A Lattice-Based Data Structure for Information Retrieval and Machine Learning

FJ (Dean) van der Merwe, DG Kourie

University of Pretoria, Pretoria 0002, South Africa
Dean.vd.Merwe@bentleywest.com; DKourie@cs.up.ac.za

A lattice-based data structure, called a compressed lattice, is proposed as a generic data structure. It is closely related to a concept lattice and can be used in applications such as machine learning, information retrieval and document browsing. The data structure, essentially a bipartite graph with an embedded-lattice, combines desirable features of concept lattices in a structure that allows for a flexible mechanism of scaling the size of the embedded-lattice, by means of defined operations that compress and expand the embedded lattice. A compression strategy or criterion is required to guide this process.

1 Introduction

> *"Knowledge is of two kinds: we know a subject ourselves or we know where we can find information upon it"*
> *Samuel Johnson, quoted in Boswell 1791*

The use and application of concept lattices is an area of active and promising research in various fields of study such as information retrieval [2], software engineering [11], and machine learning [5,6,10].

Here a lattice-based data structure is defined that contains features of both a concept lattice and bipartite graph. The data structure is described in reference to a generic information retrieval problem that is essentially a query operation on a database (section 2). Examples are given of query operations on bipartite graphs and concept lattices. After discussing the merits of both approaches, a compressed lattice is defined as a lattice from which some of the concepts have been removed, subject to a set of so-called compressed lattice properties. An equivalent query on a compressed lattice is also defined. A compressed lattice in essence allows the removal or "compression" of concepts in a concept lattice in such a way that the resulting structure retains the desirable properties of the lattice. It is essentially a bipartite graph with an embedded-lattice. Its properties ensure that removed lattice concepts can be re-instated.

We end by discussing the implementation and use of these ideas as well as promising (albeit preliminary) results of compressed lattices. We argue that a lattice may contain many concepts that are redundant (due, for example, to noisy data) and that these can be removed via the compressed lattice operations. Examples of other approaches to constrain the lattice size are also briefly mentioned. Finally a number of key questions are posed. These are topics for further research.

2 An Information Retrieval (IR) Problem Definition

This paper assumes a working knowledge of concept lattices and readers are referred to [1, 4] for a formal introduction.

Consider a domain of discourse in which each element of a set of *entities*, Ent = $\{e_1, e_2, \ldots, e_j\}$ possesses one or more observable *attributes* from a set of attributes Attr = $\{a_1, a_2, \ldots, a_k\}$. We refer to entities to as *objects*, whilst attributes are sometimes referred to as *features* or *descriptors*. The triple C = $\langle$Ent, Attr, I$\rangle$, where I is a *binary relation* between Ent and Attr, I $\subseteq$ Ent $\times$ Attr, is referred to as a *context* and denotes this domain of discourse. The binary relation I, also called an *incidence relation*, identifies the attributes of each entity.

Consider a set S and arbitrary elements x, y and z in S. A partial ordering relation $\preccurlyeq$ on S is one that is reflexive ($x \preccurlyeq x$), antisymmetric ($x \preccurlyeq y \wedge y \preccurlyeq x \Rightarrow x = y$) and transitive ($x \preccurlyeq y \wedge y \preccurlyeq z \Rightarrow x \preccurlyeq z$). The set S in conjunction with an associated partial ordering relation $\preccurlyeq$ is called a *partially ordered set* or *poset* and is denoted by $\langle S, \preccurlyeq\rangle$. For x, y $\in$ S, $x \neq y$, x is said to *cover* y, indicated by $y \prec x$ when $y \preccurlyeq x$ and there is no $z \in S$, $z \neq x$, $z \neq y$ such that $y \preccurlyeq z$ and $z \preccurlyeq x$. Some texts refer to x as the *parent* or *predecessor* of y. Similarly y is referred to as the *child* or *successor* of x.

One way of visually representing a poset is by means of a directed acyclic graph in which elements of the poset form the nodes and a directed arc is drawn from node y to x iff $y \prec x$. This graph is called a *Hasse diagram* and provides a natural data structure for representing a poset. By convention, instead of showing the direction of arcs explicitly in the Hasse diagram, node x is shown above node y if $y \prec x$.

For this problem domain we define a database D = $\langle S, \preccurlyeq\rangle$ related to context C as consisting of a set, S, of concepts which are partially ordered by the relation $\preccurlyeq$. The database is restricted in that the maximal elements are the attributes (Attr) in C and the minimal elements are the objects (Ent) in C. In addition D may contain any number of such intermediate concepts M (i.e. S = Attr $\cup$ Ent $\cup$ M). The *upward closure* of any concept c, indicated by UpwardClosure(D, c), is the set of concepts greater than or equal to c in terms of the partial ordering. The *downward closure* is the set of concepts that are less than or equal to c, and is indicated by DownwardClosure(D, c). The *extent* of a concept is defined as the set of objects in its downward closure. Similarly the *intent* of a concept is the set of attributes in its upward closure.

We consider the problem of retrieving a set of objects relevant (in some abstract and as yet undefined way) to a specific query. The query is formulated as a set of attributes in the form Q = $\{a_1, a_2, \ldots., a_m\}$ and different query operations can be defined on D. The result of a specific query operation O based on D with respect to query Q is indicated by O(D, Q). A query operation may return any number of concepts from D. For convenience, we place an extra restriction on the query operation: that the operation may return only non-object concepts.

Notwithstanding the foregoing, we choose to interpret the final results in terms of objects, namely those objects that are in the union of the extents of the concepts returned by the query operation (i.e. the union of a number of possibly intersecting attribute downward closures). As a consequence the results of a query can be interpreted as clusters of objects represented by the concepts returned by the query operation.

Different query operations may be evaluated in terms of the information retrieval (IR) metrics, precision and recall. Conversely using the same query operation, different databases of the same context can be evaluated against each other in terms of these metrics. Clearly only concepts in a given database can be returned. Therefore it can be expected that a database containing "meaningful" concepts will return concepts that result in high precision and recall values. We argue that a compressed lattice is a versatile data structure to represent various databases of this type and could prove useful in researching information retrieval and machine learning strategies.

2.1 A Bipartite Database and Query Operation

Attributes	
nw	Needs water
lw	Lives in water
ll	Lives on land
nc	Needs chlorophyl
1lg	1 leaf germination
2lg	2 leaf germination
mo	Is motile
lb	Has limbs
sk	Suckles young

Incidence relation									
	nw	lw	ll	nc	1lg	2lg	mo	lb	sk
LE	✓	✓					✓		
BR	✓	✓					✓	✓	
FR	✓	✓	✓				✓	✓	
DG	✓		✓				✓	✓	✓
SW	✓	✓		✓	✓				
RD	✓	✓	✓	✓	✓				
BN	✓		✓	✓		✓			
MZ	✓		✓	✓	✓				

Entities/objects	
LE	Leech
BR	Bream
FR	Frog
DG	Dog
SW	Spike-weed
RD	Reed
BN	Bean
MZ	Maize

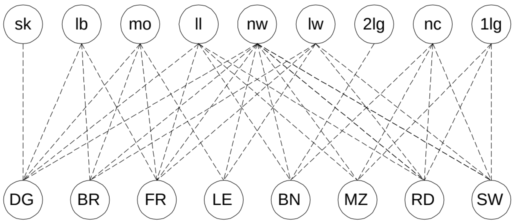

Fig. 1. The living context and its associated bipartite graph

The simplest example of a database is that of a bipartite graph (essentially representing the incidence relation) with objects at the bottom, attributes at the top and arcs from each object to all the attributes it possesses in a specific context as illustrated in figure 1. This simple context, called the living context, is taken from Ganter [2] and was originally used in an Hungarian educational film.

Consider $O_{BP}(D, Q)$, a query operation on a bipartite database. For a query Q we define O_{BP} (D, Q) = Q. As an trivial example we see that for Q = {mo, lw} the query $O_{BP}(D, Q)$ would return {mo, lw}, effectively referencing the set of objects {LE, BR, FR, DG, SW, RD}. This can be verified by inspecting the Hasse diagram of the database for DownwardClosure(D, mo) = {LE, BR, FR, DG} and DownwardClosure(D, lw) = {LE, BR, FR, SW, RD}.

A shortcoming of the bipartite database and of O_{BP} is the fact that the query operation returns a very general set of objects, each of which has any, but not all, of the attributes specified in the query Q. In IR terms the query operation has low precision but high recall. One way of improving the precision is to introduce a new intermediate concept called " mo_lw" that groups all objects that possesses both mo and lw into the database. This concept would be connected via upward-arcs to mo and lw and all objects possessing mo and lw would have upward-arcs to the new concept. In this way, a query operation might be able to use the new concept in arriving at the results of the query.

Continuing this line of thought, the other end of the spectrum would be to use a concept lattice as the database. This option is discussed in the next section.

2.1 A Lattice Database and Query Operation

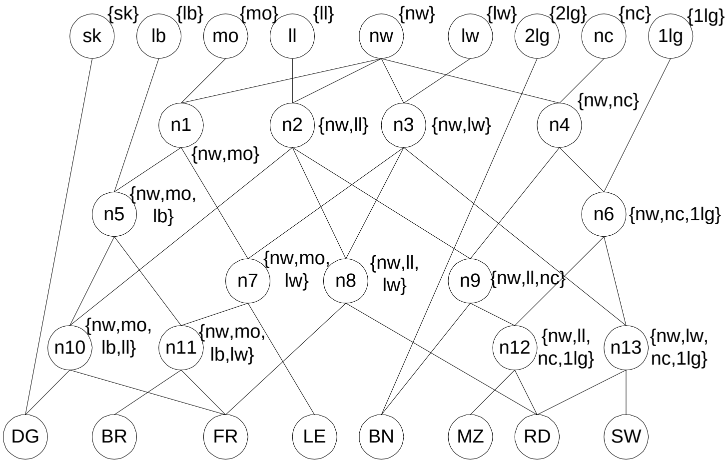

Fig. 2. An EA-lattice of the living context

Figure 2 is a concept lattice for the living context. Note that a modified version of the Galois lattice called an entity-attribute lattice or EA-lattice is used. It has clearly partitioned sets of attributes, objects and intermediate concepts. Consequently it also has a slightly revised ordering relation compared to that of a Galois lattice. The universal- and empty concepts of the lattice have been excluded from the database. (They are, however, implied.) The intents of some of the concepts are shown as a guide.

O_{Meet}(D, Q) is a query operation on a lattice database and is defined as the meet of the attributes of Q in the lattice. For the query Q={mo, lw} the resulting objects are therefore {BR, FR, LE} since the meet of {mo, lw} is concept n7 which has an extent of {BR, FR, LE} (i.e. objects that possesses all of the attributes in Q are returned). Note that it is now possible to obtain a result with a higher precision due to the fact that concepts lower down in the database discern between objects in a more granular way.

2.3 An Adapted Lattice Database and Query Operation

Assuming that we were looking for all the fish objects in the living context (in this case only BR qualifies) with query Q = {mo, lw}. The lattice-based query operation O_{Meet} has a higher precision and recall than O_{BP}. O_{Meet} has however the disadvantage that it is not tolerant of errors or ambiguity in either the context or formulation of the query terms. Assume, for example, that we are looking for all edible plants in the context and used the query Q={nw, nc, 1lg, 2lg}. $O_{Meet}(D, Q)$ would not return any relevant objects since the meet of Q is not in the database – the meet is the empty concept ∅ of the implied lattice. One strategy that could help in this case and increase the tolerance for errors is to specify a query operation O for a context of n attributes that will return the minimal concepts that have at most $k \leq n$ attributes, all of which all are in Q. Since the domain of discourse as defined requires that queries only be formulated in terms of concepts already contained in the database we can adopt one of two strategies. The first is to redefine the query operation to examine all concepts and return the appropriate minimal concepts. In this case, the database D is kept the same. A second option is to modify the query operation and the database. For reasons that will become clear, we will pursue the latter option.

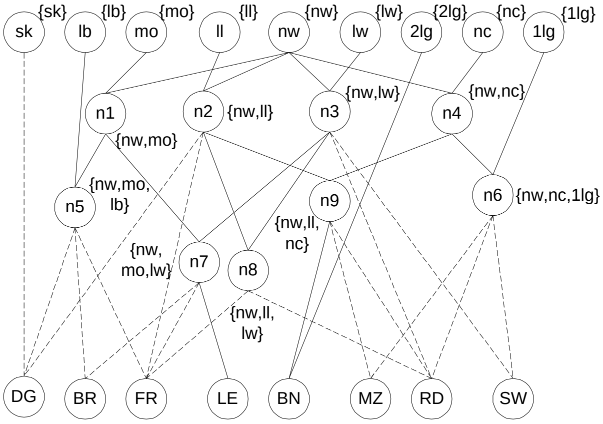

Fig. 3. A database with concepts with an intent of more than three attributes removed

Removing all concepts in the lattice in figure 2 that have more than k = 3 attributes creates the new database in figure 3. Where the original lattice concepts have been removed, dashed-arcs indicate successors defined by the partial ordering relation [1]. Note that a subset, L, of the database namely all the concepts except DG, BR, FR, MZ, RD, SW still forms a lattice when the implied universal and empty concept is inserted. This lattice (identified by all the concepts connected with solid arcs) does not correspond to either the Galois or EA-lattice lattice for the given context. A query

---

[1] Note that, for technical reasons, there is a dashed-arc between DG and sk, even though no concept has been removed. This is done to ensure that, by removing all intermediate concepts, a bipartite graph having only dashed arcs can be derived.

operation on the database in figure 3 using L now cannot discern between objects that have more than k attributes in common.

3 Compressed Lattice Definitions

In this section the database in figure 3 is used as an example of a compressed lattice. We define the set of approximate and exact representatives; the CompressLattice and ExpandLattice operations; and the notion of a compressed lattice. The need for compression strategies for creating a compressed lattice is addressed.

3.1 Approximate and Exact Representatives

Repeating the query operation O_{Meet} for Q={nw, nc, 1lg, 2lg} in figure 3, we find that there is no meet in the database. A revised query operation on the database as defined below will however solve the problem. Consider the lattice, L, and exclude the empty- and the universal concepts of L from the discussion.

Let S be the set of all meets of all subsets of Q in L. The *set of approximate intent representatives* of Q in the lattice L, denoted by AIR(L, Q), is the set of minimal concepts in S[2].

Suppose a query operation for Q returns the approximate intent representatives of Q in the lattice embedded in D, i.e. $O_{AIR}(D, Q) = AIR(D_{Embed}, Q)$ where D_{Embed} is the lattice embedded in D . If Q ={nw, nc, 1lg, 2lg} then S = { nw, 2lg, nc, 1lg, n4, n6, BN} and {BN, n6} is the set of minimal elements of S. Thus O_{AIR} returns AIR(L, Q) = {BN, n6} for figure 3, assuming D_{Embed} is the lattice, L, identified in section 2.3. This refers to the objects {BN, MZ, RD, SW}.

Inspecting the intents of the concepts in AIR(L, Q) we see that BN, for example, has attributes in its intent that are not in Q. If we wish to restrict a query operation to find only concepts possessing attributes in Q, then we need to define a related operation on the database, called the exact intent representative operation.

Let S be the set of all meets of all subsets of Q in L. Let T be that subset of S whose elements have intents that are not subsets of Q. The *set of exact intent representatives* of Q with respect to a lattice L, denoted by EIR(L, Q), is the set of minimal elements in S – T. If T = ∅ then clearly EIR(L, Q) = AIR(L, Q).

For Q = {nw, nc, 1lg, 2lg} we saw that AIR(L, Q) = {n6, BN} whilst EIR(L, Q) = {2lg, n6} since T = {BN}. As before, $O_{EIR}(D, Q) = EIR(D_{Embed}, Q) = EIR(L, Q)$ = {2lg, n6}. Thus, in the present example, O_{EIR} (D, Q) references the same set of objects as before, namely {BN, MZ, RD, SW}.

The O_{EIR} and O_{AIR} operations can be applied to figure 1, in which the embedded lattice is reduced to the set of attributes. In that case, $O_{BP}(D, Q) = O_{EIR}(D, Q) = O_{AIR}(D, Q)$. If D is a pure lattice, as in figure 2, then $O_{Meet}(D, Q) = O_{EIR}(D, Q)$ for a non-trivial meet(Q).

The point is that both the O_{AIR} and O_{EIR} operations are defined in terms of a lattice. and should the lattice be changed as in the example, for figures 1 to 3, the

---

[2] (x ∈ S is minimal, iff ∄y ∈ S such that $y \prec x$.)

representative sets also change. When Q has a non-trivial meet (i.e. not the universal concept) in the lattice then EIR(D, Q) = AIR(D, Q) = Meet(D, Q). The representative sets of Q were defined to deal with situations when Q has a trivial meet. The operations may be seen as extentions of the meet operation.

Dual operations for AIR(D,Q) and EIR(D,Q) can be defined as follows. Q is a set of objects and the meet and the minimal operations can be substituted by join and maximal operations in the above definitions respectively. This defines the set of *approximate extent representatives*, AER(D, Q) and the set of *exact extent representatives*, EER(D, Q). If Join(D, Q) is non-trivial, Join(D, Q) = AER(D, Q) = EER(D, Q).

Finally, it is useful to define a further related set of concepts, namely EIR(D,Q,C). This is the set of exact intent representatives of Q *excluding* C. It corresponds identically to EIR(D,Q), except that in determining the minimal elements of S above, a designated concept, C, is specifically excluded from consideration. As a result, if C is in EIR(D,Q), then EIR(D,Q,C) contains all the concepts that cover C, instead of C itself. In particular, if C = meet(D,Q), then EIR(D,Q,C) is the set of concepts covering meet(D,Q). The set of *exact extent representatives excluding* C, EER(D, Q, C) is defined similarly.

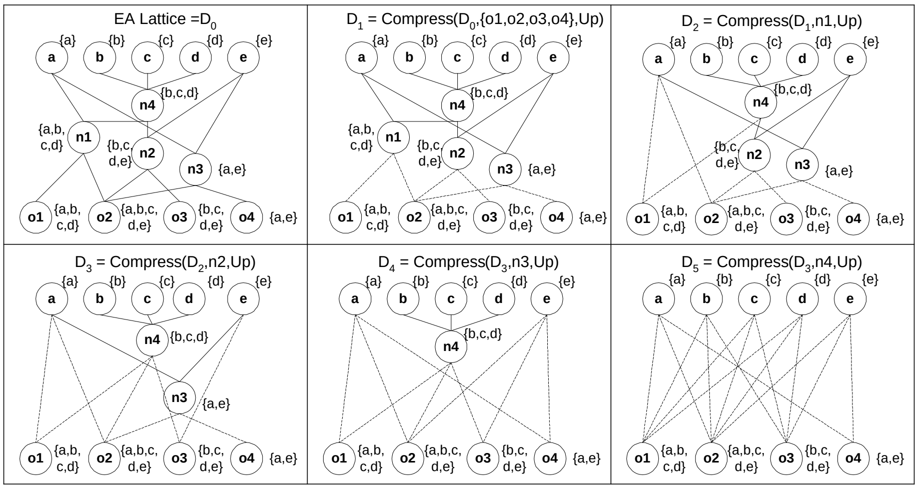

Fig. 4. A CompressLattice example compressing a lattice to a bipartite graph

3.2 CompressLattice Operation

The CompressLattice operation removes a concept y from the embedded-lattice in the lattice-based data structure and replaces the concept with virtual-arcs (indicated as a dashed-arcs). These arcs interconnect all the parent- with all the child concepts of y. Figure 4 shows an example of a lattice-based structure where all the concepts have been removed by successively using CompressLattice operations. Similarly, figure 3 can be verified to be the result of successive upward CompressLattice operations on DG, BR, FR, MZ, RD, SW, n10, n11, n12, and finally n13.

It is important to note that the CompressLattice operation works from a particular direction. In the examples, the lattice was compressed in the upward direction, but the operation is equally valid when compressing the lattice from the top downward (or any combination of the two). The CompressLattice operation creates a data structure that is not a lattice. We call it a compressed lattice. Using parameter names to imply types, the CompressLattice operation is defined as follows in terms of its pre- and post-conditions:

> **`CompressLattice(aCompressedLattice, aConcept, aDirection) returns outCompressedLattice`**
>
> **`Pre-condition:`** `aConcept is in aCompressedLattice, it has at least one lattice-arc in aDirection and no lattice-arcs in the opposite direction.`
>
> **`Post-condition:`** `outCompressedlattice retains all the nodes and arcs of aCompressedLattice, except in the following respects. If aConcept is an attribute or entity concept, then lattice-arcs connecting it to other concepts in aCompressedLattice are replaced by virtual-arcs in outCompressedLattice. Otherwise aConcept and its arcs are not in outCompressedLattice. Instead, virtual-arcs link each of aConcept's parents to each of aConcept's children.`

3.3 Definition and Properties of Compressed Lattices

A *compressed lattice* is a data structure that represents a particular context C = ⟨Ent, Attr, I⟩. The data structure consists of a number of concepts that are connected by one of two types of directed arcs: *lattice-arcs* and *virtual-arcs*. The concepts are partitioned into three sets: the attributes (of the context), the objects (of the context) and any number of intermediate concepts. A compressed lattice contains an *embedded-lattice.* The embedded-lattice is the set of all concepts complying with one of the following: the concept is an attribute node; or at least one lattice-arc connects to or from the concept. Note that in an embedded lattice, lattice- and/or virtual arcs may be incident on an attribute node.

The following are compressed lattice properties. They define sufficient conditions for a data structure to be a valid compressed lattice. Note that the conditions listed are not disjoint – they may be related to or imply one another.

- *Poset***:** The concepts in the compressed lattice form a poset with respect to the partial ordering relation specified by the directed arcs (lattice or virtual)
- *Context preservation*: Attributes and objects in the context are present as concepts. Objects contain in their upward closure all their attributes specified in the context,

but no other attributes. Similarly attributes contain in their downward closure all their objects specified in the context, but no other objects.
- *Unconnected objects and attributes*: Attributes may have no upward arc and similarly objects may have no downward arcs.
- *Unique intermediate concepts*[3]: No two intermediate concepts may have the same extent or the same intent. This property implies that any intermediate concept has at least two upward and two downward arcs. This does not preclude attributes and objects from having the same extent or intent respectively. Such concepts are represented as different concepts in a compressed lattice.
- *Empty intent*: No concept may have an empty intent (i.e. all objects must possess at least one attribute but some attributes may not have any object possessing the attribute). This limits the contexts for which a valid compressed lattice may be constructed. Although the property is not strictly required, the practical benefits of contexts that do not conform to this requirement are not immediately clear.
- *Embedded-lattice*: The set of all concepts together with the ordering used in the embedded-lattice, constitute a lattice when appropriately connected to the implied universal and the empty lattice concepts.
- *Meet and join*: Consider any set, S, of concepts connected via lattice-arcs. Any subset of S has at most one meet or no meet at all (the lattice property). Similarly any subset of S has at most one join or no join at all.
- *Intermediate virtual-arcs*: Intermediate concepts may not have any virtual-arcs to any other intermediate concepts. Their virtual-arcs must end in an attribute concept or start at an entity concept.
- *Exact representative connection*: Let I(C) be the intent of any concept C in a compressed lattice. Let S_{Embed} and S_{Full} be the sets of exact intent representatives of I(C) excluding C in the embedded lattice and fully elaborated EA-lattice, respectively. Then a lattice arc connects C to each element of S_{Embed} if the element is also in S_{Full} and a virtual arc connects C to every other element of S_{Embed}. A similar property holds for the set of exact extent representatives of E(C) excluding C, where E(C) is the extent of C. These dual properties are critical in ensuring that the closure operations function as expected.
- *Arc duplication*: A concept may only have one arc (either lattice or virtual) to any concept that covers it.
- *Cover*: A concept may not have an arc to any other concept to which it is indirectly linked[4].

The definition and properties show that a compressed lattice is essentially a bipartite graph that contains an embedded-lattice. Furthermore the compressed lattice properties ensure a well-defined and unique structure for a given context and a given sequence of compressed lattice operations. A number of operations can be defined on a compressed lattice, but the most important are:
- ApproximateRepresentatives and ExactRepresentatives for an intent and extent
- CompressLattice and ExpandLattice (described in the next section)

---

[3] This is not a property of Galois lattices. For example, the Galois lattice of the living context (not shown here) does not possess this property.

[4] Concept x is indirectly linked to concept y iff it has a path via one or more intermediate concepts.

- Closure and LatticeClosure, where LatticeClosure follows only lattice-arcs when discovering concepts whilst Closure follows any type of arc
- InsertNewLatticeObject, i.e. insert a new object into the context and embedded-lattice by using a modified incremental lattice construction algorithm
- InsertNewVirtualObject, an alternative to InsertNewLatticeObject that does not use a computationally expensive lattice construction algorithm to update the embedded-lattice. The object is inserted into the compressed lattice by simply creating virtual-arcs to its exact representatives

3.4 ExpandLattice Operation

A complementary operation to CompressLattice, namely ExpandLattice, can be defined to enlarge the embedded lattice of a compressed lattice. The operation works in a particular direction, starting with a concept that has virtual arcs in that direction.

In the downward direction starting with concept C, the operation determines what the children of C would be in the fully elaborated ("uncompressed") EA-lattice. To do this requires the determination of the set of exact extent representatives excluding C, of the extent of C. Concepts in this set but not yet in the compressed lattice are inserted into it. C is directly connected to these concepts by lattice-arcs and C's virtual arcs are removed. To comply with compressed lattice and "pure" lattice properties[5] further generation of concepts and/or creation, removal or re-labelling of arcs may be necessary. Similar remarks apply *pari passu* when expanding a given concept in the upward direction.

Note that the CompressLattice and ExpandLattice operations *are not symmetric* in that the one does not reverse the other. In most instances a single CompressLattice operation cannot destroyed the portion of the compressed lattice that an ExpandLattice operation builds. It is however always possible to completely compress a lattice into a bipartite graph or to use ExpandLattice operations to completely rebuild the lattice from a bipartite graph. Our implementation of this latter series of operations indicates that it is computationally more expensive than using a "traditional" incremental lattice construction algorithm to construct a lattice. The context preservation and exact representative connection properties of a compressed lattice play important roles in the ability to rebuild the lattice.

ExpandLattice is defined below in terms of its pre- and post conditions. Again, parameter names imply their corresponding types.

`ExpandLattice(aCompressedLattice, aConcept, aDirection) returns outCompressedLattice`

`Pre-condition:` `aConcept is a concept in aCompressedLattice that has virtual-arcs in aDirection.`

---

[5] The properties labeled above as **Embedded-lattice** and **Meet and join** embody the lattice properties to be retained by a compressed lattice.

Post-condition: outCompressedlattice retains all the nodes and arcs of aCompressedLattice, except in the following respects. If aDirection is down (up), then all possible child (parent) concepts of aConcept that would occur in the associated EA lattice are inserted into outCompressedLattice's embedded lattice. Additional concepts are created and arcs are created, removed or relabelled if and only if necessary to maintain compressed lattice (including embedded-lattice) properties.

As an example of the ExpandLattice operation, consider figure 4 with the compressed lattices D_0 to D_5. When starting with D_5, i.e. the bipartite graph, the following order of ExpandLattice operations will reconstruct D_0 (using ↓ to indicate "downward"): D_4 = ExpandLattice(D_5, c, ↓); D_2 = ExpandLattice(D_4, e, ↓); D_1 = ExpandLattice(D_2, a, ↓) and finally D_0 is the result of successive ExpandLattice calls that expand the concepts n1, n2 and n3 in a downward direction. Note that ExpandLattice(D_4, e, ↓) does not produce D_3 because D_3 does not contain *all* of e's children that exist in the underlying and implied full lattice, D_0.

3.5 Pruning Strategies and Criteria for Creating Compressed Lattices

Section 2 defined a very specific domain of discourse of which there are three components: the context, the database and the query operation. A compressed lattice is such a database. The separation of the database and query operation as well as the requirement that results only be formulated in terms of non-object concepts actually represented in the database creates an interesting deviation from some traditional information retrieval approaches: the organization of the database co-determines the outcome of the of the query operation in that for a given context the result of a query may be different, depending on the compressed lattice being used to represent the context. The question that arises is thus: "Are there databases derived from compressed lattices that, on average, result in better retrieval for the same context?"

Limited experimental results (currently unpublished) show that there are indeed better methods of organization. Specifically, it appears that a database consisting of the complete lattice of a given context need not, in general, be the best database. In many instances significantly compressed lattices performed better. Further experimentation is required in order to explore compression strategies and node pruning criteria that lead to optimal performance.

In general, there are a number of possible compression strategies that seem to deserve such exploration. The most obvious is the one stated above where the lattice is compressed up to a specific level. It is useful to have a pruning strategy combined with a threshold on the embedded-lattice size. The embedded-lattice is then repeatedly compressed until the lattice size is below the threshold. This can be combined with an adapted incremental lattice construction algorithm where the pruning mechanism is invoked after each individual object or batch of objects has been inserted into the compressed lattice. Combinations of the operation

InsertNewVirtualObject can also be used. This has the added advantage of limiting the size of the lattice and therefore the time taken to build a compressed lattice.

Compression strategies that have been preliminarily tested use a combination of the following:

- Compress concepts with an intent of size smaller than t and larger than u.
- Compression is based on the number of arcs to child or parent concepts in the lattice.
- Compression is based on EP(c), an estimate of prior probability of the concept c. EP(c) is the number of objects in the intent of c divided by the total number of concepts in the context. Refer to Oosthuizen [10] for a discussion and examples
- Compress, based on the difference between an estimate of the expected probability ExP(c)[6] and EP(c). This concept property performed the best in most preliminary test results.

4 Implementation and Discussion of Preliminary Results

The compressed lattice data structure and associated operations have been implemented and tested in C++. To construct the lattice a new incremental lattice construction algorithm was developed. This algorithm is currently being compared to other documented incremental construction algorithms such as Godin [5] and Oosthuizen [8]. Early indications are promising. The algorithm is explicitly relies upon exact representative and approximate representative sets. (The algorithm's main loop has an invariant and terminating condition that is based on these.) It has been extended to operate on compressed lattices by incrementally inserting new objects as well as the required new intermediate concepts into an embedded-lattice whilst maintaining the compressed lattice properties.

The potential gain in computational efficiency of having a compressed lattice should be weighed against the advantages of having a larger and more complete set of concepts available in a particular domain. Results however suggest that a compressed lattice may be a useful generic data structure for various IR and machine learning problem domains.

In Oosthuizen [9, 10] and Kourie and Oosthuizen [7], a data structure based on a modified lattice was proposed as a means to reduce the number of concepts in a lattice. That data structure (also called a compressed lattice) was a limited version of the compressed lattice data structure defined above. It did not retain the lattice properties and other desirable properties (e.g. context preservation) of the data structure proposed above. Despite those limitations, improved results in lattice-based machine learning tests were achieved. It was argued that although a structure such as a lattice contains a concept node for every possible combination of objects supported by the data, it seemed to contain many concepts that are not, in some sense, useful or meaningful. Removing them appeared to improve the outcome.

---

[6] $ExP(c) = EP(a_1) \times EP(a_2) \times \ldots \times EP(a_n)$ and where a_i is an attribute in the intent of c. This estimate assumes that the attributes are independent.

One application of this idea was to remove concepts in the embedded-lattice that are the meets of statistically independent attributes. Being randomly related, these attributes will to occur in any number of combinations in a given context. As a result, a large number of concepts are generated in a lattice to reflect these random relationships. Indeed, the theoretical limit of the lattice size is for example reached when all attributes in the context are statistically independent [10]. Such concepts are not really worth learning as rules in a machine learning context and are therefore in some sense not “meaningful”.

The compressed lattice we described here is a more generalised and versatile version of this approach. Alternative methods of reducing concepts are also discussed in [10]. Godin [5] on the other hand also proposed ways of reducing concepts in a lattice called a pruned concept lattice. In general a compressed lattice is not directly comparable to a pruned concept lattice but the two concepts can be combined.

Finally, the scaling of the size of a compressed lattice has an interesting implication in IR contexts. In a lattice the results of a query operation $O_{EIR}(D, Q)$ is the intersection or conjunction of the downward closures of the attributes in Q (i.e. the meet) and can be expressed as in equation 1. In the bipartite graph $O_{EIR}(D, Q)$ is the union of the downward closures of the attributes in Q expressed as in equation 3. By iteratively scaling the lattice the conjunctive expression progressively turns into a disjunctive expression with an intermediate expressions typified by equation 2. For a suitable compression strategy, a proximity based query operation can thus be defined.

$$O_{EIR}(D, Q) = a_1 \wedge a_2 \wedge \ldots \wedge a_n \text{ for } a_i \in Q\text{, provided Meet(Q) is non-trivial} \qquad \textbf{(1)}$$

$$O_{EIR}(D, Q) = (a_{1,1} \wedge a_{1,2} \wedge \ldots a_{1,j}) \vee (a_{2,1} \wedge a_{2,2} \wedge \ldots \wedge a_{2,k}) \vee \ldots \vee (a_{n,1} \wedge a_{n,2} \wedge \ldots \wedge a_{n,x}) \text{ for } a_{y,z} \in Q \qquad \textbf{(2)}$$

$$O_{EIR}(D, Q) = a_1 \vee a_2 \vee \ldots \vee a_n \text{ for } a_i \in Q \qquad \textbf{(3)}$$

5 Conclusion and Areas of Further Research

We defined a compressed lattice as a generic lattice-based data structure that shows promise in many field of research due to its close resemblance to that of a lattice. A number of critical questions remain:

- In what areas of application is a compressed lattice beneficial? Specifically: is a compressed lattice more suitable than a Galois lattice in areas where the latter has proven successful?
- What compression strategies and criteria should be used and in which areas of application? Specifically, is there a universal compression strategy applicable to many areas of application or is a compression strategy domain specific?
- What is the relation of a compressed lattice and associated operations to other fields of research in databases, rough sets, etc given its seeming ability to deal with ambiguity?

The authors intend pursuing these and related areas of research. Initial results indicate that the answers to these and other questions hold promise in many applications. In addition, the previously mentioned lattice construction algorithm is being evaluated against documented incremental lattice construction algorithms.

Acknowledgement

The authors wish to acknowledge the invaluable contribution that Deon Oosthuizen made to this research during his tenure as an associate professor at Pretoria University.

References

1. B. Birkhoff. *Lattice Theory*, volume 25. American Mathematical Society Colloquium Publ., Providence, revised edition, 1973.
2. B. Ganter, K. Rindfrey, and M. Skorsky. Software for formal concept analysis. In *Classification as a tool of research,* Elsevier Science, 1986.
3. C. Carpineto and G. Romano. Information retrieval through hybrid navigation of lattice representations. *International Journal of Human-Computer Studies* 45, pp 553-578. 1996.
4. B. Ganter and R. Wille. Formal Concept Analysis, *Mathematical Foundations*. Springer-Verlag, 1999.
5. R. Godin and R. Missaoui. An Incremental Concept Formation Approach for Learning from Databases. *Theoretical Computer Science, Special Issue on Formal Methods in Databases and Software Engineering*, 133, 387-419. 1994.
6. R. Godin, G. W. Mineau, and R. Missaoui. Incremental structuring of knowledge bases. In *Proceedings of the first International Symposium on Knowledge Retrieval, Use and Storage for Efficiency* (KRUSE'95), Santa Cruz, CA, USA, pages 179-198. 1995.
7. DG. Kourie, GD. Oosthuizen. Lattices in machine learning: complexity issues. *Acta Informatica* 35, 269-292. 1998.
8. GD. Oosthuizen. Lattice-based Knowledge Discovery. In *Proceedings of AAAI-91 Knowledge Discovery in Databases Workshop*, Anaheim pp221-235, 1991.
9. GD. Oosthuizen. A Dynamic Indexing Mechanism for Memory-based Reasoning. *Proceedings of the international AMSE conference on "intelligent systems"* , SMSE Press pp127-136, 1994.
10. GD. Oosthuizen. The application of concept lattices to machine learning. *Technical Repor*t CSTR 94/01 Department of Computer Science University of Pretoria, 1994.
11. G. Snelting and F. Tip. Reengineering class hierarchies using concept analysis. In *Proceedings of ACM SIGPLAN/SIGSOFT Symposium on Foundations of Software Engineering*, Orlando, FL, pages 99-110. 1998.