Chapter 3: Lattice construction

This chapter discuss the considerations when algorithmically constructing a concept lattice or rather the line diagram of the concept lattice. This is done through the formulation of an ineffective lattice construction algorithm.

3.1 ALGORITHMIC LATTICE CONSTRUCTION

Lattice construction algorithms use A, O and I of a context $C = \langle O, A, I \rangle$ as input (or a labelled version of cross table as shown in the incidence relation of the Living Context in figure 2.2 in chapter 2). All the concepts are discovered and connecting arcs representing the cover relationship are constructed between appropriate pairs of concepts. The basic output of such an algorithm is thus a set containing the concepts of the lattice as well as a set of arcs connecting these concepts.

3.2 INCREMENTAL VS. BATCH LATTICE CONSTRUCTION ALGORITHMNS

There are two basic strategies for algorithmically constructing a concept lattice form I. The first is to consider the whole context and construct the lattice in a non-incremental or batch way (i.e. if more objects are added to the context, the whole lattice must be reconstructed from the start). (Bordat (1986), Chein (1969), Ganter (1984), Kuznetsov (1993), Lindig (1999, 2000), Zabezhailo et al. (1987)) have followed this strategy. The second strategy is to incrementally build the lattice, adding objects to the lattice until the lattice of the whole context is constructed. In each invocation of an incremental algorithm an existing lattice L_i is used as input. The new object is added to the lattice along with any new concepts and attributes that may be required to create a new lattice L_{i+1}. Therefore unlike the non-incremental strategy, a valid lattice exists after each iteration of the algorithm. The incremental algorithm will also modify the arcs of L_i to create L_{i+1}. Godin (1991), Carpineto and Romano (1993, 1996b), Oosthuizen (1991), Dowling (1993) and Norris (1978) have followed an incremental lattice construction strategy (refer to section 5.1 for references to more algorithms).

The main advantage of incremental lattice construction algorithms is that new objects can efficiently be added to the lattice without rebuilding the whole structure and therefore the incremental construction algorithms are often used. This may however be at some expense since the algorithm cannot optimise across all objects in the context.

3.3 CONSTRUCTING THE LINE DIAGRAM

A number of published non-incremental lattice construction algorithms only generate the set of concepts of the lattice and do not generate the line- or Hasse diagram that represents the cover relationships between lattice elements. Although there are applications such as the generation of all implication rules of a context in which only the set of concepts is used, the majority of lattice-based applications do explicitly use the ordering of concepts in terms of generalisation and specialisation. This ordering is after all one of the primary benefits of lattice-based applications.

For the purposes of comparing the algorithmic performance of various lattice construction algorithms a common framework is required and therefore it is argued that only construction algorithms that do produce the line diagram of the lattice should be considered since they have a more general application. Kuznetsov and Obiedkov (2002) have adapted a number of the non-incremental algorithms that do not generate the line diagram to generate it.

3.4 AN INEFFICIENT BATCH LATTICE CONSTRUCTION ALGORITHM

A simple way to demonstrate the basic considerations and challenges in concept lattice construction algorithms is to consider a very inefficient construction algorithm which uses the basic definition of an EA-lattice to construct the lattice data structure. (Note that the algorithm can be easily adapted to formal concept lattices by changing the conditions for testing.)

The BruteForceEAConstruct algorithm below, "blindly" applies the definition of the EA-lattice in a very inefficient way. From the definition of EA-concepts it is clear that all the concepts of an EA-lattice can be discovered by enumerating and inspecting all possible combinations of $E \subseteq O$ and $F \subseteq A$ (i.e. $\mathcal{P}(O) \times \mathcal{P}(A)$) and then inspecting each couple $\langle E, F\rangle$ to test whether it is EA-formal. Once all the lattice concepts have been discovered, the definition of the cover relationship between concepts is used to test each pair of concepts to determine whether they cover each other. As one might expect this algorithm is very inefficient since it inspects all possible combinations of attributes and objects without having any strategy to prune the search space.

```
//=================================================================
Function BruteForceEAConstruct (anObjSet, anAttrSet,
                anIncidenceRelation) Return aLattice
//=================================================================
CreateNewLattice(L)
L.Concepts = ∅
For ∀ E ∈ 𝒫(anObjSet)
   For ∀ F ∈ 𝒫(anAttrSet)
      // Determine E'
      E' = ∅
      For ∀ o ∈ E
         For ∀ a ∈ anAttrSet
            If oIa with I = anIncidenceRelation then E' = E' ∪ {a}
         Rof
      Rof
      // Determine F'
      F' = ∅
      For ∀ a ∈ F
         For ∀ o ∈ anEntSet
            If oIa with I = anIncidenceRelation then F' = F' ∪ {o}
         Rof
      Rof
      // Test against EA-formal concept definition
      If ||E|| = 1 and F=O' then L.Concepts = L.Concepts ∪ {⟨E, F⟩}
      If ||F|| = 1 and E=A' then L.Concepts = L.Concepts ∪ {⟨E, F⟩}
      If E = ∅ and F = anAttrSet then L.Concepts =L.Concepts ∪ {⟨E, F⟩}
      If F = ∅ and E = anObjSet then L.Concepts = L.Concepts ∪ {⟨E, F⟩}
      If F'= E and E'= F then L.Concepts = L.Concepts ∪ {⟨E, F⟩}
   Rof
Rof
// Discover the cover relationships
L.Cover = ∅    // L.Cover is a set of concepts in the form ⟨b, c⟩ this
               // assumes a unique symbol is associated with each
For ∀ x ∈ L.Concepts // x and y are concepts in the form ⟨E, F⟩
   For ∀ x ∈ L.Concepts, x ≠ y, $x \leq_{EA} y$
      CoverFlag = True
      For ∀ z ∈ L.Concepts, z ≠ x, z ≠ y
         If $x \leq_{EA} z \leq_{EA} y$ then CoverFlag = False
      Rof
      If CoverFlag then L.Cover = L.Cover ∪ {⟨x, y⟩}
   Rof
Rof
Retrun L
End BruteForceEAConstruct
//=================================================================
```

Because of the non-specific and unfocussed way in which the algorithm creates concepts and arcs it soon becomes hopelessly inefficient for all but the smallest of contexts.

This algorithm does however address the basic functions of a concept lattice construction algorithm:

- Generating all the concepts of the lattice and representing them in a data structure.

- Discovering the cover relationship between the concepts and representing it in a data structure.

There is however a number of other issues that need to be addressed in order to be more efficient:

- Avoiding the generation of duplicate concepts or at least determining whether a concept is generated for the first time.
- Efficiently testing and/or generating the cover relationships.
- Efficiently searching for concepts in the set of concepts that has been generated up to that point.
- Avoiding the generation of cover relationships that might be deleted later in the algorithm.

One important property of the algorithm is that it is non-incremental in that we cannot incrementally construct the lattice by starting with a lattice and adding a new object to create a new lattice – the whole lattice needs to be constructed anew. With an incremental algorithm we can create a lattice L_{n+1} with $n + 1$ objects by taking the lattice of n objects, L_n and add the $n + 1$'th object. The key to the incremental algorithm is the observation that the generated lattice (L_{n+1}) always contains all the concepts of the original lattice (L_n) but concepts were either added, modified or left unchanged. Such an algorithm will discover and new concepts and arcs needed to create L_{n+1} as well as deleting arcs where the newly created concepts redefine the cover relationship between specific concepts. This is the strategy followed by the AddAtom algorithm defined in the next chapter.

Another important property of the algorithm is that it constructs the set of all concepts of the lattice as well as the line diagram of the lattice. In applications such as those using association rules where only the set of concepts are required, the last part of the algorithm can be skipped. Section 5.1 lists a number of published lattice construction algorithms and compare them with regards to their key characteristics.

Although this algorithm can be improved upon in a number of ways, we will not pursue this further but will instead define a new algorithm using more elegant strategy in the next chapter.