

Chapter 5: AddAtom algorithmic performance

The AddAtom algorithmic performance was studied both theoretically (worst case behaviour) and empirically. This chapter starts with a short survey of published concept lattice construction algorithms (section 5.1) before deriving a theoretical upper bound to the complexity of AddAtom (section 5.2). A theoretical worst-case performance bound of AddAtom is $O(\|L\| \cdot \|O\|^2 \cdot \max(\|O'\|))$. This performance upper bound is however a higher of magnitude as the current best performer namely that of Nourine and Raynaud (1999, 2002) with an upper bound of $O((\|O\| + \|A\|) \cdot \|O\| \cdot \|L\|)$. Despite being cubic in nature relative to the lattice size, it is argued that this bound of AddAtom is not a very sharp upper bound and that the terms in the complexity expression are in practice, much more of an overestimate than the $\|O\|$ and $\|A\|$ terms that appear in the upper bound estimates of other construction algorithms. The performance is thus best confirmed via experimental comparisons.

For the purposes of experimental comparison, a two-step approach was taken. Firstly, a pilot study comparing the original AddAtom implementation in C++ (described in chapter 7) to implementations of two published construction algorithms, namely that of Godin (1991) and Carpineto and Romano (1993, 1996b). The pilot comparison showed that there is *prima facie* evidence that the algorithm performs very well and that wider study is justified. The second study, involving a wider set of experimental comparisons across a larger number of lattice construction algorithms, was conducted in collaboration with another researcher. For the sake of reference, the first, smaller experimental comparison will be referred to as the “pilot study” (section 5.3) and the second as the “wide performance study” (section 5.4).

The results of both experimental comparisons indicate that the algorithmic performance of AddAtom is very good, and often the best of the test bed of 11 concept lattice construction algorithms. AddAtom performs especially well compared to other algorithms with “natural” data sets (i.e. non-random generated context). When the density of the cross table of the context is either very high (i.e. every object possesses almost all attributes) or very low (every object possesses only very few attributes) there are other concept lattice construction algorithms that do outperform AddAtom. AddAtom is however still the next-to-best performer in these circumstances and therefore a worthy candidate for a general-use algorithm. AddAtom was the fastest incremental lattice construction algorithm in the study. The experimental comparison results are therefore consistent with the argument that the theoretical complexity bound of AddAtom derived here is not a very sharp upper bound for AddAtom.

Note that the discussion of both the theoretical and empirical performance, is with reference to the optimised version of the AddAtom algorithm (see section 4.6). In the comparisons, performance issues are related to constructing both the set of all concepts as well as the cover relationships (i.e. the line diagram).

5.1 A SURVEY OF CONCEPT LATTICE CONSTRUCTION ALGORITHMS

It is not the objective of this dissertation to analyse and describe other construction algorithms. Readers are referred to recent comparative studies by Kuznetsov and Obiedkov (2001, 2002) for a broad discussion and pseudo code of other algorithms. A number of optimisations of these algorithms as well as adaptations to generate the line diagram of concept lattices where the algorithm does not generate it already are also described by Kuznetsov and Obiedkov. Unless otherwise stated, references to the complexity or experimental performance of these algorithms refer to the improvements and adaptations propose by Kuznetsov and Obiedkov. Although not exhaustive, the following table lists a number of the published concept lattice construction algorithms and briefly describes all the algorithms referred to in this chapter (adapted from Kuznetsov and Obiedkov (2001, 2002)). The theoretical and experimental comparisons will be made to a subset of these algorithms.

In each case an algorithm is classified as either incremental or batch (non-incremental) and also whether it generates only the set of all concepts or the line diagram of the lattice.

Algorithm	Incremental / Batch	Notes
Chein	Batch	Chein (1969) Concepts are represented as extent-intent pairs and each new concept is generated as the intersection of the intents of two existent concepts. Similar to AI-tree. Modifications were suggested by Kuznetsov and Obiedkov (2002). Generates the set of all concepts of the lattice.
Ganter, NextClosure	Batch	Ganter (1984) Batch algorithm adding one object to earlier generated extent and calculating closure. Generate concepts in topological order using lexical order for concept lookup and comparison. Modifications were suggested by Kuznetsov and Obiedkov (2002). Generates the set of all concepts or the line diagram of the lattice.
Bordat	Batch	Bordat (1986) Batch algorithm intersecting the intent of concepts with intents of objects that don't belong to concept. Generate concepts in depth-first order using a tree for concept lookup and comparison. Modifications were suggested by Kuznetsov and Obiedkov (2002). Generates the set of all concepts or the line diagram of the lattice.
AI-tree	Batch	Zabezhalo et al. (1987) A top-down batch algorithm that searches for concepts in the set of concepts generated thus far. Similar to Chein. Generates the set of all concepts of the lattice.



Algorithm	Incremental / Batch	Notes
CbO, Close by One	Batch	Kuznetsov (1993) Batch algorithm (similar to NextClosure) adding one object to earlier generated extent and then calculating the closure. Generate concepts in depth first order using lexical order for concept lookup and comparison. Also use an intermediate structure for concept searches and the generation of the line diagram. Generates the set of all concepts or the line diagram of the lattice.
Lindig	Batch	Lindig (1999, 2000) Bottom-up batch algorithm adding one attribute at a time to the intent of generated concepts and then calculating its closure. Generate concepts in a depth-first order using tree for concept searches. Generates the diagram of the lattice.
Titanic	Batch	Stumme et al. (2000)
Yevtushenko	Batch	Yevtushenko (2002)
Norris	Incremental	Norris (1978) Incremental algorithm intersecting the new object intent with that of concepts generated earlier. Keep list of added objects, checking whether new concepts can be generated using intersection of objects added earlier. This has been described as being an incremental version of the CbO algorithm. Modifications were suggested by Kuznetsov and Obiedkov (2002). Generates the set of all concepts or the line diagram of the lattice.
Godin, GodinEx	Incremental	Godin et al. (1991, 1995b) Incremental algorithm intersecting new object intent with that of concepts generated earlier. Use a heuristic hash function to sort the concepts when generating and searching concepts. There are two versions of the algorithm: GodinEx refers to the version that uses the size of the extents and Godin the size of the intents. Modifications were suggested by Kuznetsov and Obiedkov (2002). Generates the set of all concepts or the line diagram of the lattice.



Algorithm	Incremental / Batch	Notes
Grand	Incremental	Oosthuizen (1991) Incremental algorithm using a graph theoretic approach to insert an object into a lattice. Grand connects objects attribute by attribute, to an increasing subset of its intent until the object is connected to all the attributes in its intent. During the process a function called transform ensures that the uniqueness of suprema and infima is maintained through the manipulation, addition and deletion of concepts and arcs. Constructs EA-lattices. Generates the line diagram of the lattice.
Carpineto	Incremental	Carpineto and Romano (1993, 1996b)
Nourine	Incremental	Nourine and Raynaud (1999, 2002) Incremental algorithm intersecting new object intent with that of concepts generated earlier. Use a lexical tree for concept lookup and comparison. Generates the line diagram of the lattice.
Valtchev, Divide and conquer	N/A	Valtchev et al. (2000) Algorithm based on the combination of two concept sub-lattices that are combined to construct the full lattice. The context of each sub-lattice is obtained by splitting the cross table of the original context either by objects or by attributes. Generates the line diagram of the lattice.
AddAtom	Incremental	Described in chapter 4. Incremental algorithm using the approximate and exact intent representative (minimal meets) of the object intent to find generator concepts and recursively generate new concepts above these. AddAtom use the lattice itself for concept lookup, comparison and avoiding duplicate generation of concepts. Generates the line diagram of the lattice.

5.2 A THEORETICAL PERFORMANCE BOUND FOR ADDATOM

In this section the notation used for the description of the theoretical complexity (section 5.2.1) and a number of lattice size related formulae are given (section 5.2.2 and 5.2.3). This will be used to derive an upper bound for the theoretical complexity of AddAtom (5.2.4)

5.2.1 Notation

The following notation is used for the theoretical performance of constructing a formal concept lattice of the context $C = \langle O, A, I \rangle$:

Notation	Description
$\ O\ $	The number of objects in the context.
$\ A\ $	The number of distinct attributes in the data set or context itself, not the theoretical limit of the domain from which the context was taken. As the number of objects increase, $\ A\ $ typically approaches the theoretical limit (e.g. in the case of randomly generated contexts).
$\ I\ $	The number of “crosses” in the cross table of the context. It is therefore the number of attribute-object pairs in the incidence relation. The maximum number of crosses in the cross table is $\ O\ \cdot \ A\ $.
$\ O'\ $	The average number of attributes per object in the context, i.e. the average intent size of atoms in the EA-lattice of the context. $\ O'\ = \ I\ / \ O\ $. For contexts with a varying number of attributes per object, the maximum number of attributes per object the notation $\max(\ O'\)$ is used to indicate the maximum number of objects per attribute.
$\ A'\ $	The average number of objects per attribute in the context, i.e. the average extent size of co-atoms in the EA-lattice of the context. $\ A'\ = \ I\ / \ A\ $. For contexts with a varying number the maximum number of attributes per object the notation $\max(\ A'\)$ is used to indicate the maximum number of objects per attribute.
$\ L\ $	The number of concepts in the lattice of the context including the unit and zero concepts. L_j indicates the lattice after the insertion of the j 'th object into the lattice.
$\ < \ $	The number of arcs in the line diagram of the lattice L . $\ < _j\ $ indicates the number of arcs in lattice L_j .
$\ O'\ / \ A\ $	This is referred to as the “density” of the cross table and is defined as the proportion of crosses in the cross table relative to the total number of possible crosses in the cross table (i.e. $\ I\ / (\ O\ \cdot \ A\) = \ O'\ / \ A\ = \ A'\ / \ O\ $). It can, of course, be specified as a percentage and is useful as a normalised metric to compare contexts with a different number of attributes.

5.2.2 Concept lattice size formulae

In this section a number of formulae and equations on concept lattice aspects related to size are derived. These will be used in deriving complexity bounds for AddAtom. Before deriving the actual lattice formulae, a number of generic equivalences are given. These equivalences will be used to refine the lattice formulae.

The following two generic equivalences, found in many texts on algebra and combinatorics (e.g. Cameron (1996)), can be proved by induction. The formulas are derived from the Binominal theorem.

$$\sum_{k=0}^n \binom{n}{k} = 2^n \quad (5.1)$$

$$\sum_{k=0}^n k \binom{n}{k} = n \cdot 2^{n-1} \quad (5.2)$$

The next two equivalences, can be found in texts on mathematical analysis (e.g. Clark(1931)), and are also proved via induction.

$$\sum_{k=0 \text{ to } n} a^k = \frac{1 - a^{n+1}}{1 - a} ; a \neq 1 \quad (5.3)$$

$$\sum_{k=1 \text{ to } n} k \cdot a^k = \frac{a}{(1 - a)^2} (n \cdot a^{n+1} - (n+1)a^n + 1) ; a \neq 1 \quad (5.4)$$

Figures 5.1 and 5.2 show the Boolean lattices of contexts with 3 and 4 attributes respectively. The discussion below refers to Boolean lattices L_j from a context $C = \langle O, A, I \rangle$. These figures are included here to serve as an aid in explaining the derivation of the lattice size equations 5.5 to 5.8.

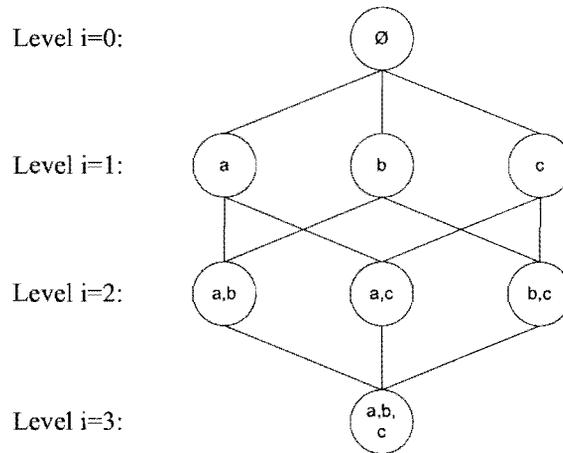


Figure 5.1: A Boolean lattice with 3 attributes (only concept intents are shown; the level of the concept is also shown)

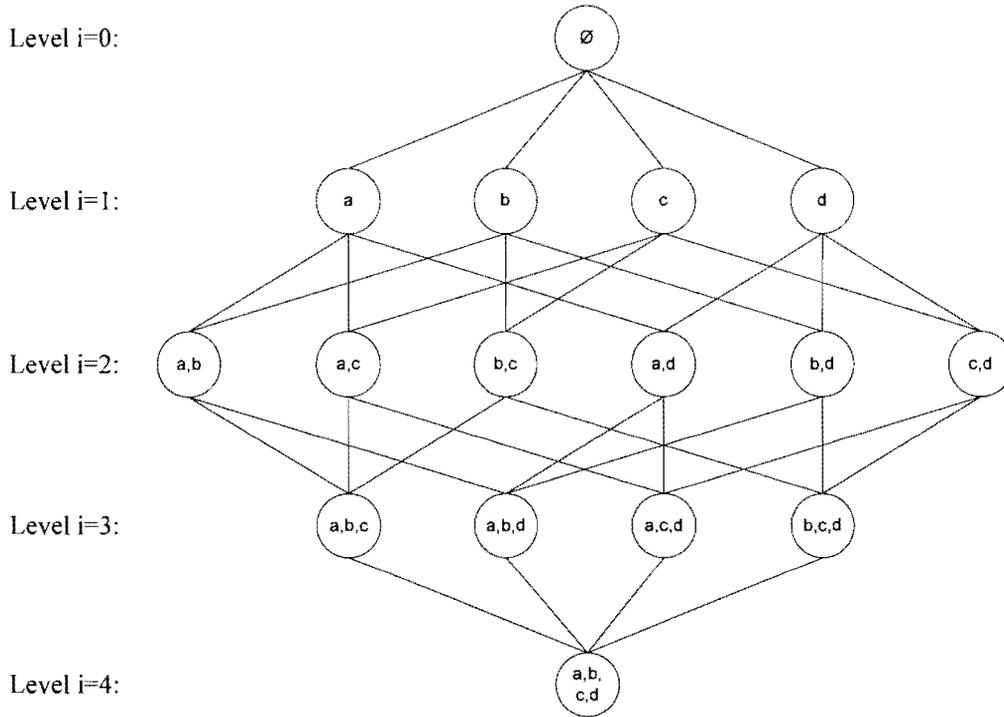


Figure 5.2: A Boolean lattice with 4 attributes (only concept intents are shown; the level of the concept is also shown)

For the purpose of discussion, the concepts in the Boolean lattice will be divided into a number of levels, where the number of attributes in the intent of the concept indicates its level. The variable i will indicate the level. Where multiple, successive lattices are under discussion, the variable j will indicate the j 'th lattice in the sequence of lattices (i.e. after the insertion of the j 'th object). The equations in the table below characterise important aspects of Boolean lattices related to size. Note that for theoretical purposes an initial lattice consisting only of a single concept with an empty intent and -extent called L_0 with $\|L_0\| = 1$ and $\|<\| = 0$ is included in the equations. For the sake of simplicity only $\|A\|$ is used since $\|O\| = \|A\|$ for Boolean lattices. The remarks indicate how these equations have been derived. These equations hold for Boolean lattices. It is assumed that $\|O\| > 0$ and $\|A\| > 0$.

Equation	Remark	Nr
$\ L\ = \sum_{i=0}^{\ A\ } \binom{\ A\ }{i}$ $= 2^{\ A\ }$	The total number of concepts on each level i of a Boolean lattice is the number of distinct combinations of subsets of A of size i . The final result follows from equation 5.1.	(5.5)
$\ <\ = \sum_{i=0}^{\ A\ } i \binom{\ A\ }{i}$ $= \ A\ 2^{\ A\ -1}$ $= \frac{1}{2} \cdot \ A\ \cdot \ L\ $	Inspecting figures 5.1 and 5.2 it can be seen that each concept on any level i , has i arcs leading to its i parents. Once again the number of concepts on level i is the number of distinct subsets of A of size i . After counting the	(5.6)

total number of arcs,
equation 5.2 is used to
simplify the result.

$$\begin{aligned} \sum_{j=0}^{\|A\|} \|L_j\| &= \sum_{j=0}^{\|A\|} 2^j \\ &= 2^{\|A\|+1} - 1 \\ &= 2 \cdot \|L\| - 1 \end{aligned}$$

The total number of concepts in all lattices L_j , $j = 0$ to $\|A\|$ follows from combining equation 5.5 and 5.3. (5.7)

$$\begin{aligned} \sum_{j=0}^{\|A\|} \|\prec_j\| &= \sum_{j=0}^{\|A\|} j \cdot 2^{j-1} \\ &= \sum_{j=0}^{\|A\|-1} j \cdot 2^j + \sum_{j=0}^{\|A\|-1} 2^j \\ &= (\|A\| - 1) \cdot 2^{\|A\|} + 1 \\ &= (\|A\| - 1) \cdot \|L\| + 1 \end{aligned}$$

The total number of arcs in all lattices L_j , $j = 0$ to $\|A\|$ can be derived by combining equations 5.6, 5.3 and 5.4. (5.8)

Non-Boolean concept lattices

Boolean FCA lattices contain the maximum number of possible concepts (i.e. unique combinations of intent and extent) for a given number of arcs and therefore contexts that do not give rise to Boolean lattices have fewer concepts in their lattices. The size of arbitrary lattices is therefore bound by the minimum of the unique number of extents or intents possible, i.e. $2^{\min(\|A\|, \|O\|)}$.

The number of outbound arcs is bound by the unique combinations of attributes in the intents of its parents and/or the unique combinations of objects in the extents of its parents. In a Boolean lattice, the number of possible unique intents of the parents of a concept c is $\|\text{Intent}(c)\| - 1$, but non-Boolean lattices may potentially have more (up to $\binom{\text{Intent}(c)}{\text{Intent}(c)+2}$). (Using the extents of the parent concepts provides a sharper bound to the number of outward arcs. The parent concepts $p_1 \dots p_n$, of a concept c must be unique and therefore have at least one object in their extents in addition to that of c , i.e. $\text{Extent}(p_i) \supset \text{Extent}(c)$. Furthermore, for any two parent concepts, p_i and p_j , $p_i \neq p_j$ of c , $\text{Extent}(p_i) \cap \text{Extent}(p_j) = \text{Extent}(c)$. Parent concepts can therefore have no object in common with the extent of any other parent concept except that of c . The extent of any concept must also be unique in the lattice. Given these constraints, the maximum number of parent concepts of any c is therefore $\|O\| - \|\text{Extent}(c)\|$ since each parent concept will have one at least additional concept of O in addition to $\text{Extent}(c)$. A bound for the maximum number of outbound (upward) arcs of any concept in a lattice is therefore $\|O\|$. In practice however the maximum number of outbound arcs may be fewer.

Using a similar argument, but based on the intent of any concept $\|A\|$ is the maximum number of inbound (downward) arcs into any concept in a lattice.

Using these bounds as a base it is clear that for non-Boolean lattices of any substantial size the number of outbound arcs $\|\prec_{\text{Outbound}}\| \leq \|O\| \cdot \|L\|$. Using a similar argument $\|\prec_{\text{Inbound}}\| \leq \|A\| \cdot \|L\|$. Since the number of outbound- and inbound arcs in any lattice should be equal to the total number of arcs, $\|\prec\| \leq \min(\|A\|, \|O\|) \cdot \|L\|$.

Most contexts used in practical applications have $\|A\| < \|O\|$. It is assumed that $\|O\| > 0$ and $\|A\| > 0$. The following inequalities hold in such cases:

Equation

Nr

$$\begin{aligned} \|L\| &\leq 2^{\min(\|A\|, \|O\|)} \\ &\leq 2^{\|A\|} \end{aligned} \quad (5.9)$$

$$\|\prec\| \leq \min(\|A\|, \|O\|) \cdot \|L\| \quad (5.10)$$

$$\sum_{j=0}^{\|O\|} \|L_j\| \leq \|O\| \cdot \|L\| + 1 \quad (5.11)$$

$$\begin{aligned} \sum_{j=0}^{\|O\|} \|\prec_j\| &\leq \min(\|A\|, \|O\|) \cdot \|O\| \cdot \|L\| + 1 \\ &\leq \|A\| \cdot \|O\| \cdot \|L\| + 1 \end{aligned} \quad (5.12)$$

From equation 5.6 it can be seen that a Boolean lattice contains, on average $\frac{1}{2} \cdot \|A\|$ outbound arcs per concept and also on average $\frac{1}{2} \cdot \|A\|$ inbound arcs per concept since the total number of outbound- and inbound arcs in a lattice are equal. It is therefore clear that the above equations do not always provide very sharp upper bounds. Where $\|L\|$ is exponential in terms of $\|A\|$ or $\|O\|$ it may be better to use equations 5.5 to 5.8 and substitute $\|L\| = 2^{\|A\|}$.

5.2.3 Complexity of set operations

For the purposes of calculating complexity upper bounds, it is assumed that sets are implemented as ordered lists defined using fixed length arrays. A linear order relationship is assumed to be defined on all possible elements of the set (i.e. set is completely ordered as opposed to partially ordered). This does not affect the result of the algorithms but will avoid unnecessary iterations and searches through the unordered elements of a set. A typical strategy is to number all concepts and implement sets as bit strings with set membership in the set indicated by the bit that correspond to the concept number. This takes advantage of modern CPU architectures with 32 and 64 bit, bitwise operations to improve the efficiency of set operations. Effectively this means that the following complexity bounds will be used on sets:

Operation	Complexity
Set operations: union, copy/assignment, set cardinality	$O(\ Set_1\ + \ Set_2\)$
Set operations: test for subset and proper subset (\subset and \subseteq), test for set equality, set intersection (\cap)	$O(\max(\ Set_1\ , \ Set_2\))$
Single element insertions	$O(1)$
Test for set membership for single element	$O(1)$
Set initialisation	$O(\ Set_1\)$
Set cardinality	$O(\ Set_1\)$

For the union, copy/assignment, set cardinality operations on concept intents the bound $O(\|A\|)$ is used whilst the bound $O(\|O\|)$ is used for set operations on concept extents. For subset and proper subset testing, test for set equality and set intersection operations on concept intents $O(\max(\|O'\|))$ is used, whilst $O(\max(\|A'\|))$ is used for concept extents.

These bounds are however not very sharp since in implementation a single CPU operation would for example perform 32 or 64 comparisons on set elements.

5.2.4 AddAtom theoretical performance

The theoretical (worst case) performance of lattice construction algorithms is expressed using the input and output sizes of the algorithms. This is done in two ways: firstly, as the time complexity associated with the construction of the complete lattice of the context. Since the output size is exponential, a second complexity metric called the delay is also used. An algorithm for listing a family of combinatorial structures is said to have *polynomial delay* (Johnson et al. 1988) if it executes at most polynomially many computational steps before either outputting each next structure or terminating. An algorithm is said to have a *cumulative delay* d (Goldberg 1993) if at any point in any execution of the algorithm with any input p the total number of computational steps that have been executed is at most $d(p)$ plus $K.d(p)$ where K is 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*.

The number of concepts of the lattice is exponential in the worst case (i.e. a Boolean lattice). Furthermore, the problem of determining the number of concepts in the lattice is NP-complete (Kuznetsov 1989, 2001). In this sense, any lattice construction algorithm unavoidably has intractable (i.e. exponential) worst case behaviour, both in time (since each node has to be generated) and in space (since each node has to be stored). Lattice construction algorithms are therefore differentiated in terms of their time delay characteristics. An algorithm can therefore be considered efficient if it generates the lattice with a polynomial time delay and space linear in the number of all concepts in the lattice. Although “dense” contexts that approach this limit may not be used very often in practice, the theoretical complexity of an algorithm nevertheless expresses an aspect of its performance and is therefore relevant.

A bound for the theoretical worst-case time complexity of AddAtom will be shown below to be $O(\|L\| \cdot \|O\|^2 \cdot \max(\|O'\|))$. (The discussion will be based on the optimised form of this construction algorithm, as described in section 4.6.)

As an aid to the discussion, appendix B contains an outline of the algorithm, highlighting its main loops and instructions that add to its complexity characteristics, assist in the analysis of the complexity.

One approach to estimating an upper time bound for constructing the lattice, L , from scratch, is to consider $AddAtom_{o_j}$ as the upper bound for inserting a single object, o_j , into the lattice (including all the time required for all the recursive calls to AddAtom and all the calls to GetMeet). Let $Housekeeping_{o_j}$ be the upper bound for doing the housekeeping in preparation for inserting o_j into the existing lattice but excluding the calls to AddAtom. The upper time bound for constructing L would then be:

$$O(\sum_{j=1}^{to \|O\|} AddAtom_{o_j} + \sum_{j=1}^{to \|O\|} Housekeeping_{o_j})$$

However, instead of attempting to derive upper bounds on each $AddAtom_{o_j}$, another more global line of reasoning route will be followed.

To this end, let $AddAtom_Total$ be the upper time bound on executing all instructions relating to all calls to AddAtom, in order to insert all objects into L including the calls to GetMeet. Let $Housekeeping_Total$ be the upper bound for the total amount of time taken for the housekeeping and preparation for the construction of the complete lattice. The complexity of the algorithm would then be bounded by:

$$O(AddAtom_Total + Housekeeping_Total)$$

It will be shown below that an upper bound on AddAtom_Total is $O(\|L\| \cdot \|O\|^2 \cdot \max(\|O'\|))$, and that an upper bound on $\text{Housekeeping_Total}$ is $O(\|L\| \cdot \|O\| \cdot \|A\|)$. Under these assumptions, an upper bound on the algorithm to construct the lattice is then:

$$O(\|L\| \cdot (\|O\|^2 \cdot \max(\|O'\|) + \|O\| \cdot \|A\|))$$

Since we are interested in order of magnitude estimates of the time for constructing a lattice, L , the lesser term may be left out since it will be dominated by the greater when constructing large lattices. A resulting upper bound (i.e. worst case) estimate for constructing L is thus $O(\|L\| \cdot \|O\|^2 \cdot \max(\|O'\|))$.

The following three subsections deal with the complexity of each of the three parts of the algorithm.

AddAtom complexity

Looking at the functioning of AddAtom and its parameters, it is clear that there is only one recursive call made to AddAtom for each concept in the lattice. This is since concepts are only created within AddAtom and there are no concepts that are deleted or duplicated. The maximum number of generator concepts for all the lattices L_j is in fact the total number of concepts in the lattice (i.e. $\|L\|$). Furthermore, for each generator concept that is used as parameter to AddAtom , the outer for loop (using candidate as variable) is executed for each of its parent concepts (a maximum of $\|O\|$ times for each generator concept). The maximum number of iterations of the outer for loop across all invocations of AddAtom would therefore coincide with $O(\|L\| \cdot \|O\|)$.

Within the first and outer for loop of AddAtom , the maximum number of algorithmic steps is determined by the maximum number of steps taken by GetMeet or the inner for loop (using g as variable), whichever is biggest. NewConcept contain only concepts that are prospective parents for the new concept and this list is reduced during each iteration. NewConcept 's number of elements is bound by the maximum number of parents of any concept i.e. $O(\|O\|)$. Within the inner for loop a number of set operations on sets of concept intents are executed. The most complex of these operations is the subset and proper subset tests which is bound by $O(\max(\|O'\|))$. Therefore the number of steps taken by the inner for loop during each iteration of the outer for loop is bound by $O(\|O\| \cdot \max(\|O'\|))$.

The complexity of the last for loop is dominated by the others and therefore it is not considered in the complexity bound.

Below it will be argued that the complexity of a single call to GetMeet is bound by $O(\|O\| \cdot \max(\|O'\|))$. The number of algorithmic steps taken by all invocations of AddAtom to inset all objects into the lattice is therefore bound by $O(\|L\| \cdot \|O\| \cdot (\|O\| \cdot \max(\|O'\|) + \|O\| \cdot \max(\|O'\|))) = O(\|L\| \cdot \|O\|^2 \cdot \max(\|O'\|))$.

GetMeet complexity

GetMeet traces a path between the parent of a generator concept and a meet of a subset of $\text{Intent}(o)$ somewhere above it. The maximum number of iterations of the outer while loop is bounded by the number of attributes in the intent of generator (i.e. $O(\max(\|O'\|))$). The maximum number of times the for loop can be executed is bounded by the maximum number of parents of a concept (i.e. $\|O\|$) since each parent has at least one attribute less in its intent. Since the instructions within the while loop is of $O(1)$ complexity, the complexity of a single call to GetMeet is $O(\|O\| \cdot \max(\|O'\|))$.

Housekeeping_Total complexity

The complexity bound of $\text{Housekeeping_Total}$ is determined by the second and outer for loop (with o as variable). Within it the two inner for loops are executed $O(\|L_j\|)$ times per

object – i.e. $O(\sum_{j=1}^{\|O\|} \|L_j\|) \approx O(\|L\| \cdot \|O\|)$ times for inserting *all* objects. Within these for loops the number of algorithmic steps of set operations executed are bounded by $O(\|A\|)$ and $O(1)$ for the first and second for loops respectively. The complexity of Housekeeping_Total is therefore bounded by $O(\|L\| \cdot \|O\| \cdot \|A\|)$.

Theoretical complexity comparison

The following table summarises the algorithmic complexity for other construction algorithms⁶

Algorithm	Incremental / Batch	Complexity
Bordat	Batch	Time complexity = $O(\ O\ \cdot \ A\ ^2 \cdot \ L\)$ Polynomial delay = $O(\ O\ \cdot \ A\ ^2)$
CbO, Kuznetsov	Batch	Time complexity = $O(\ O\ ^2 \cdot \ A\ \cdot \ L\)$ Polynomial delay = $O(\ O\ ^3 \cdot \ A\)$
Chein	Batch	Time complexity = $O(\ O\ ^3 \cdot \ A\ \cdot \ L\)$ Polynomial delay = $O(\ O\ ^3 \cdot \ A\)$
Dowling	Incremental	Time complexity = $O(\ O\ ^2 \cdot \ A\ \cdot \ L\)$
Godin	Incremental	Time complexity = $O(\ L\ ^3)$
Lindig	Batch	Time complexity = $O(\ O\ ^2 \cdot \ A\ \cdot \ L\)$ Polynomial delay = $O(\ O\ ^2 \cdot \ A\)$
NextClosure, Ganter	Batch	Time complexity = $O(\ O\ ^2 \cdot \ A\ \cdot \ L\)$ Polynomial delay = $O(\ O\ ^2 \cdot \ A\)$
Norris	Incremental	Time complexity = $O(\ O\ ^2 \cdot \ A\ \cdot \ L\)$
Nourine	Incremental	Time complexity = $O((\ O\ + \ A\) \cdot \ O\ \cdot \ L\)$
Valtchev	N/A	The complexity of the procedure assembling lattices L_1 and L_2 into the global lattice L is $O((\ O\ + \ A\)(\ L_1\ \cdot \ L_1\ + \ L\ \cdot \ A\))$ L_1 and L_2 can however be built in parallel.
AddAtom	Incremental	Time complexity = $O(\ L\ \cdot \ O\ ^2 \cdot \max(\ O'\))$

For the purpose of direct comparison and since $\|O'\| < \|A\|$, $\|O'\|$ can be substituted with $\|A\|$. A slightly less sharp complexity bound for AddAtom is therefore $O(\|L\| \cdot \|O\|^2 \cdot \|A\|)$.

The AddAtom complexity estimate is therefore cubic in nature relative to the lattice size. This is a feature that it shares with most other algorithms. Since this estimate is not quadratic relative to the number of concepts, as is the Nourine algorithm, it might seem that AddAtom does not offer very much in terms of theoretical performance overall.

The complexity bound as stated is however not very sharp. One area where the theoretical complexity is overstated is within GetMeet. The maximum length of a path in GetMeet is stated as $\|O'\|$ but in general no path would stretch from O_L to I_L (implied by a

⁶ Where these algorithms have been improved as discussed in Kuznetsov and Obiedkov (2002), the complexity of the improved algorithm is given.

path length of $\|O'\|$). It is interesting to note that if it could be proved that GetMeet return each of the respective meet concepts above a particular generator concept only once, the total combined length of all paths traced in calls to GetMeet to insert a single object would not exceed the total number of concepts in the lattice. This is because none of such paths can cross each other except at the meet of a subset of $\text{Intent}(o)$. Under this assumption the maximum number of iterations of inner for loop for each concept on the path is the number of parents of that concept. The total number of iterations of the for loop across all invocations for the insertion of one object is therefore the total number of arcs in the lattice. Therefore $O(\sum_{j=1}^{\|O\|} \|j\|) \leq O(\|A\| \cdot \|O\| \cdot \|L\|)$ (or $O(\frac{1}{2}\|A\| \cdot \|L\|)$ in the case of a Boolean lattice) would be an upper bound on the complexity of all calls to GetMeet across all the recursive calls to AddAtom to insert all objects of the context (for Boolean lattices that is). For the algorithm as stated in section 4.6, used in the wide comparison study in section 5.4 this is not the case, but the version of the algorithm in appendix A makes use of this optimisation. The complexity bound derived here is however still an upper bound for this algorithm.

Another area where the theoretical complexity bound is not very sharp is in the AddAtom part of the algorithm. The theoretical complexity bound assumes that the number of iterations of the outer for loop is bounded by the number of arcs in the lattice. In the algorithm itself however, only concepts with at least some attribute in common with the to-be inserted object will be visited and therefore not all arcs will be “followed” during the iterations of the for loop. For non-Boolean lattices with $\|O'\| \ll \|A\|^7$ this will be a very significant portion of the concepts in the lattice that will not be visited by the for loop. To quantify this further, consider a Boolean lattice and an object intent o' . There are in general $2^{\|A\| - \|o'\|}$ concepts in the lattice that have no attribute in common with o' . Clearly for non-Boolean lattices this number will be significantly less, but for many contexts this is still very significant, indicating an overestimation of the overall complexity.

The use of $\|O\|$ as the upper bound to the number of parents of a lattice leads to an overestimate of the total number of arcs in a lattice. A case in point is the fact that Boolean lattices have on average $\frac{1}{2}\|A\|$ inbound or outbound arcs per concept – far fewer than the upper bound $\|O\|$ used here.

The AddAtom algorithm can be easily adapted to be symmetrical and insert attributes into the lattice and link them to their extents instead of inserting objects into the lattice and linking them to their intents. Using the same reasoning AddCoatom, the dual incremental concept lattice construction algorithm would have a complexity bound of $O(\|L\| \cdot \|A\|^2 \cdot \max(\|A'\|))$ which may include smaller terms than that of AddAtom.

The best way to obtain clarity on this and other issues is via empirical studies. The next two sections present the results of the pilot and wider empirical studies. The results of the empirical studies support the claims on the over estimation of the theoretical complexity of AddAtom and indicate that it does indeed perform very well and is often the best performer of the algorithms surveyed.

5.3 EMPIRICAL PERFORMANCE: PILOT STUDY

The pilot study was conducted to establish the relative performance of AddAtom using the code described in chapter 7 to seek *prima facie* evidence that would justify a wider study. The basic strategy of the pilot study was to implement the incremental lattice construction

⁷ The notation $a \ll b$ indicates that A is significantly smaller than b by some measure.

algorithms of Godin et al. (1995b)⁸ and Carpineto and Romano (1993) using the same base code and data structures as AddAtom (described in chapter 7). The pseudo code of the implemented algorithm can be found in appendix A (note that there are differences to the algorithm in section 4.6). This would serve as a good indication of the relative performance of the algorithm and clearly indicate if the time performance was worse (or not) than that of the Godin or Carpineto algorithms, justifying the effort of a wider study.

Note that for the pilot study EA-lattices were generated and the Godin and Carpineto algorithms were modified to generate EA-lattices.

In addition to the Godin and Carpineto algorithms, the Grand algorithm (Oosthuizen (1991)) was also available for comparison but due to it using different data structures and utility functions as well as being implemented in a different programming language (refer to chapter 7 for further discussion), it was not included in the study since it would not make an apples-with-apples comparison possible. The performance of Grand is however worse than AddAtom in all types of contexts by a significant margin.

The pilot study comparison showed that AddAtom is indeed faster than the Godin and Carpineto algorithms and this suggested that a more thorough study of the algorithm's performance would be worth while. However, it also exposed the fact that the code base and data structures were inefficient and that a wider study would require a revised strategy towards the data structures and utility functions (also refer to chapter 7).

For the pilot study, care was taken to ensure a valid comparison. To this end, the algorithms were implemented on the same base-code and performance tests run under the same platform. However, any inefficiency in the particular implementation approach and data structures could have negatively penalised the relative performance of the Godin and Carpineto algorithms. This is because the data structures used could have conceivably suited AddAtom better and could have given it an unfair advantage under the experimental comparison. To avoid this situation from influencing the outcome, a number of additional performance metrics, other than time, were collected. These metrics tracked basic lattice operations such as lattice closures and set operations and did confirm the trend of the time based results.

A number of artificial and "natural" data sets were used as contexts for the experimental comparisons. The artificial data sets were randomly generated whilst the natural data sets were taken from the well-known UCI Machine Learning Repository (Blake and Merz 1998).

The following table provides an overview of the data sets and describes the notation used to identify the data sets.

Data set	Description
Rnd-100-YY-XXX	A random data set of XXX objects. Each object possesses exactly YY attributes, randomly chosen from 100 possible attributes. When referring to the data set as a whole, the notation Rnd-100-YY is used.
Bool-XX	A data set of XX objects. The data set has XX attributes. Every object has XX – 1 attributes and differs from each of the other objects in only one attribute. The resulting lattice of this arrangement forms a Boolean lattice. When referring to the data set as a whole, the notation Bool is used.

⁸ The implementation follows the description in Godin et al. (1995b) and not the improvements suggested by Kuzetnov and Obiedkov (2002).

Data set	Description
SPECT	A natural data set taken from the UCI repository called the Single Proton Emission Computed Tomography (SPECT) set. The dataset has 22 binary feature patterns and one overall diagnosis attribute. When referring to the data set as a whole, the notation SPECT is used.
BCW-XXX	The Breast-Cancer-Wisconsin natural data set taken from the UCI repository. XXX indicates the number of objects in the context. Objects were randomly selected from the data set. The set of discrete attributes was used unaltered. The total data set consists of 698 objects, each object has 10 attributes, whilst each of the 10 attributes could assume any one of 10 values. Some objects do not possess a value for a specific attribute (the value is unknown). Such objects were still included in the set and the unknown value was treated as an eleventh value of that specific attribute. Each value of each attribute was treated as a separate attribute in the experimental results. Theoretically there were thus $10 \times 11 = 110$ attributes, but in practice the data set contained only 86 attributes since all attribute values were not observed. When referring to the data set as a whole, the notation BCW is used.

The key metrics describing the data sets that were used are as follows:

Set name	$\ O\ $	$\ A\ $	$\ I\ $	$\ L\ $	$\ <\ $	$\ O'\ / \ A\ $
Rnd-100-10-40	40	98	379	312	871	10%
Rnd-100-10-45	45	100	434	351	990	10%
Rnd-100-10-50	50	98	477	413	1198	10%
Rnd-100-10-75	75	100	725	697	2197	10%
Rnd-100-10-100	100	100	975	1058	3425	10%
Rnd-100-10-150	150	100	1433	1957	6567	10%
Rnd-100-10-200	200	100	1915	3031	10423	10%
Rnd-100-30-15	15	100	392	426	1206	26%
Rnd-100-30-20	20	100	520	799	2588	26%
Rnd-100-30-25	25	100	643	1313	4589	26%
Rnd-100-30-30	30	100	779	2183	7962	26%
Rnd-100-30-35	35	100	914	3329	12623	26%
Rnd-100-30-40	40	100	1039	4288	16652	26%
Bool-07	7	7	42	128	448	86%
Bool-08	8	8	56	256	1024	88%



Set name	O	A	I	L	<	O' / A
Bool-09	9	9	72	512	2304	89%
Bool-10	10	10	90	1024	5120	90%
Bool-11	11	11	110	2048	11264	91%
Bool-12	12	12	132	4096	24576	92%
BCW-030	30	69	300	240	564	14%
BCW-035	35	71	350	317	795	14%
BCW-040	40	75	400	312	751	13%
BCW-045	45	77	450	323	783	13%
BCW-050	50	84	500	499	1349	12%
BCW-075	75	84	750	701	1948	12%
BCW-100	100	84	1000	1091	3331	12%
BCW-200	200	86	2000	1704	5455	12%

Tests for the pilot study were performed on an Intel 110 mhz Pentium processor based platform with 256 megabytes of memory under the Windows 2000 Professional operating system. Note that EA-lattices were generated for the pilot study.

The following graphs summarise the results.

Pilot study - time (Rnd-100-10)

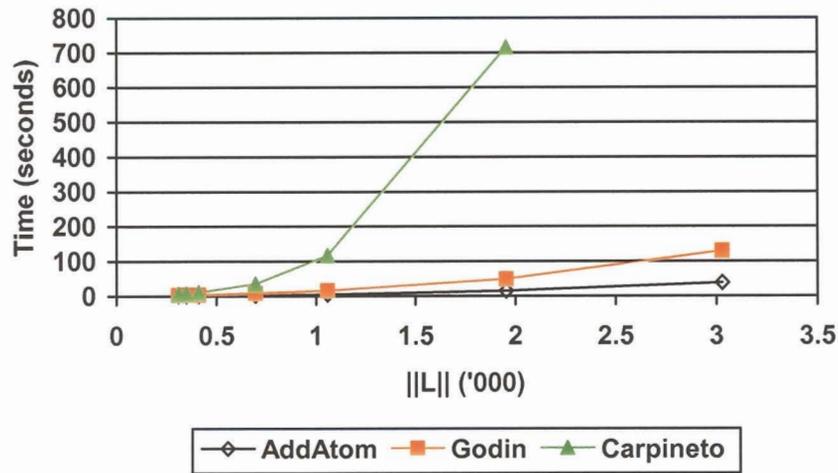


Figure 5.3: Pilot study performance test results for the Rnd-100-10 data set plotting time performance against lattice size ($||O||/|A| = 10\%$)

Pilot study - time (Rnd-100-30)

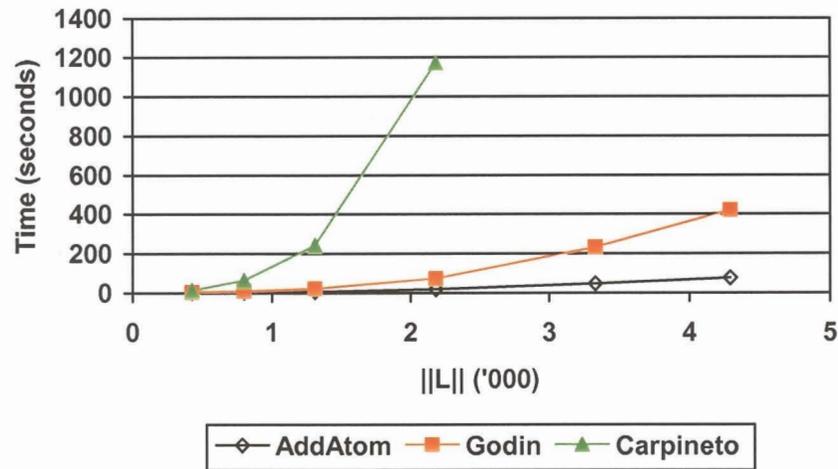


Figure 5.4: Pilot study performance test results for the Rnd-100-30 data set plotting time performance against lattice size ($||O||/|A| = 26\%$)

Pilot study - time (Bool)

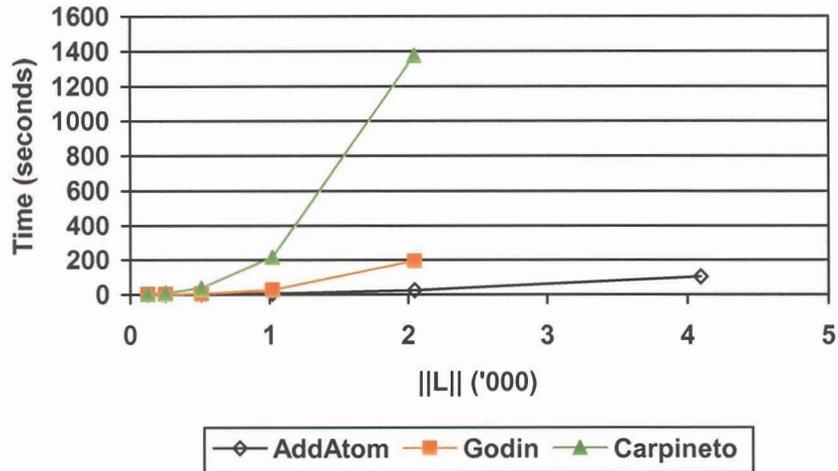


Figure 5.5: Pilot study performance test results for the Bool data set plotting time performance against lattice size ($||O' ||/||A|| = 86-92\%$)

Pilot study - time (BCW)

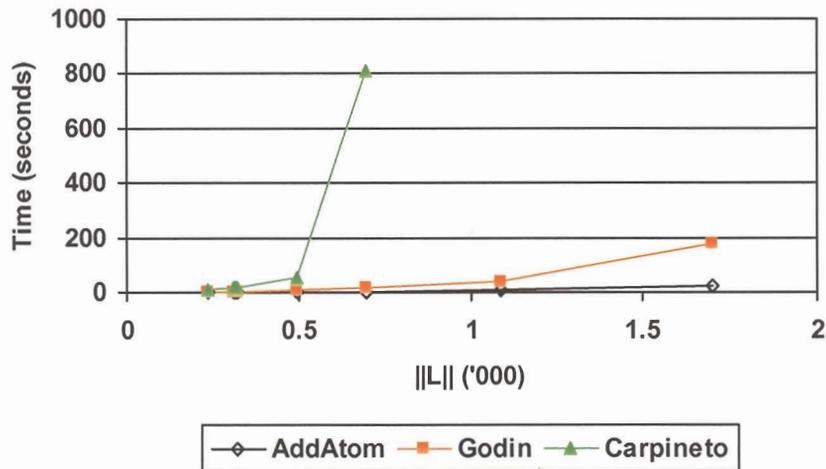


Figure 5.6: Pilot study performance test results for the BCW data set plotting time performance against lattice size ($||O' ||/||A|| = 14-12\%$)

The following table summarise the relative algorithmic performance on the largest contexts in the experimental comparison (the construction time is expressed as a percentage of the time of the fastest algorithm for a particular context):

	Performance index (fastest algorithm = 100%)						
	Rnd-100-10		Rnd-100-30		Bool	BCW	
$ L $	1955	3029	2183	4288	2046	699	1702
Context density	10%	10%	26%	26%	86-92%	14-12%	14-12%

AddAtom	100%	100%	100%	100%	100%	100%	100%
Godin	320%	337%	360%	540%	804%	400%	738%
Carpineto	4753%		5840%		5717%	27067%	

Discussion: The results clearly indicate that the Carpineto algorithm is not very efficient in constructing lattices of any type of context. The AddAtom algorithm performance was consistently better than that of the Godin algorithm. The relative performance was however not the same across the different contexts. The Godin algorithm performed worst, compared to AddAtom in the Boolean and BCW contexts.

Set operations are operations such as the union, intersection, subtraction etc. The graph below compares the number of set operations used by each of the algorithms in the construction of the lattices. (Similar results were obtained with all data sets.)

Pilot study - set operations (Bool)

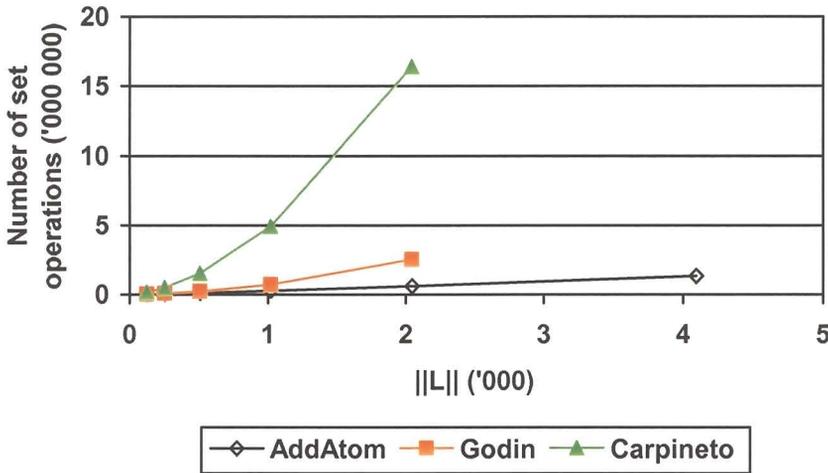


Figure 5.7: Pilot study performance test results for the Bool data set plotting the number of set operations against lattice size ($||O||/||A|| = 86-92\%$)

Node references are instructions that require the reference to a member variable of a concept object (e.g. its intent, extent or testing whether it is a concept that still exists in L). The graph below compares the number of node operations used by each of the algorithms in the construction of the lattices.

Pilot study - node references (Bool)

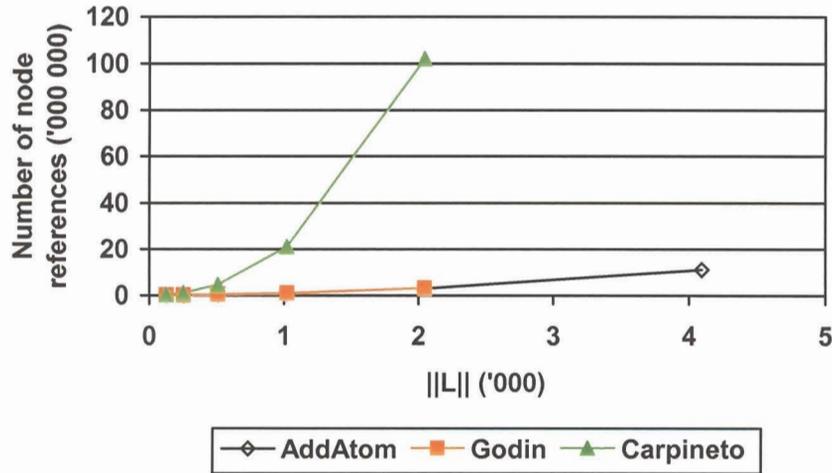


Figure 5.8: Pilot study performance test results for the Bool data set plotting the number of node operations against lattice size ($||O' ||/||A|| = 86-92\%$)

The graph below compares the number of closure operations used by each of the algorithms in the construction of the lattices.

Pilot study - closure operations (Bool)

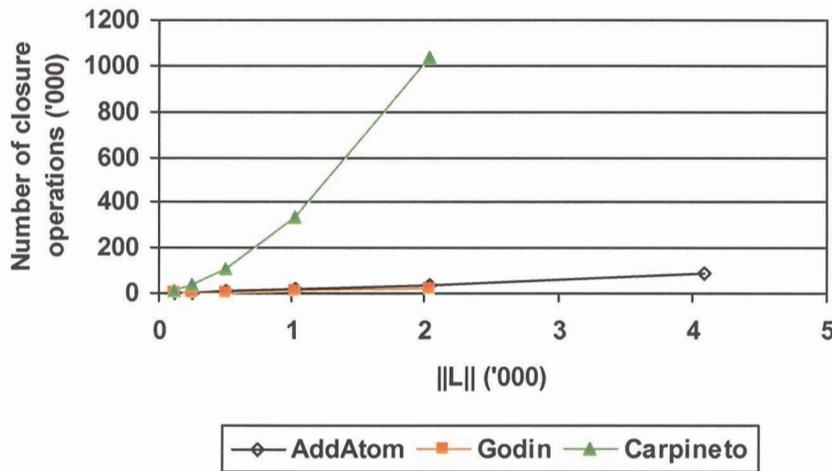


Figure 5.9: Pilot study performance test results for the BCW data set plotting the number of closure operations against lattice size ($||O' ||/||A|| = 86-92\%$)

For the sake of brevity, all performance test results are not included here. It is interesting to note that the biggest performance gap was in the natural data set (BCW). The results show that the number of operations in general follows the trend of the time-based results, except that the Godin algorithm uses less closure operations. The use of less closure operations is a feature of the Godin algorithm which is defined in terms of set operations on intents and extents rather than closure operations and is therefore not inconsistent with the results.

The results of the pilot experimental performance tests indicate that AddAtom indeed performs very well compared to the Godin and Carpineto implementations. The relative performance was good both from a time and number of basic operations perspective. This result is in line with the relative theoretical performance of AddAtom.

However the time taken to construct the lattices was well outside that reported by other authors such as Kuznetsov and Obiedkov (2001, 2002) when compensating for the differences in CPU size. This issue is further discussed chapter 7 and is mainly due to structural inefficiencies in the data structures and utility functions used. In addition, the tests did not include comparisons with the current best incremental lattice construction algorithms (e.g. that of Nourine with the best theoretical complexity). The performance tests were therefore not conclusive and lead to a further set of wider experimental comparisons.

In the wider performance comparison discussed in the next section it can be seen that the relative performance of the Godin implementation is in line with the results obtained in the pilot study and that AddAtom indeed perform better than other algorithms in most contexts.

5.4 EMPIRICAL PERFORMANCE: WIDE STUDY

In the past number of years a number of lattice construction algorithms with improved performance have been published. Meanwhile the computing power and memory capacity of even personal computers are now sufficient for building very large lattices. The result of these two factors is that the relative importance of fast and efficient construction algorithms has decreased whilst making more advanced applications using larger concept lattices possible. For practical purposes it is however still relevant to make use of the best known algorithms wherever possible since computing resources are not infinite.

The first comparative study of several algorithms was Guénoche (1990) whilst more recently Kuznetsov and Obiedkov (2001, 2002) have studied a large number of the most well-known algorithms. Kuznetsov and Obiedkov (2002) is currently the most comprehensive study of algorithmic performance and therefore provide a useful benchmark with regards to the algorithmic performance of lattice construction algorithms. The author has collaborated with S.A. Obiedkov (2003) and a version of the AddAtom algorithm adapted to Obiedkov's base code and data structures was implemented and included in Obiedkov's on-going research. The results⁹ show that AddAtom performs very well in constructing lattices from most data sets compared to other incremental and non-incremental algorithms and is often the best performer.

The key metrics describing the data sets that were used are as follows:

Data set used	$\ O\ $	$\ A\ $	$\ I\ $	$\ L\ $	$\ <\ $	$\ O'\ / \ A\ $
Rnd-100-4-100	100	100	400	2222	4936	4%
Rnd-100-4-200	200	100	800	4256	9939	4%
Rnd-100-4-300	300	100	1200	6522	15496	4%
Rnd-100-4-400	400	100	1600	9153	22050	4%

⁹ The author acknowledge contribution of S.A. Obiedkov in conducting the performance tests and making the data available for inclusion in section 5.4.



Data set used	O	A	I	L	<	O' / A
Rnd-100-4-500	500	100	2000	12032	29598	4%
Rnd-100-4-600	600	100	2400	15093	37943	4%
Rnd-100-4-700	700	100	2800	18213	46792	4%
Rnd-100-4-800	800	100	3200	21511	56601	4%
Rnd-100-4-900	900	100	3600	24816	66522	4%
Rnd-100-25-010	10	100	250	1286	3568	25%
Rnd-100-25-020	20	100	500	7057	23568	25%
Rnd-100-25-030	30	100	750	18962	69122	25%
Rnd-100-25-040	40	100	1000	37296	143136	25%
Rnd-100-25-050	50	100	1250	63118	251416	25%
Rnd-100-25-060	60	100	1500	95619	391827	25%
Rnd-100-25-070	70	100	1750	135618	569250	25%
Rnd-100-25-080	80	100	2000	181877	778487	25%
Rnd-100-25-090	90	100	2250	236729	1031820	25%
Rnd-100-25-100	100	100	2500	300257	1329793	25%
Rnd-100-50-10	10	100	500	5537	22152	50%
Rnd-100-50-20	20	100	1000	128748	644901	50%
Rnd-100-50-30	30	100	1500	752491	4112730	50%
Rnd-100-50-40	40	100	2000	2493490	14388396	50%
Bool-10	10	10	90	1024	5120	90%
Bool-11	11	11	110	2048	11264	91%
Bool-12	12	12	132	4096	24576	92%
Bool-13	13	13	156	8192	53248	92%
Bool-14	14	14	182	16384	114688	93%
Bool-15	15	15	210	32768	245760	93%
Bool-16	16	16	240	65536	524288	94%
Bool-17	17	17	272	131072	1114112	94%
Bool-18	18	18	306	262144	2359296	94%
SPECT	267	23	2042	21549	110589	33%

The following set of algorithms that generate the line diagram of a (CFA) lattice were used for the comparison. Note that these algorithms incorporate the improvements and changes described in Kuznetsov and Obiedkov (2002) alluded to between brackets.

- CbO.
- Ganter (use binary search to find the canonical generation of a concept).
- Norris (use a tree).
- Bordat (not using a tree).
- Godin (use heuristic based on the size of concept extents).
- Lindig.
- Nourine.
- Valtchev (using horizontal splitting of the object set).
- AddAtom (interpretation of Obiedkov (2003) similar to the algorithm in section 4.6 but using a different version of GetMeet that has a greater complexity bound namely $O(\max(\|O\|^2, \|O\|))$).

Note that for the wide study, FCA-lattices rather than EA-lattices were generated.

Tests were performed on a Pentium 4–2GHz with 1 Gigabyte RAM. Also note that a much faster processor was used than that used in the pilot study.

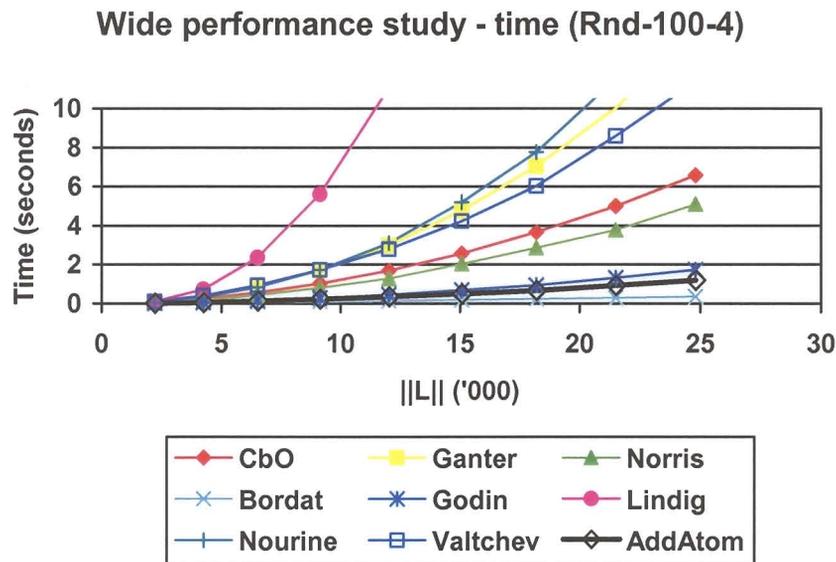


Figure 5.10: Wide performance study test results for the Rnd-100-4 data set plotting the time performance against lattice size ($\|O\|/\|A\| = 4\%$)

Analysis: In random contexts with a very low density (4%) the Bordat algorithm performs the best with AddAtom coming in second.

Wide performance study - time (Rnd-100-25)

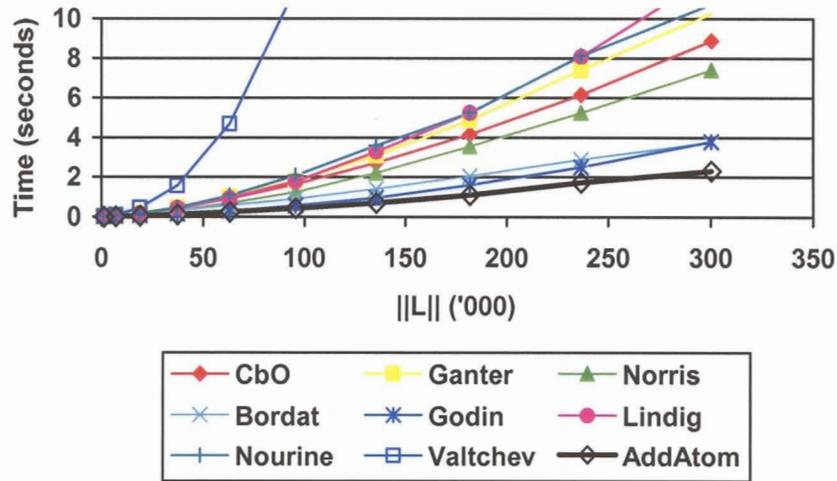


Figure 5.11: Wide performance study test results for the Rnd-100-25 data set plotting the time performance against lattice size ($||O'|||A|| = 25\%$)

Analysis: In random contexts with a relatively modest density (25%) the AddAtom algorithm performs the best with Godin coming second.

Wide performance study - time (Rnd-100-50)

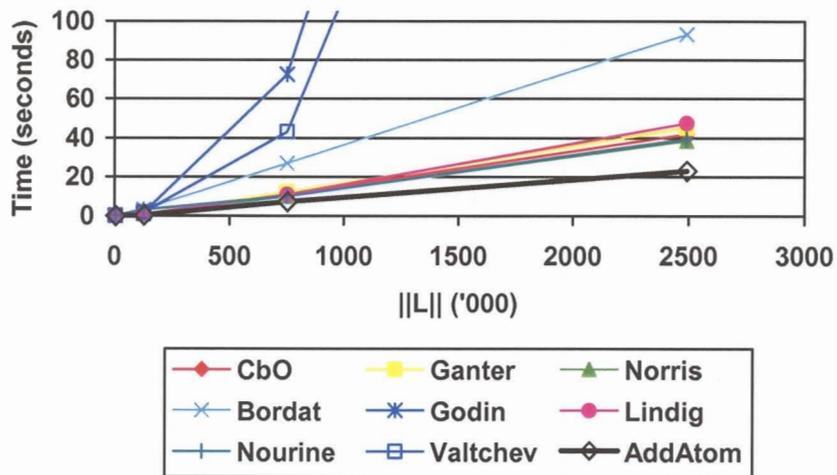


Figure 5.12: Wide performance study test results for the Rnd-100-50 data set plotting the time performance against lattice size ($||O'|||A|| = 50\%$)

Analysis: In random contexts with a relatively high density (50%) AddAtom is still the best performer with Norris second and Nourine a close third.

Wide performance study - time (Bool)

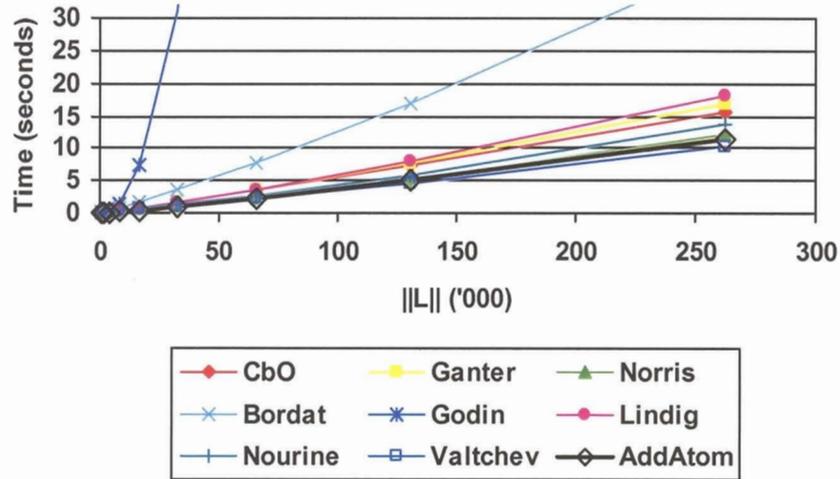


Figure 5.13: Wide performance study test results for the Bool data set plotting the time performance against lattice size ($||O'||/|A|| = 90-94\%$)

Analysis: In artificial contexts that create Boolean lattices (very high density 94%), Valtchev performs the best with AddAtom second and Norris a close third. These contexts can be described as the theoretical limit of random contexts in the sense that the attributes in these contexts are functionally completely independent in that no association rules can be formulated upon any of the attributes of these contexts whereas random contexts of a similar density generated using random functions that are not exceedingly large will always have at least some association rules on some of the attributes.

Wide performance study - time (SPECT)

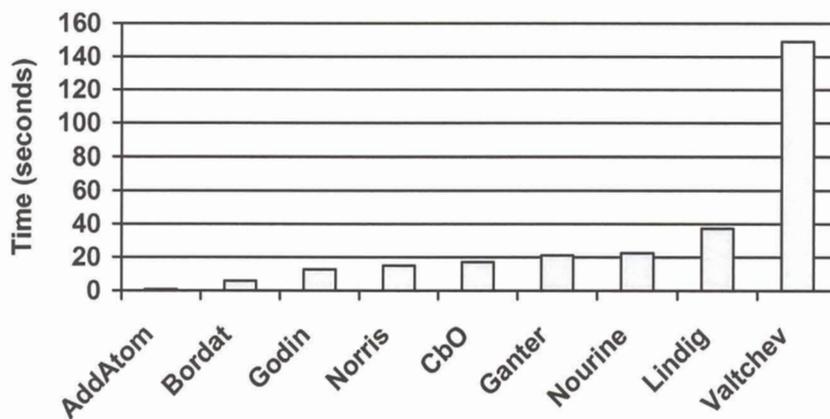


Figure 5.14: Wide performance study test results for the SPECT data set plotting the time performance for different algorithms ($||O'||/|A|| = 33\%$)

Analysis: In a context taken from natural or “real life” data the performance gap between AddAtom and the nearest best performers (in this case Bordat and Godin) even bigger than with artificial contexts. This observation is however based on only one data point.

The table below summarise the performance of the algorithms relative to the best performing algorithm in the largest of each type of context. The letters “B” and “I” are used to indicate which of the algorithms are incremental and batch algorithms.

	Performance index (fastest algorithm = 100%)				
	Rnd-100-4	Rnd-100-25	Rnd-100-50	Bool	SPECT
L	2222	300257	2493490	2493490	21549
Context density	4%	25%	50%	90-94%	33%

CBo (B)	1884%	385%	182%	151%	1973%
Ganter (B)	3960%	445%	196%	164%	2462%
Norris (I)	1461%	320%	166%	119%	1743%
Bordat (B)	100%	162%	404%	374%	684%
Godin (I)	493%	164%	3057%	19431%	1457%
Lindig (B)	24636%	529%	206%	177%	4330%
Nourine (I)	4564%	465%	172%	133%	2606%
Valtchev (-)	3261%	5411%	2326%	100%	17304%
AddAtom (I)	341%	100%	100%	113%	100%

From the above table it is clear that the various algorithms perform differently across the various context densities. Only AddAtom perform very well (albeit not always the best) over the range of context densities. The graph below depicts this graphically but the data points used are only that of the largest contexts. It shows the average time taken per inserted concept for the various artificial contexts and plots it against the density of the context. It clearly shows that both Bordat and Valtchev perform well at opposite ends of the spectrum, but that each (especially Valtchev) performs relatively poorly at the other end. In the graph Boolean contexts are seen as the extreme case of random contexts since there is no implication rule on any of the attributes as would be expected in a infinitely large random context with $||A'|| = ||O'|| = ||A|| - 1 = ||O|| - 1$.

Based on the results and compared to the other algorithms AddAtom is more robust in having less variation in its performance across the range of random contexts in that the standard deviation of the time taken per concept to construct the lattice for these various context were lower. Therefore should a heuristic be used to select the algorithm in a multi-algorithm strategy and the incorrect algorithm be chosen, AddAtom could behave in a more predictable and narrower range of performance that others.

The relative closeness of the AddAtom and Nourine performance confirms that the theoretical performance upper bound of AddAtom as derived is overstated.

Wide performance study - time (largest artificial contexts)

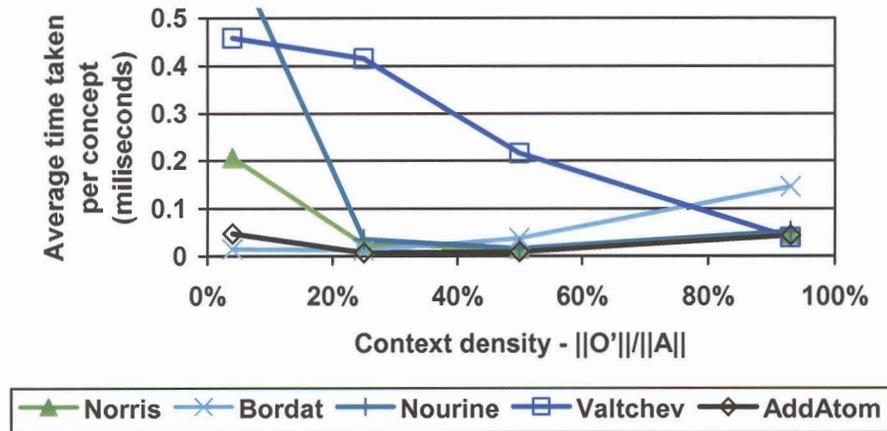


Figure 5.15: Wide performance study test results for the largest of the artificial contexts comparing the time taken per concept to the context density

It should be noted that the graph is constructed on very few data points on contexts of different sizes and the picture it presents is therefore not conclusive. It does however lead to a hypothesis: “for random or artificial contexts, the density of the context is a good predictor of algorithm with the best algorithmic performance”.

When looking at the natural context, there is however a significant difference between the time taken to build the lattice for the SPECT dataset compared to any of the random contexts with similar sized lattices. Once again the data is insufficient to support any conclusions, but the observation lead to a number of hypotheses such as “the algorithmic performance of the various algorithms is significantly different between random and ‘natural’ contexts” and “context density is not a good predictor of algorithmic performance for ‘natural’ contexts”. Investigating these hypotheses is an area for further study and will be useful for a lattice construction strategy of using a combination of algorithms that depend on the context and event the object to be inserted. Such a strategy is proposed by Kuznetsov and Obiedkov (2002).

As was discussed in section 5.2.4, the theoretical complexity bound of AddAtom derived here is not very sharp, especially for non-Boolean lattices and the actual performance of the algorithm may be significantly better than what is suggested by the cubic nature of the theoretical complexity. The experimental results support this claim and also indicate the extent to which the performance is dependent on the nature of the context itself. It is also interesting that the three best performing algorithms in this study namely Bordat, Valtchev and AddAtom are not of the same type. Bordat is a batch algorithm, AddAtom incremental whilst the approach of Valtchev can not be classified as either. AddAtom is however the best performing incremental algorithm for the contexts included in the study.

The good performance of AddAtom can be attributed to the focussed way it traverses the L_{n-1} lattice to find generator concepts. Instead of visiting a substantial number of the concepts in L_{n-1} , it focuses only on concepts that have at least some attributes in common with the object to be inserted. At each generator concept, the search for additional generator concepts higher up in the lattice is more and more focussed resulting in an efficient algorithm.

Lastly, it should be noted that the AddAtom algorithm and not AddCoatom was in the tests. As was indicated in section 5.2, AddCoatom has a smaller theoretical performance

bound. In preliminary tests an approximately 10% performance improvement (in terms of the number of set and lattice operations) in all random contexts was noted on the code used for the pilot performance study when using AddCoatom. Although this might reflect inefficiencies in the implementation; it is likely that some improvement can be gained in certain contexts by using AddAtom in stead of AddCoatom. Other algorithms may also benefit from this approach. This is another area of further study.

5.5 CONCLUSIONS

From the results of the performance tests it is clear that AddAtom is a very good lattice construction algorithm and compares well with other algorithms from a theoretic but especially from an experimental point of view. It does however not perform the best across all types of contexts. In contexts with either very low or very high densities other algorithms (Bordat in low density contexts and Valtchev in high density contexts) perform better than AddAtom. In these contexts AddAtom is still the second best performer. AddAtom is however the best performing incremental lattice construction algorithm. Further study is however required to better quantify the size and nature of the AddAtom performance relative to other algorithms in especially non-random contexts.

Kuznetsov and Obiedkov (2001, 2002) have suggested that the choice of algorithm should be based on the context. The study shows that the context density may be a good predictor for such a test in the case of random contexts but due to the small number of data points as well as the completely different performance on natural data sets it is clear that this is a topics for further study. The density of a context can be pre-computed with relatively little effort before building the concept lattice of a context and could be a very useful tool in selecting an appropriate lattice construction algorithm for building a lattice. This strategy may also benefit from using algorithms adapted to insert attributes (i.e. construct the lattice column by column from the cross table) as well as objects (i.e. constructs the lattice row by row from the cross table) into concept lattices.

On natural / “real world” contexts AddAtom performs particularly well compared to random contexts. Although the performance tests in this study do not fully explore this, AddAtom seems to be a prime contender, for the best performing algorithm for natural data contexts, especially since its performance range was relatively limited compared to other algorithms.