structures that have been output so far. If *d*(*p*) can be bounded by a polynomial of *p*, the algorithm is said to have a *polynomial cumulative delay*.

3 Algorithms: A Survey

Here we give a brief version of the survey found in [17]. First, we consider batch algorithms. The top-down algorithm **MI-tree** from [25] generates the concept set, but does not build the diagram graph. In **MI-tree**, every new concept is searched for in the set of all concepts generated so far. The top-down algorithm of Bordat [2] uses a tree (a “trie,” cf. [1]) for fast storing and retrieval of concepts. Our version of this algorithm uses a technique that requires O(|*M*|) time to realize whether a concept is generated for the first time without any search. An auxiliary tree, which is actually a spanning tree of the diagram graph, is used to construct the latter. **Ch**((*A*, *B*)) is the set of children of the concept (*A*, *B*) in this tree; it consists of the lower neighbors of (*A*, B) generated for the first time.

```
Bordat
0. $L := \emptyset$
1. Process (($G$, $G'$), $G'$)
2. $L$ is the concept set.

Process (($A$, $B$), $C$)
1. $L := L \cup \{(A, B)\}$
2. $LN$ := LowerNeighbors (($A$, $B$))
3. For each ($D$, $E$) $\in LN$
  3.1. If $C \cap E = B$
```

```
    3.1.1. $C := C \cup E$
    3.1.2. Process($(D, E)$, $C$)
    3.1.3. Ch($(A, B)$) := Ch($(A, B)$) $\cup \{(D, E)\}$
  3.2 Else
    3.2.1. Find($(G, G')$, $(D, E)$)
  3.3. $(A, B)$ is an upper neighbor of $(D, E)$
```

The full version of the algorithm can be found in [17]. The time complexity of the algorithm is $O(|G||M|^2|L|)$; its polynomial delay is $O(|G||M|^2)$.

The well-known algorithm proposed by Ganter computes closures for only some of subsets of G and uses an efficient canonicity test, which does not address the list of generated concepts. The subsets are considered in lexicographic order [7, 8]. The **Ganter** algorithm has polynomial delay $O(|G|^2|M|)$. Its complexity is $O(|G|^2|M||L|)$.

The **Close by One** (**CbO**) algorithm uses a notion of canonicity similar to that of **Ganter** and a similar method for selecting subsets. It employs an intermediate structure that helps to compute closures more efficiently using the generated concepts. Objects are assigned numbers; g ✐ h holds whenever the number of g is smaller than that of h. The **CbO** algorithm obtains a new concept from a concept (A, B) generated at a previous step by intersecting B with the intent of an object g that does not belong to A. The generation is considered canonical if the intersection is not contained in any object from $G \setminus A$ with smaller number than that of g. The algorithm repeatedly calls *Process*($\{g\}$, g, ($\{g\}''$, $\{g\}'$)) for each object g.

```
Process($A$, $g$, $(C, D)$)                          $C = A''$, $D = A'$
1. If $\{h \mid h \in C \setminus A$ & $h$ ✐ $g\} = \varnothing$
  1.1. $L := L \cup \{(C, D)\}$
  1.2. For each $f \in \{h \mid h \in G \setminus C$ & $g$ ✐ $h\}$
    1.2.1. $Z := C \cup \{f\}$
    1.2.2.$Y := D \cap \{f\}'$
    1.2.3. $X := Y'$ $(= Z \cup \{h \mid h \in G \setminus Z$ & $Y \subseteq \{h\}'\})$
    1.2.4. Process($Z$, $f$, $(X, Y)$)
```

The CbO algorithm has polynomial delay $O(|G|^3|M|)$ and complexity $O(|G|^2|M||L|)$. To construct the diagram graph with the **CbO** algorithm, we use a tree, which is not a spanning tree of the diagram graph, but it agrees with the concept order.

The idea of the bottom-up algorithm in [18] is to generate the bottom concept and then, for each concept that is generated for the first time, generate all its upper neighbors. Lindig uses a tree of concepts that allows one to check whether some concept was generated earlier. The description of the tree is not detailed in [18], but it seems to be the spanning tree of the inverted diagram graph (i.e., with the root at the bottom of the diagram), similar to the tree from **Bordat**. Finding a concept in such a tree takes $O(|G||M|)$ time. In fact, this algorithm may be regarded as a bottom-up version of the **Bordat** algorithm. The time complexity of the algorithm is $O(|G|^2|M||L|)$. Its polynomial delay is $O(|G|^2|M|)$.

The **AI-tree** [25] and **Chein** [4] algorithms operate with extent–intent pairs and generate each new concept intent as the intersection of intents of two generated concepts. At every iteration step of the **Chein** algorithm, a new layer of concepts is created by intersecting pairs of concept intents from the current layer and the new intent is searched for in the new layer. We introduced several modifications [17] that made it possible to improve the performance of the algorithm. The time complexity of the modified algorithm is $O(|G|^3|M||L|)$; its polynomial delay is $O(|G|^3|M|)$.

Now we consider incremental algorithms, which cannot have polynomial delay. Nevertheless, all algorithms below have cumulative polynomial delay.

L. Nourine [21] proposes an algorithm for the construction of the lattice using a lexicographic tree with the best known worst-case complexity bound $O((|G| + |M|)|G||L|)$. Edges of the tree are labeled with attributes, and nodes are labeled with concepts whose intents consist of the attributes that label the edges leading from the root to the node. Clearly, some nodes do not have labels. First, the tree is constructed incrementally (similar to the **Norris** algorithm presented below). An intent of a new concept C is created by intersecting an object intent g' and the intent of a concept D created earlier, and the extent of C is formed by adding g to the extent of D; this takes $O(|M| + |G|)$ time. A new concept is searched for in the tree using the intent of the concept as the key; this search requires $O(|M|)$ time. When the tree is created, it is used to construct the diagram graph. For each concept C, its parents are sought for as follows. Counters are kept for every concept initialized to zero at the beginning of the process. For each object, the intersection of its intent and the concept intent is produced in $O(|M|)$ time. A concept D with the intent equal to this intersection is found in the tree in $O(|M|)$ time and the value in the counter increases; if the counter is equal to the difference between the cardinalities of the concepts C and D (i.e., the intersection of the intent of C and the intent of any object from D outside C is equal to the intent of D), the concept D is a parent of C.

The algorithm proposed by E. Norris [20] can be considered as an incremental analogue of the **CbO** algorithm. The concept tree (which is useful only for diagram construction) can be built as follows: first, there is only the dummy root; examine objects from G and for each concept of the tree check whether the object under consideration has all the attributes of the concept intent; if it does, add it to the extent; otherwise, form a new node and declare it a child node of the current one; the extent of the corresponding concept equals the extent of the parent node plus the object being examined; the intent is the intersection of this object intent and the parent intent; next, test the new node for the canonicity; if the test fails, remove it from the tree. The original version of the **Norris** algorithm from [20] does not construct the diagram graph. In this case, **Norris** is preferable to **CbO**, as the latter has to remember how the last concept was generated; this involves additional storage resources, as well as time expenses. The **Norris** algorithm does not maintain any auxiliary structure. Besides, the closure of an object set is never computed explicitly.

The algorithm proposed by Godin [11] has the worst-case time complexity quadratic in the number of concepts. This algorithm is based on the use of an efficiently computable hash-function f (which is actually the cardinality of an intent) defined on the set of concepts.

C. Dowling proposed an incremental algorithm for computing knowledge spaces [5]. A dual formulation of the algorithm allows generation of formal concepts. The worst-case complexity of the algorithm is $O(|M||G|^2|L|)$, the constants in this upper bound are large and in practice, the algorithm performs worse than other algorithms.

4 Results of Experimental Tests

The algorithms were implemented in C++. The tests were run on a Pentium II–300 computer, 256 MB RAM. Here, we present a number of charts that show how the execution time of the algorithms depends on various parameters. More charts can be found in [17].

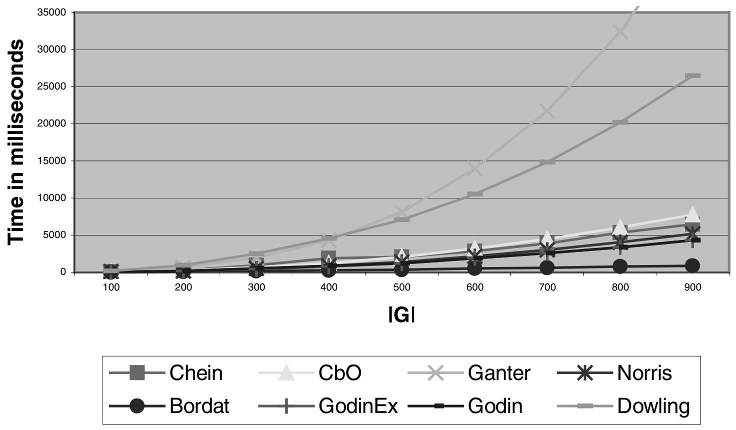


Fig. 3. Concept set: |*M*| = 100; |*g*′| = 4

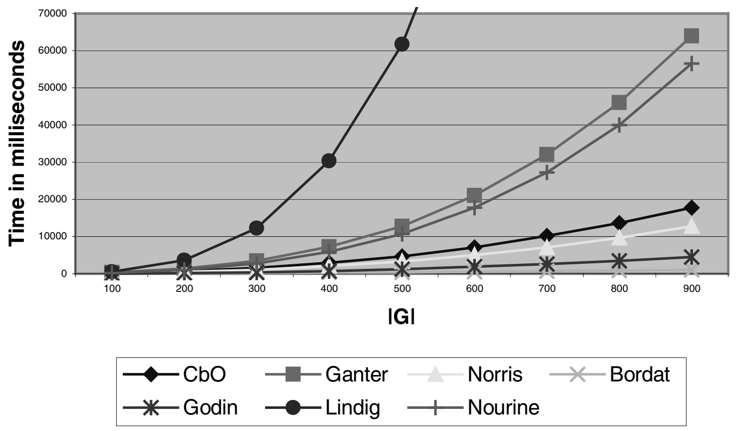


Fig. 4. Diagram graph: |*M*| = 100; |*g*′| = 4.

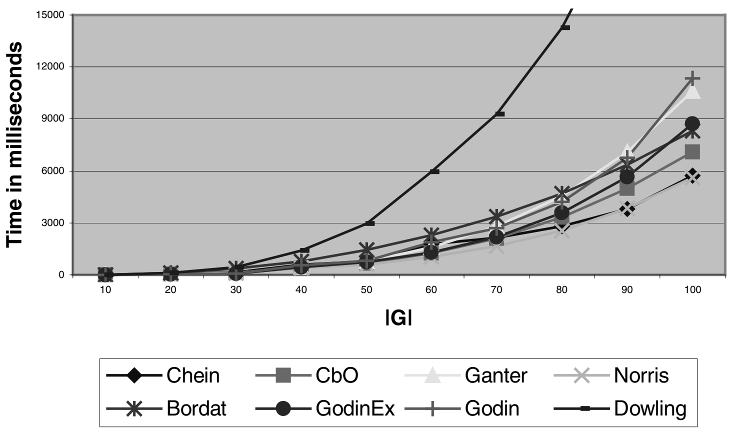


Fig. 5. Concept set: $|M| = 100$; $|g'| = 25$

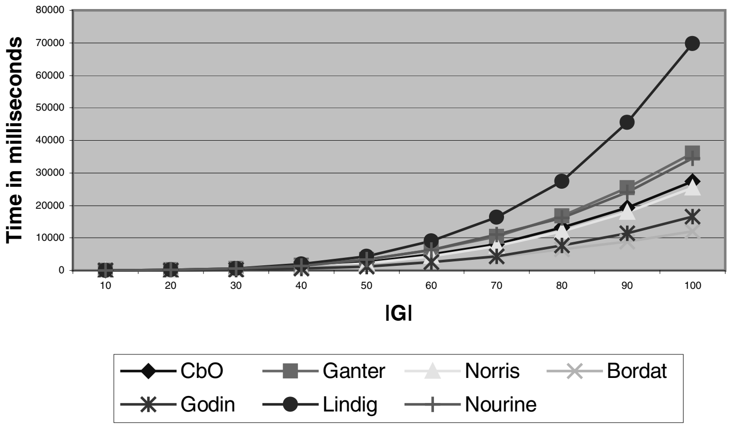


Fig. 6. Diagram graph: $|M| = 100$; $|g'| = 25$

The **Godin** algorithm (and **GodinEx**, which is the version of the **Godin** algorithm using the cardinality of extents for the hash-function) is a good choice in the case of a sparse context. However, when contexts become denser, its performance decreases dramatically. The **Bordat** algorithm seems most suitable for large contexts, especially if it is necessary to build the diagram graph. When $|G|$ is small, the **Bordat** algorithm runs several times slower than other algorithms, but, as $|G|$ grows, the difference between **Bordat** and other algorithms becomes smaller, and, in many cases, **Bordat** finally turns out to be the leader. For large and dense contexts, the fastest algorithms are bottom-up canonicity-based algorithms (**Norris, Ganter, CbO**).

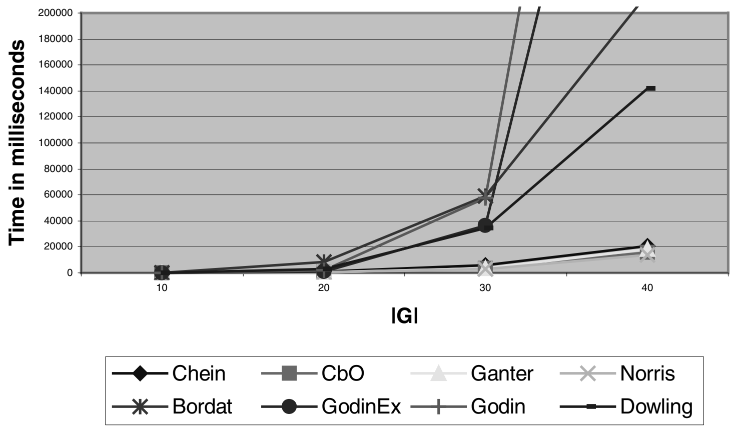


Fig. 7. Concept set: $|M| = 100$; $|g'| = 50$

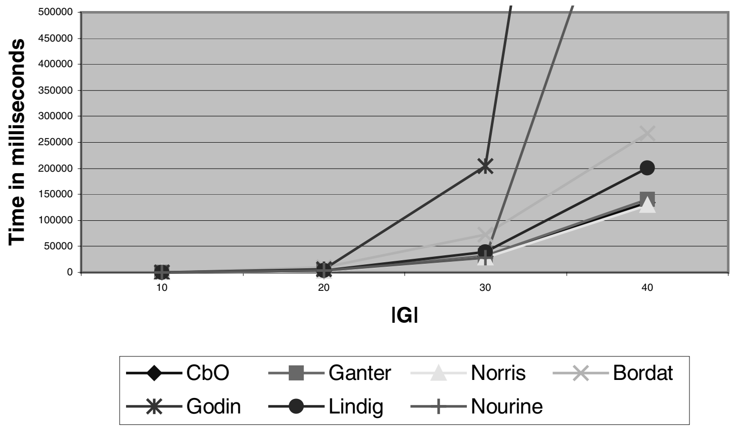


Fig. 8. Diagram graph: $|M| = 100$; $|g'| = 50$

It should be noted that the **Nourine** algorithm with the best worst-case time complexity did not show the best performance in our experiments: even when contexts of the form (G, G, $\neq$) were processed (which corresponds to the worst case of Boolean concept lattice), it was inferior to the **Norris** algorithm. Probably, this can be accounted to the fact that we represent attribute sets by bit strings, which allows very efficient implementation of set-theoretical operations (32 attributes per one processor cycle); whereas searching in the Nourine-style lexicographic tree, one still should individually consider each attribute labeling edges.

Figures 9–10 show the execution time for the contexts of the form (G, G, ≠) with $2^{|G|}$ concepts.

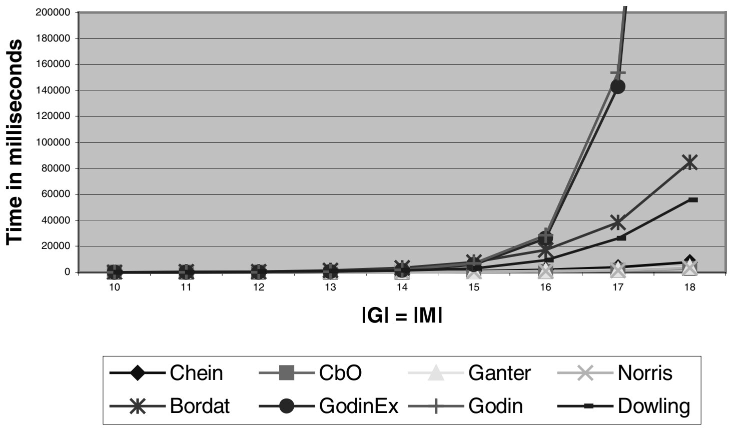


Fig. 9. Concept set: contexts of the form (G, G, ≠)

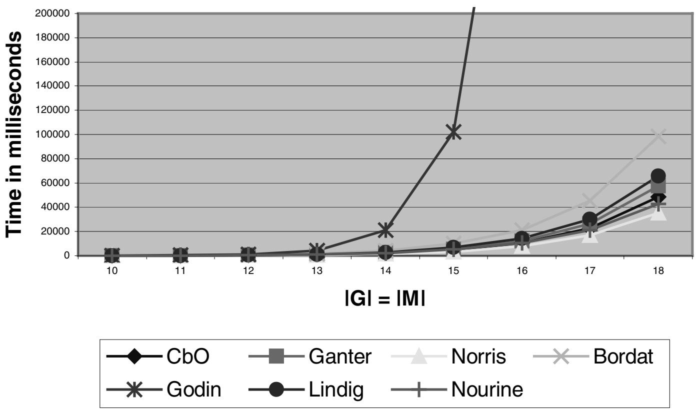


Fig. 10. Diagram graph: contexts of the form (G, G, ≠)

5 Conclusion

In this work, we attempted to compare some well-known algorithms for constructing concept lattices and their diagram graphs. A new algorithm was proposed in [22] quite recently, so we could not include it in our experiments. Its worst-time complexity is not better than that of the algorithms described above, but the authors report on its good practical performance for databases with very large number of objects. Comparing the performance of this algorithm with those considered above and testing the algorithms on large databases, including "classical" ones, will be the subject of the further work. We can also mention works [3, 19] where algorithms were applied for learning and data analysis, e.g., in [19] a **Bordat**-type algorithm was used. The description of the algorithm in [3] does not give details about the test for uniqueness of a generated concept, i.e., whether it is already in the list. As we have seen, this test is crucial for the efficiency of an algorithm.

The choice of the algorithm for construction of the concept set and diagram graph should be based on the properties of the input data. The general rule is as follows: the **Godin** algorithm should be used for small and sparse contexts; for dense contexts, the algorithms based on canonicity tests, linear in the number of output concepts, such as **Close by One**, **Norris**, and **Ganter,** should be used. The **Bordat** performs well on contexts of average density, especially, when the diagram graph is to be constructed.

As mentioned above, the experimental comparison of execution times of algorithms is implementation-dependent. To reduce this dependence, we implemented a program that made it possible to compare algorithms not only in the execution time, but also in the number of operations performed. Such comparison is both more reliable and more helpful, as it allows choosing an algorithm based on the computational complexity of the operations in particular implementation.

References

1. Aho, A.V., Hopcroft, J.E., Ullmann, J.D., Data Structures and Algorithms, Reading, Addison–Wesley (1983).
2. Bordat, J.P., Calcul pratique du treillis de Galois d'une correspondance, Math. Sci. Hum., no. 96, (1986) 31–47.
3. Carpineto, C., Giovanni, R., A Lattice Conceptual Clustering System and Its Application to Browsing Retrieval, Machine Learning, no. 24 (1996) 95–122.
4. Chein, M., Algorithme de recherche des sous-matrices premières d'une matrice, Bull. Math. Soc. Sci. Math. R.S. Roumanie, no. 13 (1969) 21–25.
5. Dowling, C.E., On the Irredundant Generation of Knowledge Spaces, J. Math. Psych., vol. 37, no. 1 (1993) 49-62.
6. Finn, V.K., Plausible Reasoning in Systems of JSM Type, Itogi Nauki i Tekhniki, Ser. Informatika, vol. 15 (1991) 54–101.
7. Ganter, B., Two Basic Algorithms in Concept Analysis, FB4-Preprint No. 831, TH Darmstadt (1984).
8. Ganter, B., Wille, R., Formal Concept Analysis. Mathematical Foundations, Springer, (1999).

9. Ganter B., Kuznetsov, S., Formalizing Hypotheses with Concepts, Proc. of the 8th International Conference on Conceptual Structures, ICCS 2000, Lecture Notes in Artificial Intelligence, vol. 1867 (2000) 342-356.
10. Ganter, B., Reuter, K., Finding All Closed Sets: A General Approach, Order, vol. 8, (1991) 283-290.
11. Godin, R., Missaoui, R., Alaoui, H., Incremental Concept Formation Algorithms Based on Galois Lattices, Computation Intelligence (1995).
12. Goldberg, L.A., Efficient Algorithms for Listing Combinatorial Structures, Cambridge University Press (1993).
13. Guénoche, A., Construction du treillis de Galois d'une relation binaire, Math. Inf. Sci. Hum., no. 109 (1990) 41–53.
14. Johnson, D.S., Yannakakis, M., Papadimitriou, C.H., On Generating all Maximal Independent Sets, Inf. Proc. Let., vol. 27 (1988) 119-123.
15. Kuznetsov, S.O., Interpretation on Graphs and Complexity Characteristics of a Search for Specific Patterns, Automatic Documentation and Mathematical Linguistics, vol. 24, no. 1 (1989) 37-45.
16. Kuznetsov, S.O., A Fast Algorithm for Computing All Intersections of Objects in a Finite Semi-lattice, Automatic Documentation and Mathematical Linguistics, vol. 27, no. 5 (1993) 11–21.
17. Kuznetsov, S.O., Obiedkov S.A., Algorithms for the Construction of the Set of All Concepts and Their Line Diagram, Preprint MATH-Al-05, TU-Dresden, June 2000.
18. Lindig, C., Algorithmen zur Begriffsanalyse und ihre Anwendung bei Softwarebibliotheken, (Dr.-Ing.) Dissertation, Techn. Univ. Braunschweig (1999).
19. Mephu Nguifo, E., Njiwoua, P., Using Lattice-Based Framework As a Tool for Feature Extraction, in Feature Extraction, Construction and Selection: A Data Mining Perspective, H. Liu and H. Motoda (eds.), Kluwer (1998).
20. Norris, E.M., An Algorithm for Computing the Maximal Rectangles in a Binary Relation, Revue Roumaine de Mathématiques Pures et Appliquées, no. 23(2) (1978) 243–250.
21. Nourine L., Raynaud O., A Fast Algorithm for Building Lattices, Information Processing Letters, vol. 71 (1999) 199-204.
22. Stumme G., Taouil R., Bastide Y., Pasquier N., Lakhal L., Fast Computation of Concept Lattices Using Data Mining Techniques, in Proc. 7th Int. Workshop on Knowledge Representation Meets Databases (KRDB 2000) 129-139.
23. Stumme G., Wille R., and Wille U., Conceptual Knowledge Discovery in Databases Using Formal Concept Analysis Methods, in Proc. 2nd European Symposium on Principles of Data Mining and Knowledge Discovery (PKDD'98).
24. Waiyamai K., Lakhal L., Knowledge Discovery from Very Large Databases Using Frequent Concept Lattices, in Proc. 11th European Conference on Machine Learning (ECML'2000), 437-445.
25. Zabezhailo, M.I., Ivashko, V.G., Kuznetsov, S.O., Mikheenkova, M.A., Khazanovskii, K.P., and Anshakov, O.M., Algorithms and Programs of the JSM-Method of Automatic Hypothesis Generation, Automatic Documentation and Mathematical Linguistics, vol. 21, no. 5 (1987) 1–14.