Comparing Performance of Algorithms for Generating Concept Lattices

Sergei O. Kuznetsov[1] and Sergei A. Ob''edkov[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 diagram graphs 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/sparseness of underlying formal contexts. Principles of comparing practical performance of algorithms are discussed.

1 Introduction

Concept (Galois) lattices proved to be a useful tool in many applied domains: machine learning, data mining and knowledge discovery, information retrieval, etc. [3, 6, 9, 22, 23]. The problem of generating the set of all concepts and the concept lattice of a formal context is extensively studied in the literature [2-5, 7, 11, 13, 16, 18–22]. 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 generating all concepts and/or a concept lattice can be considered optimal if it generates the lattice with polynomial time delay and space linear in the number of all concepts (modulo some factor polynomial in the input size). First, we give some standard definitions of Formal Concept Analysis (FCA) [8].

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 $B \subseteq M$: $A' = \{m \in M \mid \forall g \in A\ (gIm)\}$, $B' = \{g \in G \mid \forall m \in B\ (gIm)\}$. 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 *extent*, and the set B is called the *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, $D \subseteq B$. For two concepts X and Y such that $X \leq Y$ and there is no concept Z with $Z \neq X$, $Z \neq Y$, $X \leq Z \leq Y$, the concept X is called a *lower neighbor of* Y, and Y is called an *upper neighbor of* X. This relationship is denoted by $X \prec Y$. We call the (directed) graph of this relation a *diagram graph*. A plane (not necessarily a pla-

nar) embedding of a diagram graph where a concept has larger vertical coordinate than that of any of its lower neighbors is called a *line* (Hasse) *diagram*. The problem of drawing line diagrams [8] is not discussed here.

The problem of comparing performance of algorithms for constructing concept lattices and their diagram graphs is a challenging and multifaceted one. The first comparative study of several algorithms constructing the concept set and diagram graphs can be found in [13]. However, the formulation of the algorithms is not always correct, and the description of the results of the experimental tests lacks any information about data used for tests. The fact that the choice of the algorithm should be dependent on the input data is not accounted for. Besides, only one of the algorithms considered in [13], namely that of Bordat [2], constructs the diagram graph; thus, it is hard to compare its time complexity with that of the other algorithms.

A later review with more algorithms and more information on experimental data can be found in [11]. Only algorithms generating diagram graphs are considered. The algorithms that were not originally designed for this purpose are extended by the authors to generate diagram graphs. Unfortunately, such extensions are not always effective: 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 efficient 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 some other algorithms, e.g., that of Nourine [21]) also show that the algorithm proposed in [11] works faster on such contexts than the others do. However, in other situations not covered in [11] this algorithm is far behind some other algorithms.

The rest of the paper is organized as follows. In Section 2, we discuss the principles of comparing efficiency of algorithms and make an attempt at their classification. In Section 3, we give a short review of the algorithms and analyze their worst-case complexity. In Section 4, we present the results of experimental comparison.

2 On Principles of Comparison

In our study, we considered both theoretical (worst-case) and experimental complexity of algorithms. As for the worst-case upper bounds, the algorithms with complexity linear in the number of concepts (modulo a factor polynomial of the input size) are better than those with complexity quadratic in the number of concepts; and the former group can be subdivided into smaller groups according to the form of the factor polynomial of input. According to this criterion, the present champion is the algorithm by Nourine [21]. On the other hand, "dense" contexts, which realize the worst case by bringing about exponential number of concepts, may occur not often in practice.

Starting a comparison of algorithms "in practice", we face a bunch of problems. First, algorithms, as described by their authors, often allow for different interpretation of crucial details, such as the test of uniqueness of a generated concept. Second, authors seldom describe exactly data structures and their realizations. Third, algorithms

behave differently on different databases (contexts). Sometimes authors compare their algorithms with other on specific data sets. We would like to propose the community to reach a consensus w.r.t. databases to be used as testbeds. Our idea is to consider two types of testbeds. On the one hand, some "classical" (well-recognized in data analysis community) databases should be used, with clearly defined scalings if they are many-valued. On the other hand, we propose to use "randomly generated contexts". The main parameters of a context $K = (G, M, I)$ seem here to be the (relative to $|M|$) number of objects $|G|$ and the (relative to $|G|$) number of attributes, the (relative, i.e. compared to $|G||M|$) size of the relation I, average number of attributes per object intent (resp., average number of objects per attribute extent). The community should specify particular type(s) of random context generator(s) that can be tuned by the choice of above (or some other) parameters.

Another major difficulty resides in the choice of a programming language and platform, which strongly affects the performance. A possible way of avoiding this difficulty could be comparing not the time but number of specified operations (intersections, unions, closures, etc.) from a certain library, but here one encounters the difficulty of weighting these operations in order to get the overall performance. Much simpler would be comparing algorithms using a single platform.

In this article, we compare performance of several algorithms for clearly specified random data sets (contexts). As for ambiguities in original pseudo-code formulations of the algorithms, we tried to find most efficient realizations for them. Of course, this does not guarantee that a more efficient realization cannot be found.

In most cases, it was possible to improve the original versions of the algorithms. Since only few known algorithms generating the concept set construct also the diagram graph, we attempted to modify some algorithms making them able to construct diagram graphs. The versions of algorithms used for comparison are presented in [17].

As mentioned above, data structures that realize concept sets and diagram graphs of concept lattices are of great importance. Since their sizes can be exponential 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 finding a concept, are performed for some representations (spanning trees [2, 10], ordered lists [7], CbO trees [16], 2-3 trees, see [1] for the definition) 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.

Table 1 presents an attempt at classification of algorithms. Note that this classification refers to our versions of the algorithms described in [17] rather than to original versions (except for **Titanic** [22], which we have not implemented; it is included into classification, because it realizes an approach completely different from that of the other algorithms). Here, we do not address techniques for building diagram graphs; the attributes of the context in Table 1 describe only construction of the concept set.

All the algorithms can be divided into two categories: incremental algorithms [3, 5, 11, 20], which, at the ith step, produce the concept set or the diagram graph for i first

objects of the context, and batch ones, which build the concept set and its diagram graph for the whole context from scratch [2, 4, 7, 16, 18, 24]. Besides, any batch 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). However, it is always possible to reverse the strategy of the algorithm by considering attributes instead of objects and vice versa; therefore, we choose not to include this property into the classification.

Generation of the concept set presents two main problems: (1) how to generate all concepts; (2) how to avoid repetitive generation of the same concept or, at least, to determine whether a concept is generated for the first time. There are several ways to generate a new intent. Some algorithms (in particular, incremental ones) intersect a generated intent with some object intent. Other algorithms compute an intent explicitly intersecting all objects of the corresponding extent. There are algorithms that, starting from object intents, create new intents by intersecting already obtained intents. Lastly, the algorithm from [22] does not use the intersection operation to generate intents. It forms new intents by adding attributes to those already generated and tests some condition on supports of attribute sets (a support of an attribute set is the number of objects whose intents contain all attributes from this set) to realize whether an attribute set is an intent.

In Table 1, attributes m2–m6 correspond to techniques used to avoid repetitive generation of the same concept. This can be done by maintaining specific data structures. For example, the **Nourine** algorithm constructs a tree of concepts and searches in this tree for every newly generated concept. Note that other algorithms (e.g., **Bordat** and **Close by One**) also may use trees for storing concepts, which allows efficient search for a concept when the diagram graph is to be constructed. However, these algorithms use other techniques for identifying the first generation of a concept, and, therefore, they do not have the m5 attribute in the context from Table 1.

Some algorithms divide the set of all concepts into disjoint sets, which allows narrowing down the search space. For example, the **Chein** algorithm stores concepts in layers, each layer corresponding to some step of the algorithm. The original version of this algorithm looks through the current layer each time a new concept is generated. The version we used for comparison does not involve search to detect duplicate concepts; instead, it employs a canonicity test based on the lexicographical order (similar to that of **Ganter**), which made it possible to greatly improve the efficiency of the algorithm. We use layers only for generation of concepts: a new intent is produced as the intersection of two intents from the same layer. (In our version of the algorithm, layers are much smaller than those in [4]; see [17] for details.) The **Godin** algorithm uses a hash function (the cardinality of intents), which makes it possible to distribute concepts among "buckets" and to reduce the search. Several algorithms (**Ganter**, **Close by One**) generate concepts in the lexicographical order of their extents assuming that there is a linear order on the set of objects. At each step of the algorithm, there is a *current object*. The generation of a concept is considered canonical if its extent contains no object preceding the *current object*. Our implementation of the **Bordat** algorithm uses an attribute cache: the uniqueness of a concept is tested by intersecting its intent with the content of the cache (for more details, see [17]).

Table 1. Properties of algorithms constructing concept lattices: m1—incremental; m2—uses canonicity based on the lexical order; m3—divides the set of concepts into several parts; m4—uses hash function; m5—maintains an auxiliary tree structure; m6—uses attribute cache; m7—computes intents by subsequently computing intersections of object intents (i.e., $\{g\}' \cap \{h\}'$); m8—computes intersections of already generated intents; m9—computes intersections of non-object intents and object intents; m10—uses supports of attribute sets.

	m1	**m2**	**m3**	**m4**	**m5**	**m6**	**m7**	**m8**	**m9**	**m10**
Bordat						X	X			
Ganter		X					X			
Close by One		X							X	
Lindig					X				X	
Chein		X	X					X		
Nourine	X				X				X	
Norris	X	X							X	
Godin	X		X	X					X	
Dowling	X	X							X	
Titanic										X

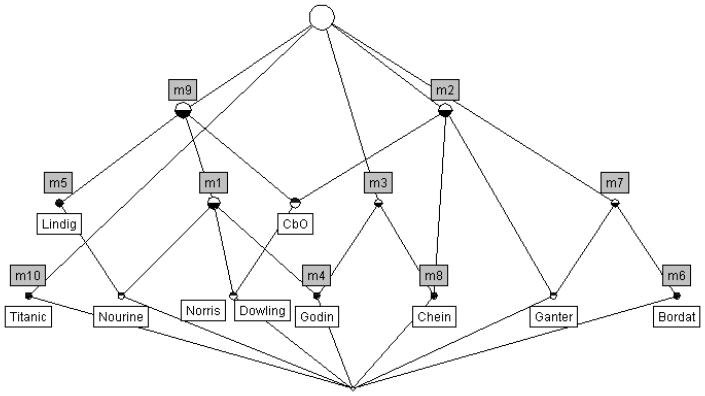

Fig. 1. The line diagram of algorithms

In many cases, we attempted to improve the efficiency of the original algorithms. 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.

In the next section, we will discuss worst-case complexity bounds of the considered algorithms. Since the output size can be exponential in the input, it is reasonable to estimate complexity of the algorithms not only in terms of input and output sizes, 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. An algorithm is said to have a *cumulative delay d* [12] 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 Short Survey

Some top-down algorithms have been proposed in [2] and [24]. The algorithm **MI-tree** from [24] 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 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. The time complexity of **Bordat** is $O(|G||M|^2|L|)$, where $|L|$ is the size of the concept lattice. Moreover, this algorithm has a polynomial delay $O(|G||M|^2)$.

The 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. It produces the set of all concepts in time $O(|G|^2|M||L|)$ and has polynomial delay $O(|G|^2|M|)$.

The **Close by One** (**CbO**) algorithm uses a similar notion of canonicity, a similar method for selecting subsets, and an intermediate structure that helps to compute closures more efficiently using the generated concepts. Its time complexity is $O(|G|^2|M||L|)$, and its polynomial delay is $O(|G|^3|M|)$.

The idea of a 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 time complexity of the algorithm is $O(|G|^2|M||L|)$. Its polynomial delay is $O(|G|^2|M|)$.

The **Chein** [4] algorithm represents the objects by extent–intent pairs and generates each new concept intent as the intersection of intents of two existent 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 that made it possible to greatly improve the performance of the algorithm. The time complexity of the modified algorithm is $O(|G|^3|M||L|)$. The algorithm has polynomial delay $O(|G|^3|M|)$.

Due to their incremental nature, the algorithms considered below do not have polynomial delay. Nevertheless, they all have cumulative polynomial delay.

Nourine proposes an $O((|G| + |M|)|G||L|)$ algorithm for the construction of the lattice using a lexicographic tree [21] with edges labeled by attributes and nodes labeled by concepts. Note that this algorithm is only half-incremental. First, this algorithm incrementally constructs the concept set outputting a tree of concepts; next, it uses this tree to construct the diagram graph.

The algorithm proposed by Norris [20] is essentially an incremental version of the **CbO** algorithm. The original version of the **Norris** algorithm from [20] does not construct the diagram graph. The time complexity of the algorithm is $O(|G|^2|M||L|)$.

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.

Dowling proposed [5] an incremental algorithm for computing knowledge spaces. A dual formulation of the algorithm allows generation of the concept set. Despite the fact that the theoretical worst-case complexity of the algorithm is $O(|M||G|^2|L|)$, the constant in this upper bound seems to be too large and in practice the algorithm performs worse than other algorithms.

4 Results of Experimental Tests

The algorithms were implemented in C++ in the *Microsoft Visual C++* environment. 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].

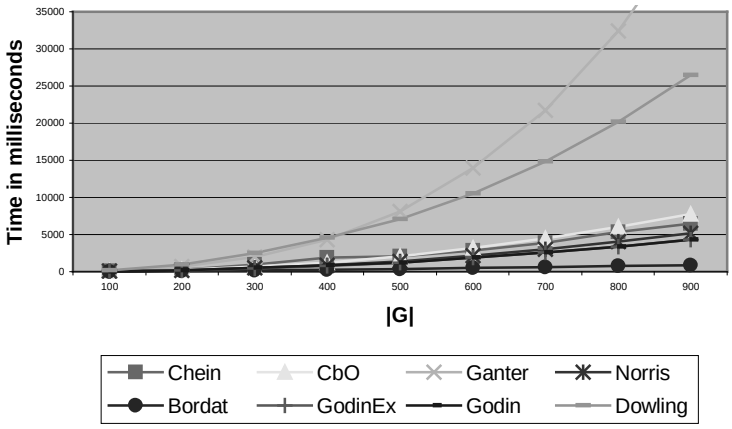

Fig. 2. Concept set: $|M| = 100$; $|g'| = 4$.

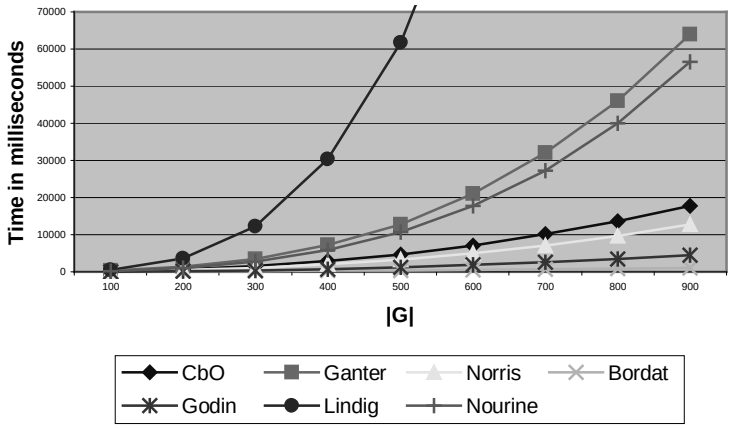

Fig. 3. Diagram graph: $|M| = 100$; $|g'| = 4$.

For tests, we used randomly generated data. Contexts were generated based on three parameters: $|G|$, $|M|$, and the number of attributes per object (denoted below as $|g'|$; all objects of the same context had equal numbers of attributes). Given $|g'|$, every row of the context (i.e., every object intent) was generated by successively calling the *rand* function from the standard C library to obtain the numbers of attributes constituting the object intent, which lead to uniform distribution.

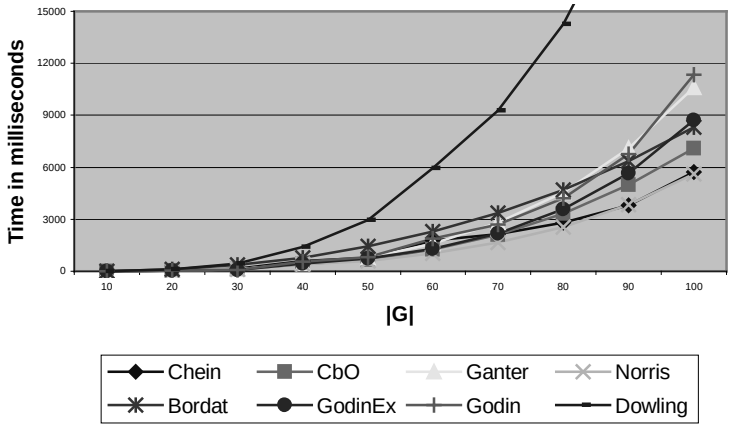

Fig. 4. Concept set: $|M| = 100$; $|g'| = 25$.

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, CbO**).

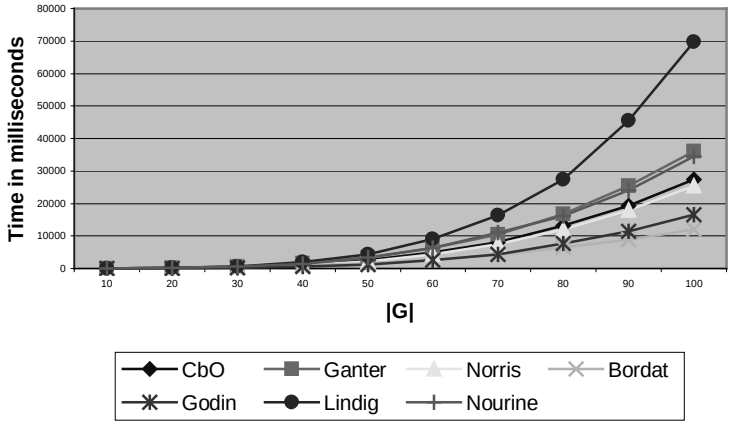

Fig. 5. Diagram graph: |*M*| = 100; |*g'*| = 25.

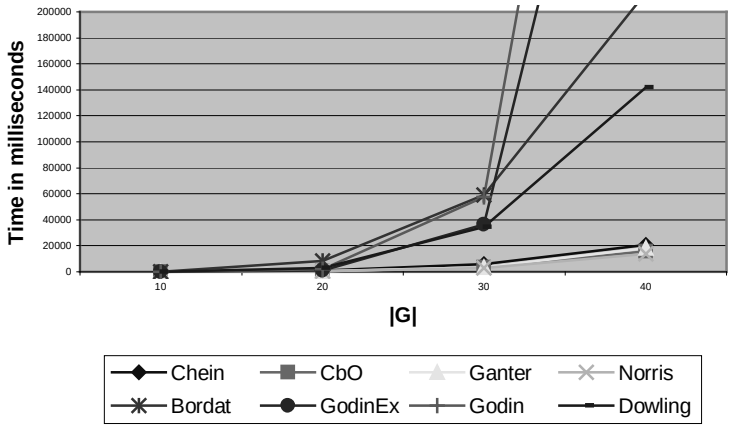

Fig. 6. Concept set: |*M*| = 100; |*g'*| = 50.

It should be noted that the **Nourine** algorithm featuring the smallest time complexity, has not been the fastest algorithm: even when diagonal contexts of the form (*G*, *G*,≠) (which corresponds to the worst case) are processed, its performance 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.

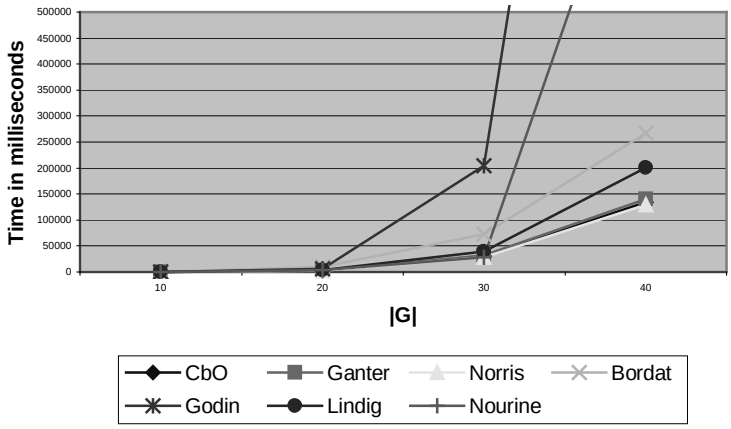

Fig. 7. Diagram graph: $|M| = 100$; $|g'| = 50$.

Figures 8–9 show the execution time for the contexts of the form $(G, G, \neq)$, which yield $2^{|G|}$ concepts.

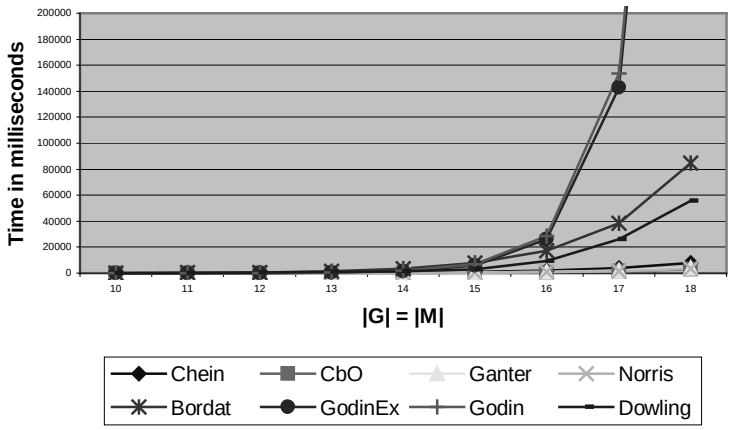

Fig. 8. Concept set: diagonal contexts.

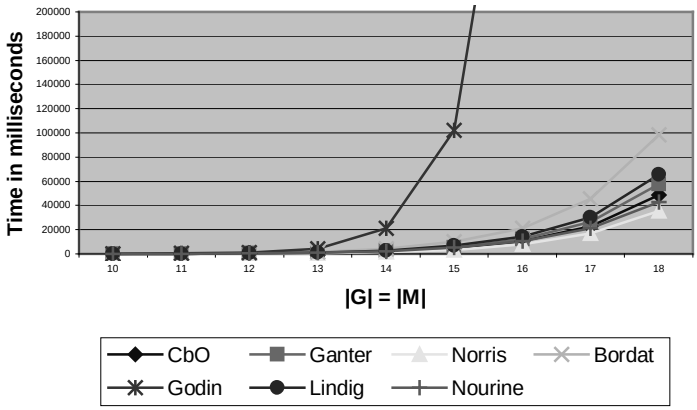

Fig. 9. Diagram graph: diagonal contexts.

5 Conclusion

In this work, we attempted to compare, both theoretically and experimentally, some well-known algorithms for constructing concept lattices. We discussed principles of experimental comparison.

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 similar algorithms were applied for machine learning and data analysis, e.g., in [19] a Bordat-type algorithm was used.

The choice of an algorithm for construction of the concept lattice should be based on the properties of input data. Recommendations based on our experiments are as follows: the **Godin** algorithm should be used for small and sparse contexts; for dense contexts, the algorithms based on the canonicity test, linear in the number of input objects, such as **Close by One** and **Norris**, should be used. **Bordat** performs well on contexts of average density, especially, when the diagram graph is to be constructed. Of course, these recommendations should not be considered as the final judgement. By this work, we would like rather to provoke further interest in well-substantiated comparison of algorithms that generate concept lattices.

References

1. Aho, A.V., Hopcroft, J.E., and 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., 1986, no. 96, pp. 31–47.
3. Carpineto, C., Romano, G., A Lattice Conceptual Clustering System and Its Application to Browsing Retrieval, Machine Learning, 1996, no. 24, pp. 95–122.
4. Chein, M., Algorithme de recherche des sous-matrices premières d'une matrice, Bull. Math. Soc. Sci. Math. R.S. Roumanie, 1969, no. 13, pp. 21–25.
5. Dowling, C.E., On the Irredundant Generation of Knowledge Spaces, J. Math. Psych., 1993, vol. 37, no. 1, pp. 49-62.
6. Finn, V.K., Plausible Reasoning in Systems of JSM Type, Itogi Nauki i Tekhniki, Ser. Informatika, 1991, vol. 15, pp. 54–101.
7. Ganter, B., Two Basic Algorithms in Concept Analysis, FB4-Preprint No. 831, TH Darmstadt, 1984.
8. Ganter, B. and Wille, R., Formal Concept Analysis. Mathematical Foundations, Springer, 1999.
9. Ganter B. and Kuznetsov, S., Formalizing Hypotheses with Concepts, Proc. of the 8th International Conference on Conceptual Structures, ICCS 2000, Lecture Notes in Artificial Intelligence, vol. 1867.
10. Ganter, B. and Reuter, K., Finding All Closed Sets: A General Approach, Order, 1991, vol. 8, pp. 283-290.
11. Godin, R., Missaoui, R., and 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., 1990, no. 109, pp. 41–53.
14. Johnson, D.S., Yannakakis, M., and Papadimitriou, C.H., On Generating all Maximal Independent Sets, Inf. Proc. Let., 1988, vol. 27, 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. and Ob''edkov S.A., Algorithm 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. and 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, 1978, no. 23(2), pp. 243–250.
21. Nourine L. and 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. 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.