Université de Montréal

Association Rule Mining for Query Expansion in Textual Information Retrieval

par

Jin Zuo

Département d'informatique et de recherche opérationnelle

Faculté des arts et des sciences

Mémoire présenté à la Faculté des Études Supérieures
en vue de l'obtention du grade de
maîtrise ès sciences M.Sc.
en Informatique

Août 2004

Université de Montréal

Faculté des Études Supérieures

Ce mémoire intitulé :

Association Rule Mining for Query Expansion

in Textual Information Retrieval

présenté par:

Jin Zuo

a été évalué par un jury composé des personnes suivantes :

Miklós Csürös

Président Rapporteur

Petko Valtchev

Jian-Yun Nie

Directeur de recherche

Guy Lapalme

Membre du Jury

Mémoire accepté : 25 février 2005

Résumé

La recherche documentaire moderne est une discipline en expansion. Un système de recherche documentaire identifie automatiquement les documents pertinents dans de grandes collections selon les besoins d'information de l'utilisateur. Cependant, l'utilisateur peut ne fournir que des questions imprécises. Ainsi, il va recevoir en réponse un très grand ensemble de documents. Pour réduire le nombre de documents inappropriés retrouvés par un système, plusieurs méthodes ont été proposées pour d'améliorer la requête, en y ajoutant des mots-clés additionnels pour augmenter la spécificité de la requête.

Des règles d'association ont été présentées comme approche pour des dépendances entre les éléments dans des bases de données de transactions commerciales. Elles représentent un cadre particulièrement approprié pour l'extraction des dépendances entre les mots-clés d'un corpus textuel. De leur côté, les treillis de Galois fournissent une base théorique pour l'extraction de règles d'association à partir d'un ensemble de données.

Dans ce mémoire nous étudions une approche par treillis de Galois pour l'extraction de règles entre termes en vue d'une expansion de requête. Nous développons des méthodes d'extraction de règles et d'expansion de requête. Un problème important ici est d'affecter des poids aux nouveaux mots-clés qui seront ajoutés à la requête. Nous avons effectué diverses expérimentations pour trouver des stratégies appropriées pour ceci. Selon nos expérimentations, l'application des règles d'association dans l'expansion de requêtes apporte quelques améliorations, mais nombreux problèmes restent encore à résoudre avant que les règle d'association puissent être utilisées pour améliorer la performance de la recherche d'information.

Mots clés : règles d'association, treillis de Galois, treillis iceberg, l'expansion de requête, recherche d'information

Abstract

Modern information retrieval is a rapidly expanding discipline. Using algorithmically derived search engines, information retrieval systems automatically identify and select documents from large document collections according to the user information need. However, the user may only be able to provide simple and inaccurate queries and thus get in return a very large set of documents. To reduce the number of irrelevant documents retrieved by a system, different strategies were proposed to improve the query, in particular, by joining in additional keywords to increase the specificity of the query.

Association rules have been introduced as an approach to the discovery of dependencies between items in transaction databases. In fact, they represent a particularly suitable framework for keyword dependency mining from a textual corpus. Galois lattices provide, in turn, a theoretical basis for the extraction of association rules from a dataset.

In this thesis, we study a Galois lattice-based approach for rule mining for query expansion and we design methods to implement it. One important problem proved to be the assignment of weights to the new keywords that will be added to a query. We did many experiments to find suitable strategies for weight computation. According to the experimental results, the application of association rules to query expansion can slightly improve the system performance. However, there are still important problems to be solved before the association rules could be used effectively in IR practice.

Key words:

Association rule mining, query expansion, information retrieval, Galois lattice, iceberg lattice.

List of Tables

List of Figures

Acknowledgments

Many thanks must go to my directors, Professor Petko Valtchev and Professor Jian yun Nie for their knowledgeable contribution and guidance throughout the research. It is them who patiently led me into this area.

I would like to thank my parents, my parents-in-law and my wife for their encouragement, belief and support in me whenever I encountered difficulty. Special thanks go to my daughter. She brings my family much joy.

I would also like to extend gratitude to my classmates and friends, without them, my work could not have been finished so soon.

Chapter 1 Introduction

Modern information retrieval is a rapidly expanding discipline, driven by the ever growing amount of available textual documents, in particular on the Web, but also in electronic libraries and documentary data bases of the large corporations. Using algorithmically derived search engines, information retrieval systems or search engines try to automatically identify and select the relevant documents from large document collections according to the user information need. A search is successful if it returns as many relevant documents as possible and as few non-relevant ones as possible. For most of the search engines at present, users have to submit a query to search documents. In these cases, the user information need is represented by a query in a search engine. Unfortunately the users are not always capable of formulating well-defined queries to represent their information need. They may only be able to provide simple and inaccurate queries and thus get in return a very large set of documents. The effectiveness of the information retrieval system will therefore be very poor. Improving the user's queries becomes a very important task of the IR system. For that purpose, the correlations between keywords can be explored. The resulting approach consists in expanding initial user queries, by joining in additional keywords to increase the specificity or the coverage of the query. By automatically expanding queries one tries to provide additional information so as to obtain a more accurately formulated queries.

Our work described in this thesis aims to expand queries automatically. Our query expansion strategy is based on the term co-occurrences. Various methods exploiting term co-occurrences for query expansion have been proposed. In these methods the relationships between terms are bi-directional. That is, if there is a relation between "database" and "data structure", then the reverse relation is also assumed. From a query expansion perspective, if it is reasonable to expand a query on "database" by adding

"data structure", it may not be the case in the reverse direction. In our application the relations between terms are asymmetrical or directional. Some terms are premise and others are treated as consequence. This is equivalent to considering a relationship between terms as a logical implication. We think that if we can find meaningful rules in our application then we can get a more relevant result by applying them in the process of query expansion. The interpretation of the rules may make the process more reasonable. If it is the case then the technique can also be used to interact with users to get better results.

The relations between terms are expressed as association rules in our study. Association rules have been introduced as an approach to the discovery of dependencies between items in transaction databases. We believe that they represent a particularly suitable framework for keyword dependency mining from a textual corpus. However, a well known problem with association rule miners is the large number of discovered rules, which can contain much noise. Galois lattices provide a theoretical basis for mining association rules which helps reduce the number of rules that can be extracted from a dataset, while preserving the global amount of knowledge gained. Furthermore, non-redundant bases can be constituted as a minimal representation of the complete set of association rules.

Our aim is to study the Galois lattice-based approach to extract important association rules, and to use them in information retrieval. More precisely, the goal is to design effective methods which implement the approach while keeping the scalability and the efficiency of the classical retrieval and expansion techniques. As the utilization of Galois lattice in IR have not been extensively investigated, our study will provide some preliminary indication on the usefulness of Galois lattices for query expansion in IR.

One of the key difficulties in information retrieval is the assignment of weights to the keywords in the query. When we find the new keywords to be added to the query we have to assign weights to the new keywords. Unfortunately, there are no available guidelines to follow. The weights of the original keywords, the support and confidence values of the association rules that suggested the addition of a keyword are factors that contribute to the weight computation. We design a series of experiments and try to find a reasonable strategy for the task.

The thesis is structured as follows. In Chapter 2 we introduce the theoretical foundation of information retrieval discipline. First, we present the general process of information retrieval. We also explain the $tf \times idf$ weighting scheme and vector space model in detail. The vector space model is a widely used model to measure the similarity of a document and a query. It is implemented in the SMART system, which is briefly described since it is used in our experiments. In the same chapter we also show how to measure the system performance in terms of precision and recall. In Chapter 3 the foundations of the association rule mining problem is explained. We introduce in detail the Galois lattice and iceberg lattice, which consists of only the top-most concepts of a concept lattice. We also describe the algorithms to construct the lattice and their implementations. We introduce the Galicia platform [VGRH2003]. The algorithms are implemented in this platform and all the association rules are generated by the platform. Two algorithms for lattice construction are described in Chapter 3: Bordat [B1986] algorithm and incremental algorithm [VHM2003]. The first algorithm is a classical one. We did several modifications to make it easy to integrate into the Galicia platform and to generate iceberg lattices. Incremental algorithms have been developed in recent years. It makes sense since it reflects the fact that in most cases the document collections are dynamic. New documents will be added into the collections from time to time. These algorithms can avoid reconstructing the whole lattice from beginning every time there

are new documents added into the collection. Chapter 4 shows an application of association rules in query expansion. We describe the process of query expansion and the algorithm implemented for it. Several experimental results are presented in Chapter 5 and we also make an analysis based on the results. We got a slight improvement of system performance in the experiments. We may conclude that the association rules can help query expansion. However, in order fully to benefit from the advantages of the data mining mechanisms, a set of problems need to be solved. In Chapter 6 we propose several future research avenues.

Chapter 2 Information retrieval

2.1 Information retrieval

2.1.1 Introduction

Information retrieval is a science about the representation, storage, organization and accessing of information items [SM1988]. Although information retrieval systems are traditionally used to process bibliographic records and textual data, there is no restriction on the type of information items. The representation and storage of information items should help the users access the information easily. There may be many information items available. People may be interested in only part of them and different people may have different interests. An information retrieval system should provide the user with a convenient and effective way to access the information he is interested in.

To illustrate the whole information retrieval process first we give some definitions. A *database* or a *document collection* is a collection of documents. Here we only discuss text documents. A *document* is a sequence of words. It is written in a natural language such as English or French. A *term* is a unit extracted from a document in an indexing process. It represents a part of the document content. A term can be a word, or just a root of a word. It can also be a more complex element such as a compound term or a phrase. In this thesis, we will only deal with single words or roots of words. A *query* is a request to the database for finding documents relevant to some topic. Generally a query is also represented as a sequence of terms.

The information retrieval process can be separated into several steps, as shown in Figure 2.1.1. [BR1999]

Step 1: Define the text database. This step specifies the collection of documents to be used and the text model. The text model provides the structure of the text and it also determines what elements can be retrieved. In this step, text operations transform the original text and generate a logical view of the documents.

Step 2: Build the index of the documents. *Indexing* is critical because it allows fast searching over a large database. There are several different index structures but the most popular one is *inverted file* (See 2.1.2 for more details).

Step 3: The user sends a request (also called user need) in natural language to the system. The same text operations as in step 1 are also applied to generate the logical view of the request. Query operations are applied to generate actual query, which is a system representation of user's request.

Step 4: The Query is processed to obtain retrieved documents, typically by computing the similarity, or the likelihood of relevance, of documents to the query.

Step 5: The retrieved documents are ranked according to a likelihood of relevance and then presented to the user.

There may be some optional operations such as user feedback. We will not deal with them in this study.

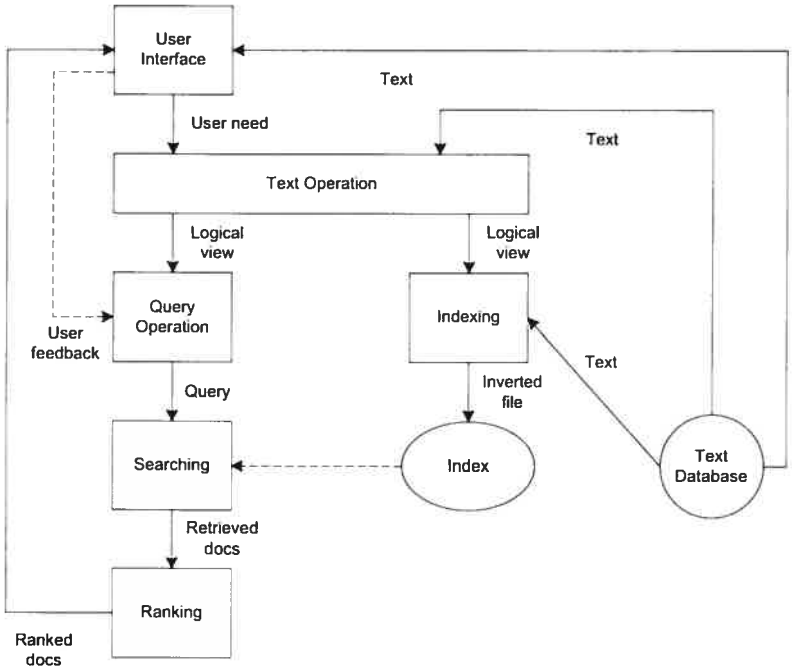

Figure 2.1.1 Information retrieval process

We will describe in more detail step 2 in Section 2.1.2. Step 4 and 5 will be discussed in Section 2.1.3.

2.1.2 Automatic indexing

The most crucial and difficult procedure in the process of information retrieval is indexing. The main task is to select appropriate terms which can represent the content of the documents. The selection of content identifier must fulfill three requirements:

1. To help locating the documents that the user is interested in.
2. To help the discovery of the relationship between documents, i.e., different documents dealing with similar or related topic areas.
3. To help the prediction of the relevance of documents to the users requirements.

The indexing task consists of two parts. The first is to select terms capable of representing document contents. The second is to assign each term a weight to represent its importance as a content identification.

Generally terms are extracted from the text of the documents themselves. Inverted file designs are applied in most information retrieval systems. So our discussion will focus on such kind of systems. In the inverted file design, a system constructs document index files and one or more auxiliary files storing the so called *inverted index*. The inverted index associates with each indexing term the list of documents containing the term. That is, $t \rightarrow \{< d_1, w_1 >, < d_2, w_2 > ...\}$. Here, t is a term, d_i is a document and w_i is the weight of the term in the document. When a user submits a query composed of one or more terms, the system searches in the inverted index to find the terms and their associated document reference numbers. Finally by combining the document reference lists, the list of required documents is composed, and the references are returned to the user.

Before indexing, the words in the documents will be pre-processed. First they should be filtered by a stoplist. A stoplist is a list of uninteresting words in indexing. In English functional words such as "a, be, have, of" and some common words such as "he, she, her, here" are included in a stoplist. These words are ignored during the procedure of indexing. Second, words should be "stemmed" to extract the root of the words. In most cases, morphological variants of words have similar semantic interpretations and can be considered as equivalent in information retrieval system. For example, the words "transform, transformation, transforming" should be stemmed to a single form "transform". After stemming we will get the standard forms of the words.

Once terms are selected, they should be weighted. There are several methods used for weighting terms. Here we introduce the method based on $tf \times idf$ weighting [BR1999].

The $tf \times idf$ weighting method takes two ideas into account for term weighting. Here tf represents the term frequency in a given document. It can be used to measure the importance of a term in a document. The value idf represents the inverse document

frequency of the term in the whole document collection. Typically, it is measured by *log(N/n)* where *N* is the total number of documents in the collection and *n* is the number of documents that contain the term. It can be used to measure how specific the term is to a document. So we can use $tf \times idf$ to set the weight of each term. Note that the term weight needs to be normalized to avoid that a longer document is always given a higher weight. The following formula is one of the most used $tf \times idf$ weighting schemas ——Cosine formula:

$$w_{ik} = \frac{tf_{ik} \times \log(N / n_k)}{\sqrt{\sum_{k=1}^{t} (tf_{ik})^2 \times [\log(N / n_k)]^2}} .$$

Where:

w_{ik} is the weight of term T_k in document D_i.

tf_{ik} is the frequency of term T_k in document D_i.

N is the total number of documents in the collection.

n_k is the number of documents in the collection that contain T_k.

$\log(N / n_k)$ is the formula to calculate idf of term T_k.

2.1.3 Vector space model

The Vector space model is one of the most commonly used models to measure the similarity of document and query [BR1999]. After indexing, each document in a given collection is represented by a set of terms. When we take the union of all these sets of terms we get a set of terms that defines a vector space: each term defines a different dimension. A document vector is defined by the set of terms occurring in it and their weights determined during the indexing process. If a term does not appear in the

document, its weight is set to zero. In this way a document D_i can be treated as a point in the document space and the weight of each term in D_i represents the coordinate of D_i. That also means that D_i can represent a vector from the origin of the document space to the point defined by the coordinate of D_i. That is:

$D_i =< w_{i1}, w_{i2}, ..., w_{in} >$.

Here, $w_{i1}, w_{i2}, ..., w_{in}$ are the weights of terms $T_1, T_2, ...T_n$ in document D_i. $w_{im} = 0$ if term T_m is absent in D_i.

So the collection can be represented by a document-term matrix as shown in Table 2.1.1. Each row of the matrix is a document and each column is a term. An element of the matrix is the weight of the term in the document.

	T_1	T_2	...	T_j	...	T_n
D_1	w_{11}	w_{12}	...	w_{1j}	...	w_{1n}
D_2	w_{21}	w_{22}	...	w_{2j}	...	w_{2n}
...	...	...	...	...	...	...
D_i	w_{i1}	w_{i2}	...	w_{ij}	...	w_{in}
...	...	...	...	...	...	...
D_m	w_{m1}	w_{m2}	...	w_{mj}	...	w_{mn}

Table 2.1.1 Matrix representation of Collection

Generally a query is specified by the user in natural language. The query will be processed in the same way as a document. So it is also transformed into a vector in the vector space. That is:

$Q = < w_{q1}, w_{q2} ..., w_{qn} >$.

Here, $w_{q1}, w_{q2} ..., w_{qn}$ are the weights of terms $T_1, T_2, ...T_n$ in query Q. $w_{qm} = 0$ if term T_m is absent in Q. The weights of terms in a query are compute in the same way as in a document.

When we have the document vectors and the query vector we can measure the similarity between them. All the documents can be ranked according to their similarity values. The document that is most similar to the query gets the highest rank. In the vector space model we assume that the document most similar to the query will be most relevant to the query. The usual way to measure the similarity is by the inner product [BR1999]:

$$Sim(D_i, Q) = \sum_{k=1}^{n} (w_{ik} \times w_{qk})$$

Remember that all weights are already normalized here by the Cosine formula. Putting the earlier Cosine formula into the similarity measure, we have:

$$Sim(D_i, Q) = \frac{\sum_{k=1}^{n} (w_{ik} \times w_{qk})}{\sqrt{\sum_{k=1}^{n} (w_{ik})^2 \times \sum_{k=1}^{n} (w_{qk})^2}}$$

This is the cosine of the angle between the document vector and the query vector. After we calculated the similarity of each document with the query and ranked all the documents in the reverse order of their similarities, the documents are returned to the user in the order of their ranking.

2.1.4 System performance evaluation

With respect to a given query, the collection of the documents can be divided into four groups according to the relevance and the retrieval result, as shown in Table 2.1.2.

	Relevant	Non-Relevant
Retrieved	A	B
Not Retrieved	C	D

Table 2.1.2 Document Group

Two main measures of system performance can be derived from Table 2.1.2:

Definition 2.1.1 Given a set of documents, *Precision* is the fraction of relevant documents retrieved from the total number retrieved.

$$Precision = \frac{A}{A \cup B}$$

Definition 2.1.2 Given a set of documents, *Recall* is the fraction of relevant documents retrieved from the set of total relevant documents.

$$Recall = \frac{A}{A \cup C}$$

Ideally, we hope that a system can give a high value of precision and recall at the same time. A system with 100% precision and recall values means that it finds all the relevant documents and only the relevant documents.

Different user queries may get different precision-recall pair values. To evaluate the performance more accurately the concept of *average precision* is introduced. The main idea is to average the precision values of different queries. Two methods are employed mostly to calculate the average precision.

Method 1:

$$Precision_{avg} = \frac{1}{N}\sum_{i=1}^{N} Precision_i = \frac{1}{N}\sum_{i=1}^{N} \frac{A_i}{A_i \cup B_i}$$

$$Recall_{avg} = \frac{1}{N}\sum_{i=1}^{N} Recall_i = \frac{1}{N}\sum_{i=1}^{N} \frac{A_i}{A_i \cup C_i}$$

Here, N is the number of queries. The subscript *"i"* represents a particular query *i*. Using this method, N queries are sent to the system and the precision and recall values of each query are recorded. Then we use these values to calculate the average *precision* and *average recall* value of the system.

Method 2:

$$Precision_{avg} = \frac{\sum_{i=1}^{N} A_i}{\sum_{i=1}^{N} (A_i \cup B_i)}$$

$$Recall_{avg} = \frac{\sum_{i=1}^{N} A_i}{\sum_{i=1}^{N} (A_i \cup C_i)}$$

Using this method, we also send N queries to the system. But we don't calculate the precision and recall values of each query. Instead, we record the values of *A, B, C* of each query. Then we calculate the average value of *A, B, C* and finally we compute the average precision and average recall value as a single query.

Precision and recall are not independent. There is a strong relation between them: when one increases, the other decreases. Measurements of Precision-Recall are not static. A system does not have only one measurement of Precision-Recall. The behaviour of a system can vary in favour of precision or of recall. Thus, for a system, there is a curve of Precision-Recall which has the general form as shown in Figure 2.1.2 [BR1999].

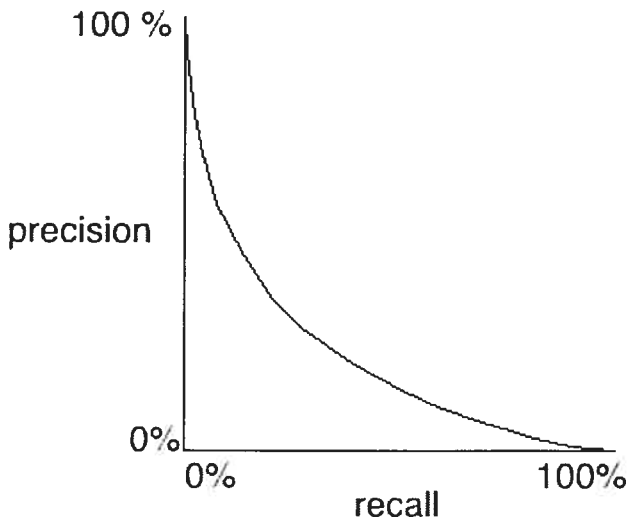

Figure 2.1.2 Precision-Recall graph

To draw a Precision-Recall graph, one simply computes the precision at 0.1, 0.2... 1.0 level of recall. The precision-Recall graph shows the trade-off between precision and recall made by the search algorithm of a system. It shows the system performance at different operating points. Different users may be interested in different point on the graph. Mean Average Precision is also defined:

Mean Average Precision = Average of Precisions at 11 points of recall.

2.1.5 Query Expansion

In most of the popular search engines, the user interacts with the system by means of queries. In order to find as many relevant documents as possible and as few non-relevant ones as possible, the users should compose their information need accurately in the form of query which is accepted (or 'understood') by the search engine. There are two major problems in the task. The first is that generally the users can not express their information needs in a precise way. What they can do is just give an approximate description of what they want. For example: if a user wants to find some documents about the new techniques applied in query expansion, he may use only "query

expansion" as the query. Obviously the search engine will return a very large set of documents that contain the term "query expansion". The user may need only part of the set. In this case the user has to refine his/her query. The second problem is that most users do not know much about the search engines. Even if they know how to express their information needs precisely, they do not know how to transform it into the form that can be understood by the search engine. So, in most cases, their queries are incomplete and inaccurate. To solve this problem, some information retrieval systems provide some kind of user interaction that can help the user optimize their query. To that end, a range of techniques has been deployed, including query expansion, which adds new terms into the original query to improve search performance.

We divide query expansion research into two major areas according to [QF1993]. The first one relies on semi-automatic and automatic query expansion techniques. In automatic query expansion, the system determines a set of related terms to be added into the original query by using either a thesaurus or by exploiting the documents returned by the system. The second one is the manual approach. In this approach, people examine the user's query and use their experience or other auxiliary tools such as thesaurus to derive or modify the original query during the query expansion. In my thesis I will focus on the automatic query expansion.

Our query expansion approach is based on the term co-occurrences. Various methods for exploiting term co-occurrences in query expansion have been proposed for decades. They can be classified into four groups [QF1993]:

1. Use of term classification. The system calculated the similarities between terms based on the classification hypothesis. Then the system classifies terms by setting a similarity threshold value. The terms in the same class will be treated as equivalent. The system will expand the query by adding all the terms of the classes that contain query terms. Researchers have analyzed the limitation of such method [PW91], and

it turns out that this idea is too simple to be applied in practice because it adds a lot of noise into the query.

2. Use of document classification. First the system classifies the documents using a document classification algorithm. Terms in the same document class are considered to be similar and will form a term class, also called thesaurus class. Thesaurus class can be used to enhance both the indexing and query process. The query is expanded by replacing a term by a thesaurus class. According to [CY92], some parameters, which have strong impact on retrieval effectiveness, are hard to determine. Since most of the document collections are highly dynamic, a system may run document classification more often than run term classification. Furthermore, most of the document collections are very large. The number of documents may be much larger than the number of terms in the collection. Thus, document classification is much more expensive than term classification.
3. Use of syntactic context. The similarities between terms are computed based on the linguistic knowledge and co-occurrences statistics. A grammar and a dictionary are used to extract for each term a list of modifiers. The modifiers in the list are used to calculate the similarities between terms. At last a query is expanded by adding those most similar terms. According to [G1992], this method can only slightly improve the system performance.
4. Use of relevance information. Pseudo relevance feedback is a commonly used approach for automatic query expansion [MSB1998]. In this method, first the original user query is used to retrieve a small set of documents. These documents are assumed to be relevant and used in a relevance feedback process to extract terms to expand query. The problem of this method is that if the assumption is wrong, say, a large portion of documents retrieved in the first step are not relevant, then the terms added to the query are likely to be unrelated to the user's need, and thus damage the system performance.

In this thesis we develop an approach for automatic query expansion based on association between terms. The associations are extracted from documents by a data mining technique called Association Rule Mining. It is also a method based on the co-occurrences between terms. However, it is different from the simple use of co-occurrences data. In the latter one, the relations between terms are bi-directional. They are treated as 'equal'. In our application, the relations between terms are asymmetrical. Some of terms are premise and others are treated as consequence. We think that if we can find insightful rules in our application then we can get a much more relevant result when applying them to the process of query expansion. The interpretation of the rules may make the process more reasonable. If it is the case, the technique can also be used to interact with users to get better results.

An association rule has the follow form:

t1, t2-->t3, t4, with *Support*=V_s, *confidence*=V_c, where

t1, t2, t3, t4 are terms, V_s, V_c are values of rule support and rule confidence, respectively. The rule confidence can be understood as the conditional probability, $V_c = P(t_3,t_4|t_1,t_2)$. The meaning of the rules is: term *t3, t4* will appear in a document when *t1, t2* both appear in that document with probability V_c. So if *t1, t2* appear in a query together, it is natural to expand that query with *t3, t4*. In contrary, unlike other term co-occurrence techniques, if if *t3, t4* appear in a query together, we will not use *t1, t2* to expand that query, i.e, use the rule in the reverse direction. This is reasonable since the probability of *t1, t2* appear in a document when *t3, t4* both appear in that document may be different from V_c. We will give a more detailed description about association rules and their meaning in Chapter 3.

2.2 SMART System

2.2.1 SMART System

SMART system is an experimental system of information retrieval [G1971]. It uses the vector model. Many current IR systems are inspired by SMART. For retrieval effectiveness, SMART is comparable to the commercial search engines available on the Internet. It is also a free system for research with source code. So it is an ideal framework to do our experiment.

Here we give a brief introduction to the system.

1. Indexing

   A database is a collection of text documents. A text document may contain various fields. A field is called section in SMART. For example, in a document we can distinguish the title, the author, the date and the text. SMART makes it possible to treat these fields in a different way. A query may also contain various fields.

   The indexing of a document has several steps:

   Step 1: Recognition of sections. This is done according to markers which denote the beginning of sections. SMART can define the markers and corresponding actions of indexing.

   Step 2: Tokenization. For each section that SMART indexes, the system cuts it in words. This consists in recognizing the separator of words, such as ",", ":", space, etc. The output of this step is a sequence of tokens.

   Step 3: Stoplist. The stoplist contains all the words that we do not want to keep as index. The *tokens* found in step 2 are compared with the stoplist to remove the insignificant ones. This step is optional since we may decide that all the words are significant and must be kept.

   Step 4: Stemming. This treatment transforms the tokens into a certain standard form. Three *options* are offered in SMART: no stemming; removing the

"s" at the end of a word; or using a trie to remove the termination. After stemming, the result word is compared with a dictionary. The dictionary stores all the indices and the identification number of each index. The purpose of the comparison is to obtain the identification number for a word. If the word is already stored in the dictionary, we simply take its identification number. If not, the system creates a new identification number for the word automatically and adds an entry into the dictionary for the new word.

Step 5: Statistic. For each *word* the system makes a statistic of its frequency of occurrence in the document. The final result for a document is a list of word identification numbers with their frequencies.

Step 6: Conversion of the result. The raw indexing results can be further processed through a conversion process. Conversion is done only when all the document is the collection were *indexed*. Two kind of conversion can be made in SMART: To convert vectors into inverted file or to convert the weight of terms. The $tf \times idf$ weighting can be employed in this step.

The indexing of a query is very similar to that of a document.

2. Similarity measurement

The evaluation of query utilizes the inverted file of documents and the indexing result of the query which is a sequence of word identification number and its weight. SMART uses inner product as similarity.

$$Sim(d,q) = \sum_i (p_i \bullet q_i)$$

Here, d is a document, q is the query, p_i and q_i are the weight of a word in document d and query q.

The SMART system implements a calculation for all documents at the same time. At the end of this calculation, we obtain the value of *Sim* for every document. The documents will be ranked according to the *Sim* values and returned to the user.

Chapter 3 Mining of association rules

3.1 Association rules

3.1.1 Introduction

Association rules are very widely applied for knowledge discovery and data mining in database [AIS1993]. The ideas of mining association rules come from market-basket analysis. It is a process of analysing customer buying habits (patterns) which are expressed by rules such as "A customer who buys a desktop computer will also buy an operating system with a certain probability." This kind of rule is called association rule since it expresses the patterns of co-occurrence of different products in the customer's basket. In a database of commercial transactions, association rule mining is used to find all interesting co-occurrence patterns of items.

A collection of documents can be treated as a database and each document is a record of this database. Terms can be seen as items of the record. This motivates us to apply association rules mining in document analysis to find co-occurrence patterns of terms in a document collection. As we mentioned in Chapter 2, these co-occurrence patterns can be used to expand a query to achieve better search performance.

3.1.2 Association rules mining

Association rules are used to identify relationships among a set of items in a transaction database. The relationships are based on co-occurrence of the data items.

A formal notation and definition of the association rule is as follows: [AIS1993]

Let $I = \{i_1, i_2, \ldots, i_n\}$ be a set of distinct attributes, also called items. A transaction set T is a multi-set of subsets of I that are identified by TID.

Definition 3.1.1: A subset $X \subseteq I$ with $|X| = k$ is called a *k-itemset*. The fraction of transactions that contain X is called the *support* (or *frequency*) of X, denoted by *supp* (X):

$$supp(X) = \frac{|\{t \in T \mid X \subseteq t\}|}{|T|}.$$

Definition 3.1.2: If the support of an itemset X is above a user-defined minimal threshold (*minsupp*), then X is frequent (or large) and X is called a *frequent itemset* (*FI*).

Definition 3.1.3: An association rule is an implication of the form $X \Rightarrow Y$, where $X, Y \subset I$, and $X \cap Y = \phi$. Here X is called *antecedent*, and Y *consequent.*

Support and *confidence* are two important measures for association rules. They are defined as follows.

Definition 3.1.4: The *support* (s) of an association rule is the ratio of the number of transactions that contain $X \cup Y$ to the total number of transactions in the transaction set. That is:

$$s(X \Rightarrow Y) = \frac{|X \cup Y|}{|D|}$$

Definition 3.1.5: The *confidence* (c) of an association rule is the ratio of the number of records that contain $X \cup Y$ to the number of transactions that contain X. That is:

$$c(X \Rightarrow Y) = \frac{|X \cup Y|}{|X|}$$

Confidence of a rule indicates the degree of correlation in the itemset between X and Y. It is a measure of a rule's strength. For a given association rule $X \Rightarrow Y$, a higher confidence means a higher probability that the itemset Y is present in transactions which contain itemset X. Support is the statistical significance of an association rule. It is a useful measure since rules with low support mean this kind of pattern rarely occurs in the transaction set. Even if such a rule has a very high confidence, it is still not interesting for the analyzer. So support measure is often used to eliminate uninteresting rules. *Minsupp* is a minimum threshold value of support specified by the user. An itemset X is frequent when *supp(X)* reaches *minsupp*. Similarly, *minconf* is defined as a minimum threshold value of confidence.

The association rule mining problem can be formally described as follow:

> Given a set of transactions T, *minsupp*, *minconf*, find all association rules with support $\geq$ *minsupp* and confidence $\geq$ *minconf*.

The total number of possible rules of a transaction dataset D with itemset I grows exponentially with the number of items. So it's impractical to enumerate all rules and then calculate their support and confidence values. To reduce the computation overhead we should find an algorithm that can prune the useless rules (i.e. support lower than *minsupp* or confidence lower than *minconf*) earlier without computing their support and confidence values. From **Definition 3.1.4** we know that the support of a rule $X \Rightarrow Y$ depends only on its itemset $X \cup Y$. For example, for the following rules

$\{c\} \Rightarrow \{d,g\}, \{d\} \Rightarrow \{c,g\}, \{g\} \Rightarrow \{c,d\}$

$\{c,d\} \Rightarrow \{g\}, \{c,g\} \Rightarrow \{d\}, \{d,g\} \Rightarrow \{c\},$

The supports are identical since they correspond to the same itemset $\{c,d,g\}$. So if the itemset is infrequent (i.e., $s(\{c,d,g)\} < minsupp$), all these six rules can be discarded.

Thus, the problem of mining association rules with given minimum support and confidence can be split into two steps:

1. Detecting all frequent itemsets (*FI*s) (i.e., all itemsets that occur in the transaction set with a support ≥ *minsupp*).
2. Generating association rules from frequent itemsets (i.e., rules whose confidence ≥ *minconf*)

The second step is relatively straightforward [AIS1994]. However, the first step presents a great challenge because the set of frequent itemsets still grows exponentially with $|I|$.

The computation of frequent itemsets is the most time consuming operation in association rule generation. Researchers have proposed search space pruning techniques based on the computation of frequent closed itemsets (*FCI*s) only, without any loss of information [PBTL1999].

The key property of *closed itemset(CI)* is any itemset has the same support as its closure since it is as frequent as its closure. *CI*s and *FCI*s may be used in the generation of all *FI*s and rules, without further accessing the transaction database.

In the next two sections we present a framework for generating *FCI*s. The generation of rules from *FCI*s is not discussed in this thesis.

3.1.3 Formal concept analysis

Concept lattice is a structured graph for conceptual hierarchy's representation of knowledge. Formal Concept Analysis (FCA) [W1982] is an approach for data analysis based on the lattice theory. In the following we give a formal notations and definitions for FCA.

Definition 3.1.6: Given a set G, a *partial order* $\leq$ on G is a binary relation that is reflexive, transitive and anti-symmetric. A set G subject to a partial order $\leq_G$ is called a *partially ordered set* or *poset* or *partial order,* which is denoted by $(G, \leq_G)$.

Definition 3.1.7: Given a partial order $P = (G, \leq_G)$ and two elements, $s, p \in G$, if $p \leq_G s$, we call p the *predecessor* of s and s the *successor* of p. The set of all common successors of s and p are called *upper bounds* of s and p. The set of all common predecessors of s and p are called *lower bounds* of s and p.

Definition 3.1.8: Given a partial order $P = (G, \leq_G)$ and A, a subset of G, if $\exists s \in G, s = \text{Min}(\textit{upper bounds of } A)$, s is called the *least upper bound* of A (LUB), denoted $sup(A)$; if $\exists p \in G, p = \text{Max}(\textit{lower bounds of } A)$, p is called the *greatest lower bound* of A (GLB), denoted $inf(A)$.

Definition 3.1.9: Given a partial order $P = (G, \leq_G)$ and $s, p \in G$, if $s <_G p$, $\exists t$, where $s \leq_G t \leq_G p$ and $t = s$ or $t = p$, s is called the *immediate predecessor* of p and p the *immediate successor* of s.

Hasse diagram is used to represent P by its covering graph $Cov\ (P) = (G, <_G)$. In Hasse diagram, each element s in G is connected to both the set of its immediate predecessors (called *lower covers* (Cov^l)) and of its immediate successors (called *upper covers* (Cov^u)).

Definition 3.1.10: A partial order $P = (G, \leq_G)$ is a lattice if and only if $\forall x, y \in G$, $inf(\{x, y\}) \in G$ and $sup(\{x, y\}) \in G$. The lattice is denoted as $L = (G, \leq_L)$. In

a lattice $inf(\{x,y\})$ is called the *meet* of x,y, denoted as $x \wedge_L y$, $sup(\{x,y\})$ is called the *join* of x,y, denoted as $x \vee_L y$.

Definition 3.1.11: If given a *lattice* $L=(G,\leq_L)$, $\forall Y \subseteq G, \exists\, inf(Y) \in G$, $sup(Y) \in G$, we call this lattice a *complete lattice.*

Definition 3.1.12: A *structure* with only one of the *join* and *meet* operations is called a *semi-lattice.*

Definition 3.1.13: A *formal context* is a triple $K=(O,A,I)$ where O and A are sets and I is a binary (incidence) relation, i.e., $I \subseteq O \times A$. The elements of O are called *transactions* (or *objects*), and the elements of A are called *items* (or *attributes*).

Definition 3.1.14: Let $K=(O,A,I)$ be a formal context, the function f maps a set of objects into the set of their common attributes, whereas g is the dual for attribute sets. f and g are denoted by ' :

$-f: P(O) \rightarrow P(A), f(X) = X' = \{a \in A \mid \forall o \in X, oIa\}$
$-g: P(A) \rightarrow P(O), g(Y) = Y' = \{o \in O \mid \forall a \in Y, oIa\}$

$P(O) = 2^O$ is the power set of O, $P(A) = 2^A$ is the power set of A,

The couple (f, g) defines a *Galois connection* of the formal context.

Definition 3.1.15: $f \circ g\,(Y)$ and $g \circ f\,(X)$ are *Galois closure operators* over 2^A and 2^A respectively. Hereafter, both $f \circ g\,(Y)$ and $g \circ f\,(X)$ are noted by ".

$$f \circ g\,(Y) = f(g(Y)) = Y"$$
$$g \circ f\,(X) = g\,(f\,(X)) = X"$$

Definition 3.1.16: An itemset X is closed if $X = X''$.

If an itemset X is closed, adding an arbitrary item i from I-X to X resulting a new itemset $\underline{X}$ which is less frequent [PHM2000].

Property 3.1.1: If X is closed ($X=X''$), then $\forall\ i \in I\text{-}X$, $supp\ (X \cup \{i\}) < supp\ (X)$.

Definition 3.1.17: If a closed itemset (*CI*) X is frequent, and then we call it a *frequent closed itemset* (*FCI*). That is: $FCI = \{X \mid X \in CIs \wedge supp\ (X) \geq minsupp\}$.

Definition 3.1.18: Give a formal context $K = (O, A, I)$, a *formal concept* c is a couple (X, Y), where $X \in P(O)$, $Y \in p(A)$, $X = Y'$ *and* $Y = X'$. X is called the *extent* and Y the *intent* of the concept.

For a concept $c = (X, Y)$, since $X = Y' = \{X'\}' = X''$, so the intent of a concept is a closed itemset.

Definition 3.1.19: The *support* of a concept $c = (X, Y)$ is defined as the ratio of the size of X to the size of O, $supp\ (c) = \dfrac{|X|}{|O|}$.

Definition 3.1.20: If the support of a concept c is above a user-defined minimal threshold (*minsupp*), then c is *frequent* and c is called a *frequent concept*.

Given a set of concepts C, the partial order is defined as:

$\forall (X_1, Y_1),\ (X_2, Y_2) \in C,\ (X_1, Y_1) \leq (X_2, Y_2)$ iff $(X_1 \subseteq X_2) \wedge (Y_2 \subseteq Y_1)$

Definition 3.1.21: Given a set of concepts C, The partial order $L = (C, \leq_L)$ is a complete lattice called a *concept lattice* with *joins* and *meets* as follows [W1982]:

$$(X_1, Y_1) \vee_L (X_2, Y_2) = (\{X_1 \cup X_2)\}'', \{Y_1 \cap Y_2\})$$

$$(X_1, Y_1) \wedge_L (X_2, Y_2) = (\{X_1 \cap X_2)\}, \{Y_1 \cup Y_2\}'')$$

The Hasse diagram of the concept lattice L from Table 3.1.1 is shown in Figure 3.1.1. The numbers in the nodes are numbers of the concepts. Intents and extents are indicated in rectangles below the nodes. For example, the join and meet of $c_{\#13} = (\{2, 5, 9\}, \{f, h\})$ and $c_{\#16} = (\{2, 3, 9\}, \{d, f, g\})$ are $c_{\#4} = (\{2, 3, 6, 7, 9\}, \{f\})$ and $c_{\#25} = (\{2\}, \{a, c, d, f, g, h\})$ respectively.

Figure 3.1.1 is an example of the Hasse diagram of lattice L drawn from $K = (\{1,2,...9\}, \{a,b,...h\}, R)$.

	a	b	c	d	e	f	g	h
1	x	x	x	x			x	
2	x		x	x		x	x	x
3				x	x	x	x	
4							x	x
5					x	x		
6	x		x	x	x			
7		x		x	x	x		
8			x					
9			x	x		x	x	x

Formal context $K = (T = \{1,2,...9\}, I = \{a,b,...h\}, R)$

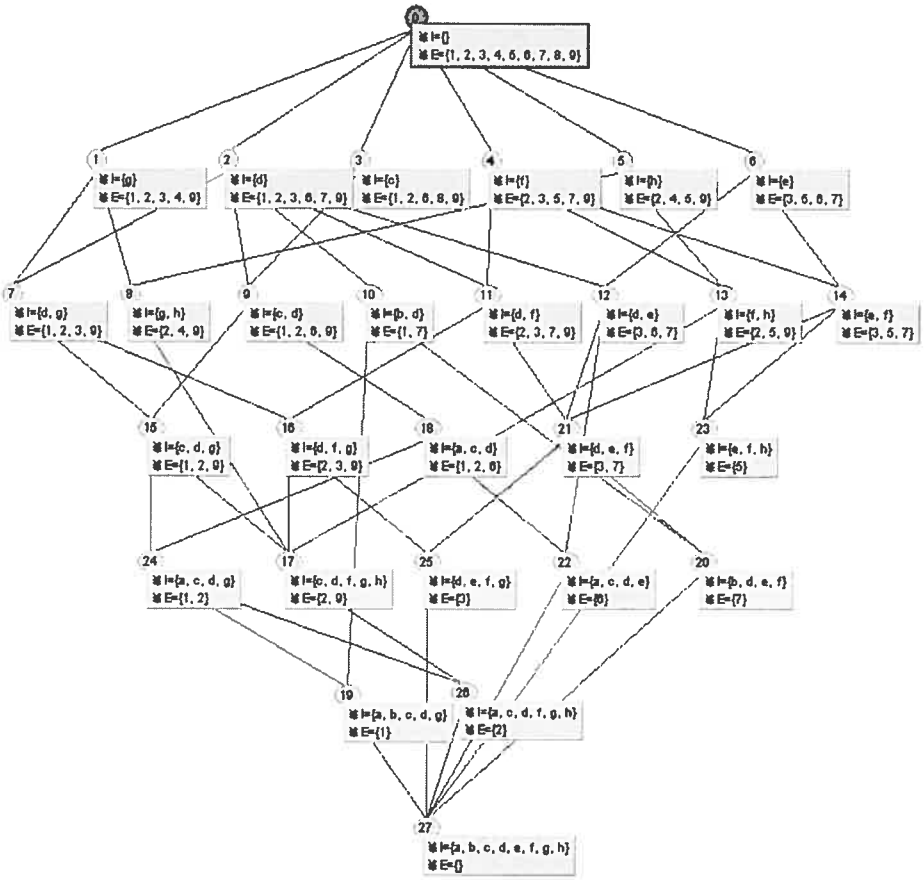

Figure 3.1.1 The Hasse diagram of the Galois lattice derived from K

3.1.4 Iceberg Concept Lattices

In the worst case, the size of a concept lattices is exponential in the size of the context. So for most application the size of the concept lattices is too large to be used. The user has to find strategies to cut off the lattice and keep relevant parts of the lattice structure at the same time. Here we present an approach named *iceberg concept lattice* which is based on frequent closed itemsets. The iceberg concept lattices will consist only of the top-most concepts of the concept lattice.

The set of all frequent concepts of a context K is called the iceberg concept lattice of the context K.

The notion and properties of iceberg concept lattice were introduced in [STBPL2000].

Definition 3.1.30: Given *minsupp* $\alpha \in [0, 1]$, C^{α} is the set of all α-frequent concepts and the partial order $(C^{\alpha}, \leq_P)$ is called the *iceberg concept lattice.*

The support function is monotonously decreasing:

$A_1 \subseteq A_2 \Rightarrow supp(A_1) \geq supp(A_2)$

So the iceberg concept lattice is an order filter of the whole concept lattice. In general, it is a sup-semi-lattice. This allows us to apply the algorithm for computing concept lattices and iceberg concept lattices. Here we give an example. For the binary table $K = (O = \{1,2,...9\}, A = \{a,b,...h\}, R)$ shown in Figure 3.1.1, given 30% as the *minsupp* value we can get the iceberg lattice as shown in Figure 3.1.2.

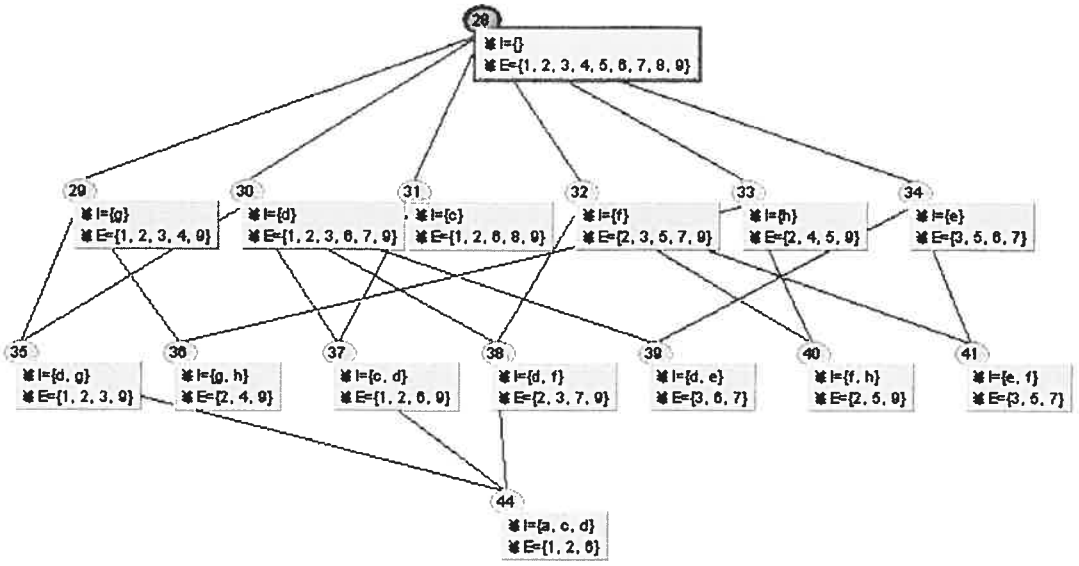

Figure 3.1.2 Iceberg lattice derived from K with *minsupp* value of 30%

3.2 Galicia Platform

3.2.1 Introduction

Galicia is an integrated software platform that can perform almost all the key operations on concept lattices. These kinds of operations might be required in both practical applications and more theoretically-oriented studies. Three major functions are includes in the platform [VGRH2003]: input of contexts, lattice construction and lattice visualization.

The architecture of the platform also includes some auxiliary functions such as interactive data inputs, result export to various formats, etc. Galicia performs all the necessary tasks during the complete life-cycle of a lattice. (See Figure 3.2.1)

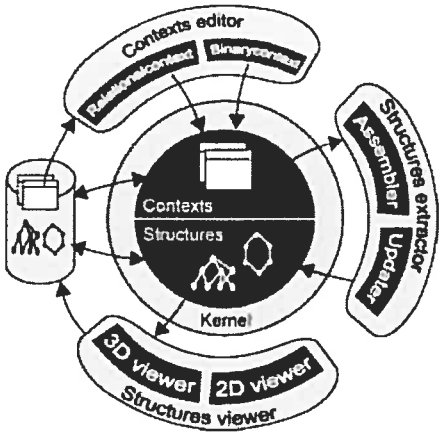

Figure 3.2.1: Architecture of Galicia

The Galicia platform can support the applications of FCA as well as the development of new techniques based on lattices. Some frequently occurring lattice sub-structures are built in, together with the respective manipulation mechanisms. Thus, for our research we can produce the whole lattice, some iceberg lattices and respective association rules within the platform.

3.2.2 Architecture of Galicia

Galicia has been designed as an open platform and its implementation in Java ensures the high portability of the entire system [VGRH2003].

The architecture of Galicia is a traditional layered architecture, with a lower-most level composed by the Java environment, and the upper one dedicated to interaction with the users of the platform (Figure 3.2.2). Two intermediate levels have been identified, the kernel of the platform and the tool layer.

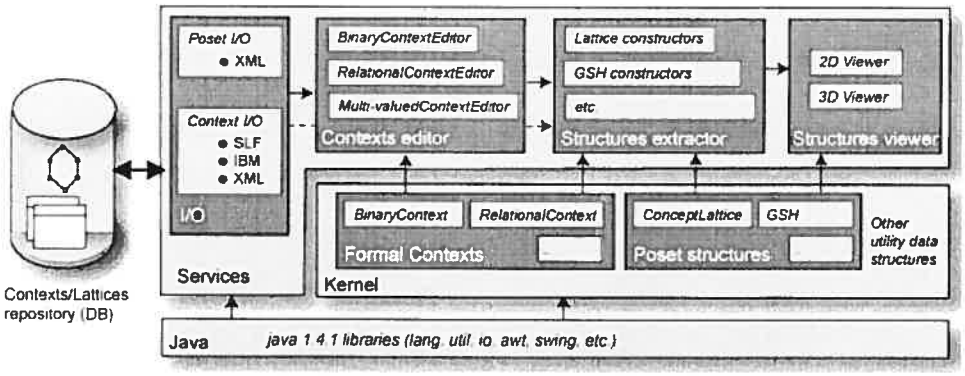

Figure 3.2.2: Dependencies between components of Galicia platform

In our research, we use Galicia to implement and develop algorithms for lattice construction and association rule generation.

Several algorithms have been implemented in Galicia platform. They are described in the following sections. We need to implement and adapt some algorithms that, although designed for lattices, can be used for iceberg lattice. To accommodate the dynamic database, we also need to implement algorithms that can construct lattices incrementally.

3.3 Bordat Algorithm

3.3.1 Description of the algorithm

The Bordat algorithm is a top-down batch algorithm. It consists of two basic steps:

- Compute the top concept *(O,O')*
- For each computed concept, generate all its lower neighbours.

Our implementation of Bordat Iceberg Lattice algorithm is based on the description of [KO2001]. A set of modifications was made. The first is the change of the algorithmic scheme from recursive to iterative. The second is that instead of using an auxiliary tree to construct the line diagram, we use the tree structure defined in the Galicia platform. The third is that we reversed the lattice traversal to top-down so that we can use the algorithm to generate iceberg lattice.

Below is the description of our implementation of the algorithm:

1. Procedure PROCESS(In: *top* the top concept of the lattice)
2. Local : *cand:*=∅ a linked list of concept
3. *cand*← *cand*∪ {*top*}
4. while c*and* is not empty
5. *c*←*cand*.getFirst
6. *Ln* ←FINDLOWERCOVER(*c*.extent, *c*.intent)
7. for all *e* in *Ln* do
8. if *e.supp*<*minsupp* continue
9. if *e is not processed before* then
10. *cand*← *cand*∪ { *e*}
11. end if
12. LINKCONCEPT2UPPERCOVER(*c*, *e*)
13. *cand*.removeFirst

1. FINDLOWERCOVER(In: *e* extent of concept c, In: *i* intent of concept *c,* Out: *Ln* a collection of concept)
2. *Ln*:= ∅
3. *a*:=*i*
4. while((*fo*=FINDFIRSTOBJECT(*a*,*e*))!=*null)*
5. *efo*=*{fo.extent}*
6. *ifo*=*fo.intent*
7. *h*:=*efo*
8. while(*e* has more objects)
9. *h*:=*e*.nextObject
10. if(¬((*ifo*∩*h*.intent) belong to *a*))
11. *efo*=*efo* ∪ *h*
12. *ifo*=*ifo*∩h.intent
13. if (*ifo*∩*a* equals *i*)
14. *Ln*:=*Ln* ∪ {(*efo*,*ifo*)}
15. *a*:=*a* ∪ *ifo*
16. Return *Ln*

In PROCESS procedure, *cand* is a linked list of concepts. It is used to store all the concepts that need to be processed to generate their lower covers in the lattice. *top* is the top concept of the lattice. The *top* concept is generated by Galicia system. It is the first concept put into *cand*. Lines 4 to 13 form a loop. At each iteration one concept in *cand* will be processed. The lower cover of the concept will also be put into *cand*. Since some concepts may have the same concept in their lower covers, Line 9 to Line 11 prevent a concept from being put into *cand* repeatedly. If the support value of a concept is lower than *minsupp* it will be discarded immediately (Line 8), and it will not

be used to generate lower covers since the support of any lower cover will also be lower than *minsupp* . The loop ends when *cand* becomes empty, which means the entire iceberg lattice was built.

LINKCONCEPT2UPPERCOVER is a function provided by the Galicia platform to construct the lattice structure. FINDFIRSTOBJECT is an auxiliary function and we do not show it here. In our implementation, if we set *minsupp* to zero, we will get the whole lattice, otherwise we get an iceberg lattice.

3.4 Incremental Algorithm

We also developed an incremental algorithm to construct the iceberg lattice. The incrementation can be either object-wise or attribute-wise. Here we only analyze the object-wise incremental algorithm. The attribute-wise one is in fact the dual although it has its own properties.

An incremental algorithm does not in fact construct the lattice. It just maintains the integrity of the lattice upon the insertion of new object into the context. [VHM2003]

The problem is defined as follows:

Given: a context $k = (O, A, I)$ with its lattice L and an object o,

Find: the lattice L^+ corresponding to $K^+ = (O \cup \{o\}, A, I \cup \{o\} \times o')$.

So an incremental algorithm can construct the lattice L starting form a single object o_1 and gradually incorporating every new object o_i into the lattice L_{i-1}. The concepts in the new lattice L_i can be divided into four categories:

1. $N^+(o)$: *New concepts* whose intents correspond to intersections of $\{o_i\}'$ with intents from C^a_{i-1}, which are not themselves in C^a_{i-1}.
2. $G^+(o)$: The set of *Genitors*. When inserting the new concepts into the lattice, each

new concept is preceded by a specific concept from the initial lattice, called its *genitor*. Its counterpart in L_{i-1} is denoted as $G(o)$.

3. $M^+(o)$: Modified concepts correspond to intersections of $\{o_i\}'$ with members of C^a_{i-1} that already exist in C^a_{i-1}. Its counterpart in L_{i-1} is denoted as $M(o)$.
4. $U^+(o)$: *Old* or *unchanged*: remain set of concepts in the initial lattice L_{i-1}. Its counterpart in L_{i-1} is denoted as $U(o)$.

Here is an example. Given the binary table $K^- = (O = \{1,2,4,...,9\}, A = \{a,b,...,h\}, R)$ and Object 3 in Table 3.4.1, the Hasse diagram of Galois lattice derived from K^- is shown in Figure 3.4.1. When inserting the Object 3 into K^- we get the Hasse diagram of Galois lattice derive from $K = (O = \{1,2,3,...,9\}, A = \{a,b,...,h\}, R)$, which is shown in Figure 3.4.2.

So in Figure 3.4.1, the three categories of concepts are $G(o)$ = {C_0, C_1, C_2, C_3, C_6, C_9, C_{10}}, $M(o)$ = {C_4, C_5, C_{11}}, and $U(o)$= {C_7, C_8}. $G^+(o)$, $M^+(o)$, and $U^+(o)$ are shown in Figure 3.4.2 with the same IDs as their counterparts shown in Figure 3.4.1. We can observe the changes on their extent and intent respectively. $N^+(o)$= {C_{13}, C_{14}, C_{15}, C_{16}, C_{17}, C_{18}, C_{19}}.

	a	b	c	d	e	f	g	h
1	x	x	x	x	x	x	x	x
2	x	x	x		x	x		
4					x	x	x	x
5							x	
6					x	x		x
7	x	x	x	x				
8		x	x	x				
9				x				
3			x	x		x	x	x

Table 3.4.1 Binary table $K^{-} = (O = \{1,2,4,...,9\}, A = \{a,b,...,h\}, R)$ and object 3

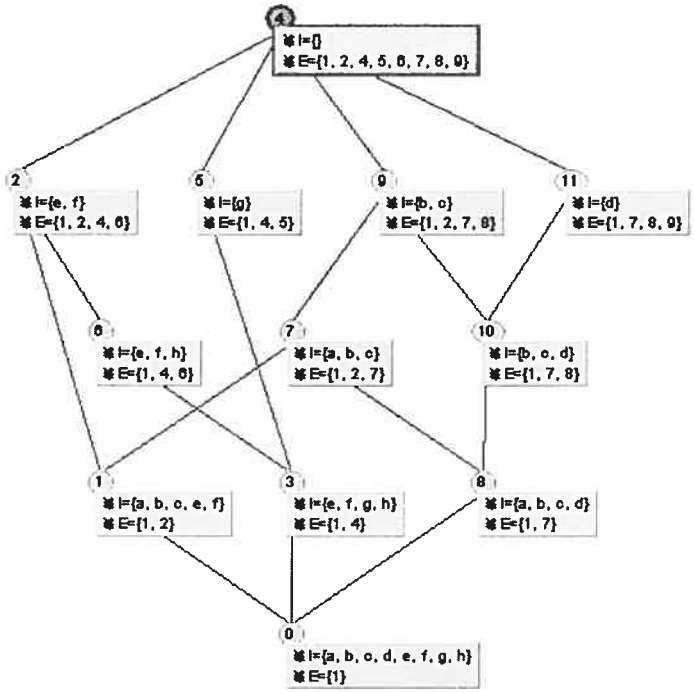

Figure 3.4.1 The Hasse diagram of the Galois lattice derived from K^{-}

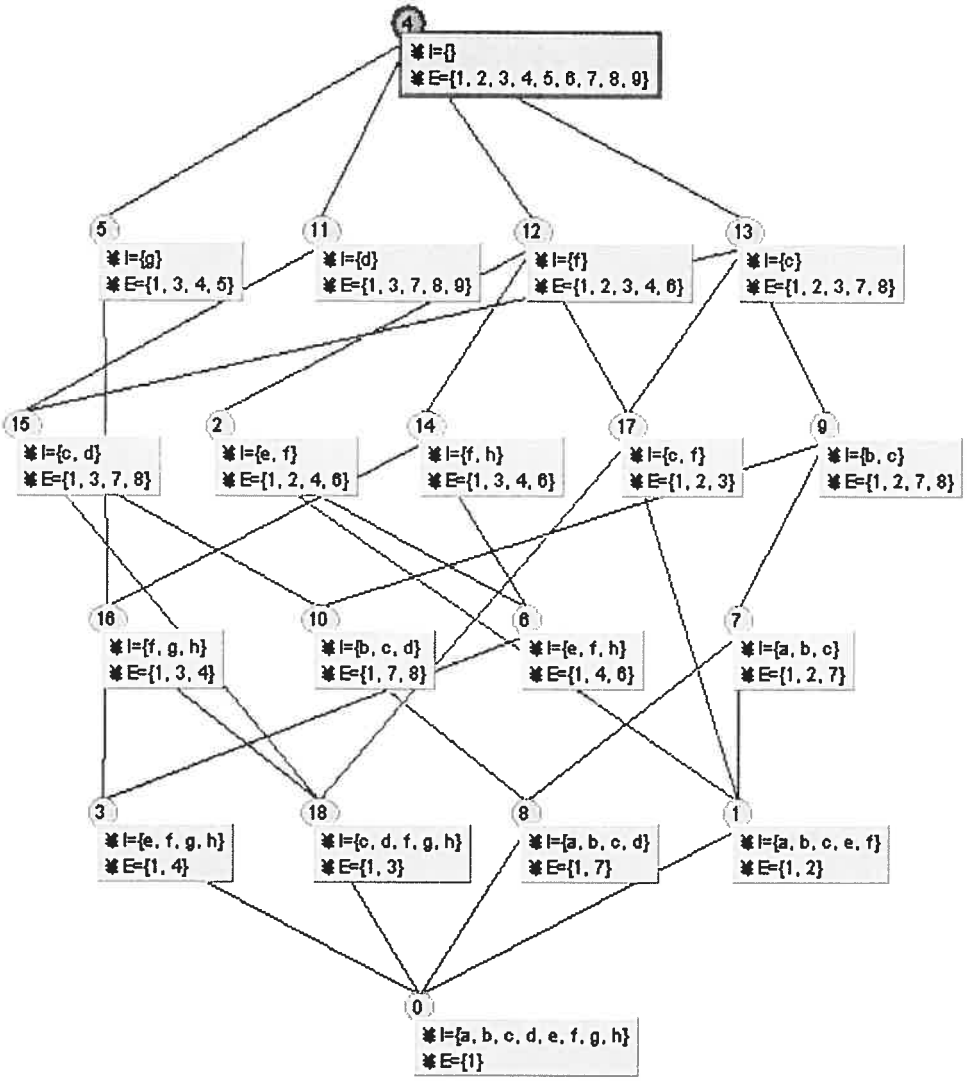

Figure 3.4.2 The Hasse diagram of the Galois lattice derived from K

An Incremental algorithm makes sense, since in many cases the data set is not static. New data is added to the set from time to time. In the context of information retrieval, we know that in many cases the collections of documents are also dynamically evolving. New documents will be added into the collections periodically. The construction of the lattice is a time consuming operation. So it is not acceptable to generate the lattice from scratch each time a new data is added to the data set. An Incremental algorithm

provides a way to generate new lattices from old ones and the new data, hence save the time and computational resources.

For example, to construct the lattice of a context with 100 objects and 1000 attributes, it takes about 2 seconds within Galicia. If we already have the lattice of the first 99 objects and the 1000 attributes, when we add the 100^{th} object, Galicia needs only several milliseconds to construct the new lattice.

3.5 Association rules generation

After we get all the frequent closed itemsets, generating association rules is straightforward. The algorithms are implemented in Galicia platform. We just give a brief description on the principle of rules generation [Z2000].

Given a frequent closed itemset X and *minconf*, rules of the form $Y \xrightarrow{p} X - Y$ are generated for all $Y \subset X, Y \notin \varnothing$, provided $p \geq minconf$.

Further research on association rules generating is beyond the scope of this study. We will not discuss them here.

Chapter 4 Query expansion using association rules

4.1 Applying association rules

In this chapter we will study how to apply association rules in query expansion. The process is shown in Figure 4.1.1. It can be divided into two main parts. The first is the association rules mining from a document collection. The second is using association rules for query expansion.

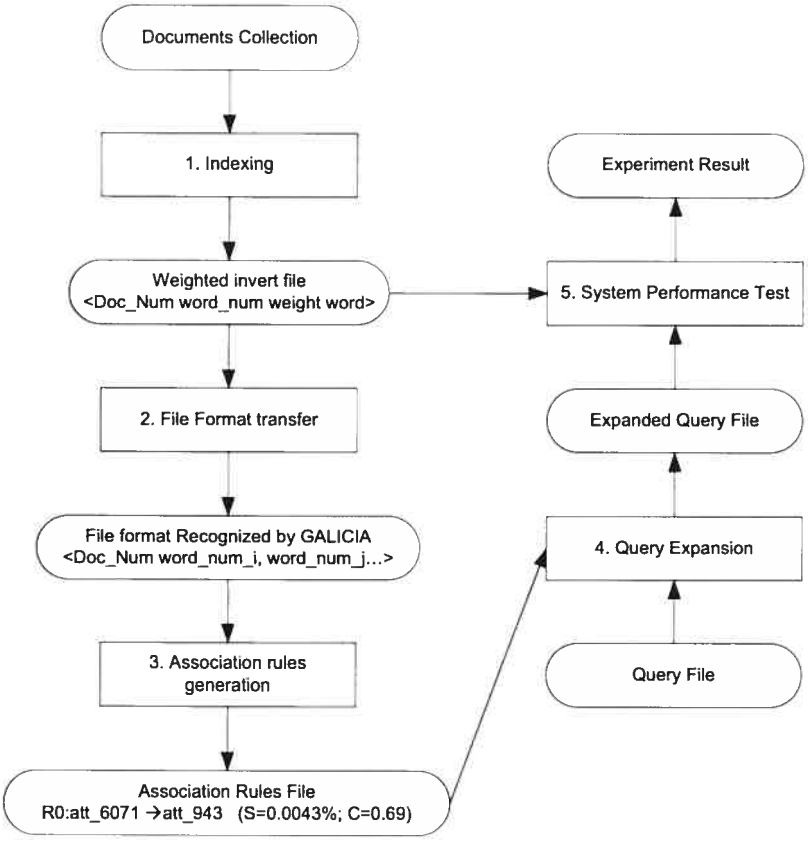

Figure 4.1.1 Experiment Process

4.1.1 Mining association rules from a document collection

Mining association rules form a document collection includes step 1 to step 3 in Figure 4.1.1.

Step 1: Using SMART system to index the document collection. This step results in the weighted inverted file and dictionary file.

Step 2 We need to transform the inverted file into the file format that can be recognized by Galicia system. We call it context file.

Step 3 Load the context file in Galicia, given the *minsupp* (minimal support) and *minconf* (minimal confidence), so that we can get all the association rules. They are recorded in a text file. Its format is like below:

Processing Base informative

Min confidence = 50% ; Min Support = 0.0035%

R0 : att_6071 --> att_943 *(S = 0.0043% ; C = 69%)*

R1 : att_2466 --> att_943 *(S = 0.0271% ; C = 77%)*

R2 : att_159 --> att_5501 *(S = 0.0096% ; C = 52%)*

R3 : att_8677 --> att_8820 *(S = 0.039% ; C = 60%)*

R4 : att_4121 --> att_7651 *(S = 0.0137% ; C =60%)*

R5 : att_4121 --> att_943 *(S = 0.0121% ; C = 53%)*

...

The first line tells us that the algorithm used to extract these association rules is *Base informative*. The second line tells us that in the calculation the minconf is 50% and minsupp is 0.0035%. The rest of the lines are rules. The prefix "att_" is added by the system and the number is the identifier of the term. *S* and *C* are the support and confidence of the rule, respectively.

4.1.2 Using association rules for query expansion

Step 4 When we get the association rules we can use them to expand the queries. Here we just show the strategy for query expansion. Later we give the detailed description. First, we get the collection of all the keywords in a query. Then, we find all the subsets of the keywords collection that can match the antecedent (left side) of the rules. Once we find a matching, the keywords in the consequent (right side) of the rule are added to the query. We repeat this process and it results in a new query set.

4.2 Algorithm for query expansion

4.2.1 Organization of association rules

A trie data structure is currently used to store sets of words over a finite alphabet [CM1995]. The trie provides a good trade-of between storage requirements and manipulation cost. In the basic form a trie is a tree where edges are letters from the alphabet. So each word corresponds to a unique path in the tree. The nodes corresponding to the end of a word are called terminal nodes, and the rest are called inner nodes.

We use the trie data structure to store all the association rules. Here we assign the words in the antecedent of a rule to the edges. The consequents of the rules are stored in the terminal nodes. The support and confidence of a rule are also stored in the terminal nodes. So the matching process is easy and quick. First, we go through the text file which contains the association rules and fill the trie with what we read. Then we get the keyword collection of a query. For each subset of the collection, we search it in the trie. When we encounter a terminal node, it means we find a matching rule. The words stored in the terminal nodes, which are the words in the consequent of the rule, will be

added to the query. The support and confidence value are also retrieved at the same time.

For example, we have a query with a set of terms: *{term1, term2, term3}*. When we expand the query, *{term1}, {term2}, {term3}, {term1, term2}, {term1, term3}, {term2, term3}* and *{term1, term2, term3}* are all used to match the rules. Each time we find a matching rule, the consequent of that rule will be inserted to the new query.

Here is an example of some rules stored in the trie data structure. Given the rules:

R0 : att_2 --> att_5 *(S = 0.0043% ; C = 0.69)*

R1 : att_2, att_3 --> att_6 *(S = 0.0271% ; C = 0.77)*

R2 : att_2,att_4 --> att_7 *(S = 0.0096% ; C = 0.52)*

R3 : att_2,att_9 --> att_8 *(S = 0.039% ; C = 0.6)*

R4 : att_4 --> att_1 *(S = 0.0137% ; C = 0.6)*

R5 : att_6,att_7,att_8 --> att_4, att_5 *(S = 0.0121% ; C = 0.53)*

Some steps of the process of constructing the trie to store these rules are shown in Figure 4.2.1

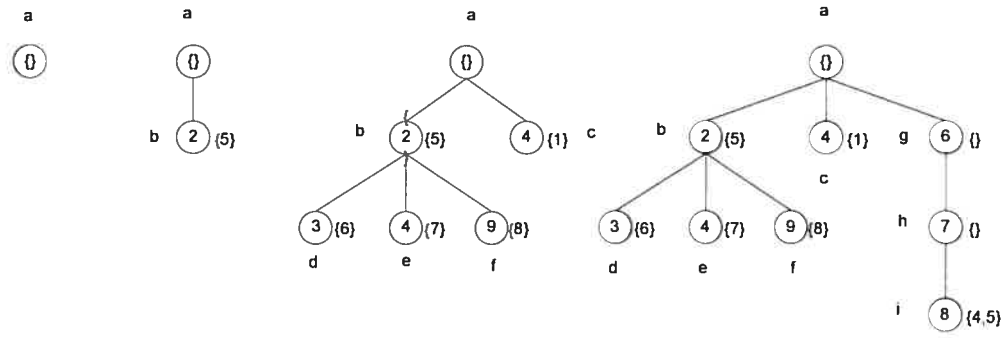

Figure 4.2.1: The process of constructing trie for rules

In Figure 4.2.1, the alphabets outside the node are identifiers of nodes. The numbers are identifiers of terms. Except the root node (node *a*), if there is no element in the pair of braces beside the node, the node is an inner node. Otherwise it is a terminal node. For example, nodes *b, c* and *d* are terminal and nodes *g* and *h* are inner. At first there is an empty node *a* as the root of trie. When we read rule *R0*, node *b* is inserted to the tree.

When we read rule *R1, R2, R3,* and *R4* nodes *c, d, e* and *f* are inserted to the tree one by one. When we read rule *R5*, three nodes (*g, h,* and *i*) are inserted to the tree.

The matching process is also very simple. For example, assume that *{att_2, att_4}* is a subset of terms in a query. First we read *att_2,* in the trie we will locate node *b.* Then we read *att_4* and reach node *e* through node *b.* Since node *b* is a terminal node and *att_7* is stored in this node, we add att_7 to the query. In constrast, if *{att_6, att_7}* is a subset of terms in a query, when we search in the trie we will reach node *h.* It is an inner node so we can not expand this query by these two terms.

4.2.2 Algorithm of query expansion

After we processed whole documents collection we got a set of association rules in the following form:

$term1, term2 \Rightarrow term3, term4$, with support value *sup* and confidence value *con.*

The interpretation of such rule is: When *term1* and *term2* are present in a document at the same time, *term3* and *term4* will also be present in that document with confidence *con*. So we can use such rules to expand the query. That is, if a query contains *term1* and *term2*, it is reasonable for us to add *term3* and *term4* to the query to achieve better search result.

Algorithms 4.2.1 and 4.2.2 describe the main steps of the query expansion process. Algorithm 4.2.1 is simple: it just uses a loop to see if a keyword in a query can match the first keyword of a rule. If there is a matching we call Algorithm 4.2.2 to continue the matching process.

1: **procedure** QUERYEXPAND(**In**: *qw* a collection of keywords, *we* a trie storing all rules, **Out**: *nqw* collection of new keywords that will be added to the query)
2: Local: *nqw* a collection of new keywords
3: **for all** $i \in qw$ **do**
4: **if** *i* matches a node *n* in the first level of *we* **then**

```
5:    EXPANSION(n, qw-i, nqw)
```

Algorithm 4.2.1

```
1: procedure EXPANSION(In: n a node in the trie we, rqw a collection of keywords, In/Out: nqw a collection of new keywords)
2: if n is a terminal node then
3:    put all keywords in n into nqw
4: for all i ∈ rqw do
5:    for all j ∈ cn {cn: the children of n} do
6:       if i matches j then
7:          EXPANSION(j, rqw-i, nqw)
```

Algorithm 4.2.2

Algorithm 4.2.2 has two nested loops and is itself recursive. Combined with algorithm 4.2.1, this algorithm ensures that every matching rule can be found.

Procedure QUERYEXPAND has two inputs. The first *'qw'* is a collection of all keywords in a query. The second *'we'* is a trie which stores all association rules. The output *'nqw'* is collection of all the new keywords that will be used to expand the query. Note that the keywords in *'qw'* are ordered by their *word_num* as well as the keywords in subset of *'qw'*. This order makes it easy to enumerate the subset of *'qw'* and procedure EXPANSION ensures no subset that can match a rule is omitted or repeated. The loop in Procedure QUERYEXPAND is used to match first keyword of every subset. If it finds a matched node, procedure EXPANSION will go through the trie path via the first matched node to find the next matched node of the consequent keyword in the subset. When a terminal node is matched, it means we find a matched rule and all the keywords stored in the terminal node will added to collection *'nqw'*. A recursion

ending by the matching of an inner node means no rule matched that subset and hence the subset can not be expanded.

4.3 Weight of new keywords in query

Now we can get the new keywords from the original query. There is another important task to be done when we add these keywords into the query file: We need to assign appropriate weights to the new keywords. Unfortunately there is no principle that can be followed in doing that. We used several different ways to calculate the new weight and tried to find a proper one to apply uniformly in the system.

Intuitively, we think the new weight must rely on the composition of the rule since the new keyword is inferred from its premise. There are several factors in a rule that may affect the new weight.

The first one is the weight of the keywords from the premise of the rule, that is to say, the weight of the keywords in the original query. Here we consider two situations:

1. Simple case: A new keyword is inferred from only one rule. That is,

   $r_1 : \{a_1,...,a_n\} \Rightarrow \{t_1...\}$. Here $a_{1,...}a_n$ are keywords in the original query. We can set the weight of the new keyword t_1 either by the maximum weight of $a_{1,...}a_n$ or their average weight.

2. Complex case: Several rules infer the same new keyword. That is,

   $r_1 : \{a_1,...,a_n\} \Rightarrow \{t_1...\}$

   ...

   $r_i : \{m_1,...,m_k\} \Rightarrow \{t_1...\}$

   In this case, we separate the calculation of the weight of the new keyword into two steps. First, for each rule we calculate the weight as in the simple case. Second, we can set the weight of the new keyword either by the maximum result of the rules or their average result.

Combine the two cases we get four basic strategies to compute the weight of new keyword. Assume:

$r_1 : \{a_1, ..., a_n\} \Rightarrow \{t_1 ...\}$

...

$r_i : \{m_1, ..., m_k\} \Rightarrow \{t_1 ...\}$

w_{nt1} is the weight of the new keyword t_1

$r_1 ... r_i$ are rules that contain keyword t_1 in their consequent.

$a_1 ... a_n$ and $m_1 ... m_k$ are keywords in the antecedent of each rule respectively.

$w_{ra1}, ... w_{ran}$ and $w_{rm1}, ... w_{rmk}$ are weights of each keyword in the query.

The four strategies are listed in Table 4.3.1

	Weight Computation
Strategy 1	$w_{nt1} = \max[\max(w_{ra1}, ... w_{ran}), ... \max(w_{rm1}, ... w_{rmk})]$
Strategy 2	$w_{nt1} = \max[average(w_{ra1}, ... w_{ran}), ... average(w_{rm1}, ... w_{rmk})]$
Strategy 3	$w_{nt1} = average[\max(w_{ra1}, ... w_{ran}), ... \max(w_{rm1}, ... w_{rmk})]$
Strategy 4	$w_{nt1} = average[average(w_{ra1}, ... w_{ran}), ... average(w_{rm1}, ... w_{rmk})]$

Table 4.3.1 Basic Strategies for weight computation

The next two factors are the support and the confidence of the rule. When we consider these factors we can combine them with the four basic strategies. According to these strategies we designed our experiments whick will be analysed in the next chapter.

Chapter 5 Experimental Results

5.1 Experimental environment

The SMART system and The Galicia platform are used in the experiments. They are introduced in Chapter 2 and Chapter 3. The implementations of the algorithms for query expansion are presented in Chapter 4. Here we give a brief description of the CACM collection which is used in our experiments.

The CACM collection is a collection of 3204 titles and abstracts from the journal CACM. Documents in the collection are separated by sections. Each section is marked by the letters in bold.

.I introduces the start of a new document.

.T introduces the section 'title'.

.W introduces the section 'abstract'.

.B introduces when the article is published.

.A introduces the author of the document.

.N identify when the document was added to the collection.

.X introduces the section 'reference'.

The documents in CACM are like the example below:

.I *1025*

.T

A Method of Syntax-Checking ALGOL 60

.W

A syntax checker was designed based on the syntax of ALGOL as described in the ALGOL 60 Report [Communications of the ACM, May, 1960]. Since the definition of the elements of the language is recursive it seemed most desirable to design the syntax checker as a set of mutually recursive processors tied together by subroutines which

perform certain bookkeeping functions. Because of the recursive nature of the language and of the syntax checker the problem of recovery after an error required much attention.

A method was devised which permits most programs to be checked completely despite errors.

.B

CACM August, 1964

.A

Lietzke, M. P.

.N

CA640805 JB March 9, 1978 7:24 PM

.X

1025 5 1025

1025 5 1025

1025 5 1025

...

There are 52 queries with relevance judgments. The query file has already been prepared by the CACM collection and SMART system so we need not to do query pre-process. The parameters of the CACM collection and the experiments are listed on Table 5.1.1.

The Galicia platform is a research environment, to make it general, it sacrifices the efficiency in a way. If the collection is too large it will cost too much time to get the association rules set. This is why we use CACM collection in our experiment instead of the TREC collection since CACM is relatively small.

Document collection	CACM
Total number of documents	3204
Total number of keywords	8994
Total number of queries	52
Value of minsupp	0.004%
Total concepts	17423
Total rules	7760
Total keywords in the query set	671
Total keywords after expansion	2056

Table 5.1.1 Parameters of the dataset used in the experiments

5.2 Query expansion using basic strategies

Experiment 1:

To find a best strategy to decide the weight of new keyword added to the query we carried out several experiments.

	Average precision for all points				
	Original	Strategy 1	Strategy 2	Strategy 3	Strategy 4
11-pt Avg:	0.2341	0.1644	0.1652	0.1847	0.1845
% Change:		-29.8	-29.4	-21.1	-20.7

Table 5.1.2 Result of experiment 1

In experiment 1 we evaluated the SMART system 5 times using all the extracted rules. The first time we did not employ query expansion. Strategies 1 to 4 are used for query expansion in the next four times respectively. 11-point average precision is the most

important metric in our experiment. Its change represents the relative change of the system performance.

The experimental results are shown in Table 5.1.2. There are five columns of data in the table. The first column is the original system performance evaluation without query expansion. The next four columns represent system performance evaluations for the four basic strategies applied to query expansion in the experiments.

The Precision-Recall graph of the experiment is shown in Figure 5.1.1

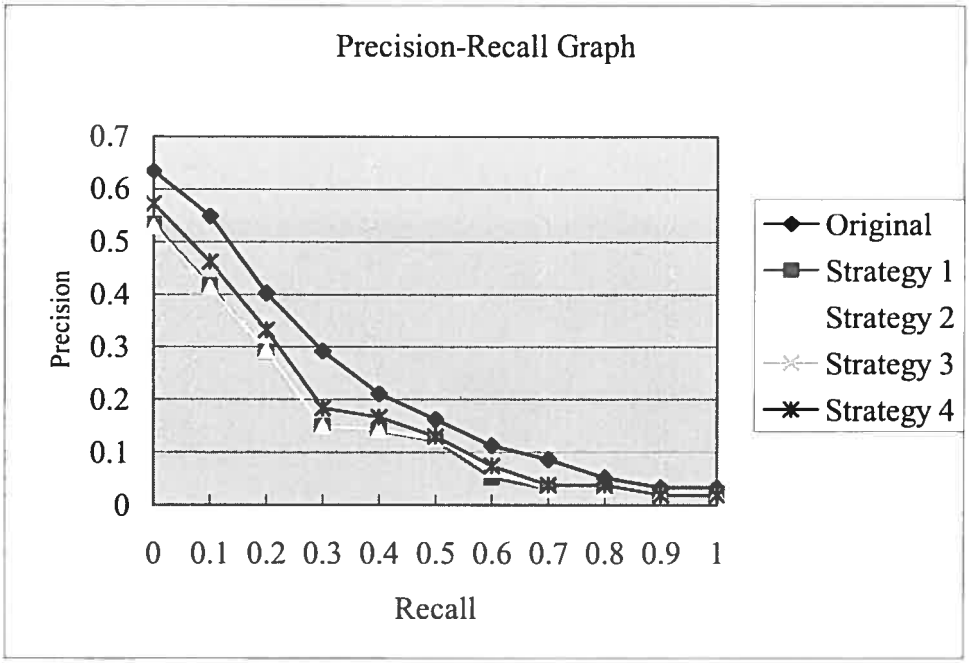

Figure 5.1.1: Precision-Recall graph of the experiment 1

Figure 5.1.1 is the Precision-Recall graph of experiment 1. We observe that after we employ the strategies to the process of query expansion, all of the results are barely satisfactory. We think the reason is the weight of a new keyword can not be as high as

that of the original ones. We also observe that the strategies using average weight are somewhat better than those using maximum weight.

5.3 Query expansion considering confidence

Experiment 2

In this experiment when we infer a new keyword from a rule we multiply the weight of the original keyword by the confidence of the rule and then apply the same strategies used in experiment 1. For strategy 1, this is:

$$w_{nt1} = \max[\max(w_{ra1},...w_{ran})\times con_{r1},...\max(w_{rm1},...w_{rmk})\times con_{ri}]$$

where

$con_{r1}...con_{ri}$ are the confidence values of the rules $r_1...r_i$, respectively.

For the other three strategies the modification are similar.

The result is shown in Table 5.1.3

	Average precision for all points				
	Original	Strategy 1	Strategy 2	Strategy 3	Strategy 4
11-pt Avg:	0.2341	0.2001	0.2006	0.2161	0.2159
% Change:		-14.5	-14.3	-7.7	-7.8

Table 5.1.3 Result of Experiment 2

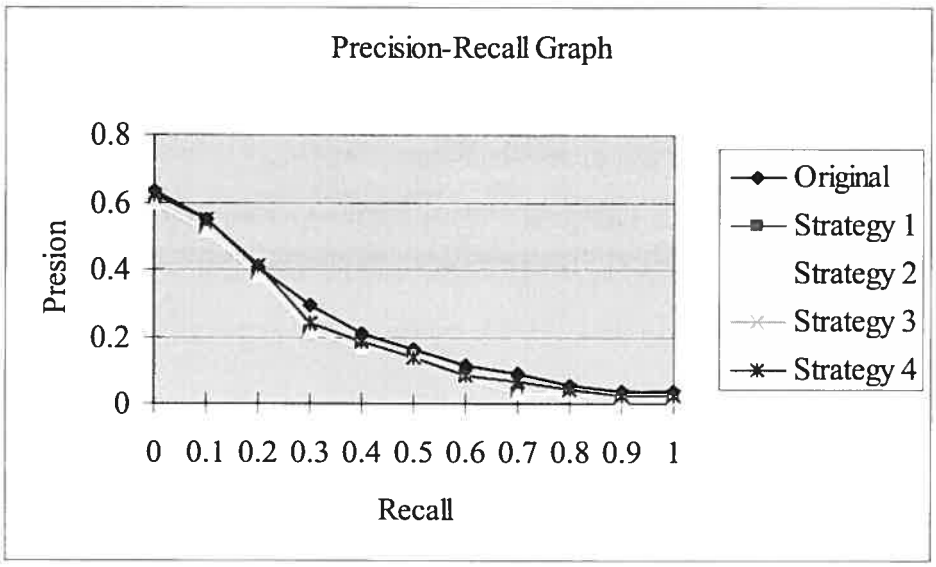

Figure 5.1.2 Precision-Recall Graph of the experiment 2

As we can see in Table 5.1.3 and Figure 5.1.2 the result is much better than the result in experiment 1 but it is still unsatisfactory. We need to decrease the weight value further.

5.4 Query expansion considering support

Experiment 3

Based on experiment 2, we introduce the support value of the rules to the calculation of the weight (i.e. further multiple support value with the weight we got in experiment 2). We normalize the support value of all rules before we apply it. The formula of normalization is:

$$Supp_{new} = \frac{Supp - Supp_{min} + 0.001}{Supp_{max} - Supp_{min} + 0.001} \times A + B$$

Here $Supp$ is the support value of a rule. $Supp_{max}$ and $Supp_{min}$ are the maximum and minimum support value of all the rules. $Supp_{new}$ is the support value of rule after normalization. We add 0.001 to both numerator and denominator to make all the values positive. This value is determined empirically. The values of A and B ($1 \geq A > 0, B \geq 0$), can make the support distribute in different interval. For strategy 1, this becomes:

$$w_{nt1} = \max[\max(w_{ra1},...w_{ran}) \times con_{r1} \times supp_{r1},...\max(w_{rm1},...w_{rmk}) \times con_{ri} \times supp_{ri}]$$

where

$con_{r1}...con_{ri}$ are confidence values of the rules $r_1...r_i$ respectively.

$supp_{r1}...supp_{ri}$ are support values of the rules $r_1...r_i$ respectively.

For the other three strategies the modification are similar.

Experiment 3.1

In experiment 3.1, we set *A=1, B=0*. The result is shown in Table 5.1.4

	Average precision for all points				
	Original	Strategy 1	Strategy 2	Strategy 3	Strategy 4
11-pt Avg:	0.2341	0.2363	0.2363	0.2355	0.2355
% Change:		1.0	1.0	0.6	0.6

Table 5.1.4 Result of Experiment 3.1

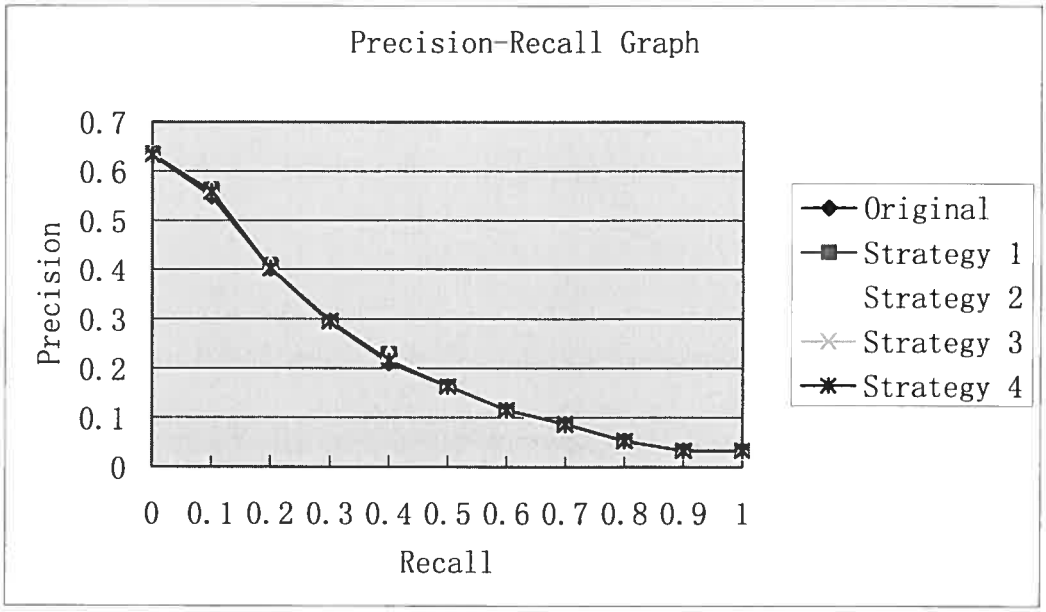

Figure 5.1.3 Precision-Recall Graph of the experiment 3.1

In this experiment the system performance is further improved. And now we can see that query expansion has improved the whole system performance.

We compare the query files in experiment 2 and experiment 3 as listed in Table 5.1.5. The keywords above the dash line are original ones. From the table we can see that the weight of the new keywords should be much lower than that of the original keywords. Note in this experiment the average weight strategy did not show advantages as they did in experiment 1 and 2. We think it means that the weight value is too small in this experiment.

Query 3 in Experiment 2 (after expansion)					Query 3 in Experiment 3.1 (after expansion)				
3	0	3352	0.34318	compiler	3	0	3352	0.34318	compiler
3	0	3874	0.43674	construction	3	0	3874	0.43674	construction
3	0	7295	0.53726	multi	3	0	7295	0.53726	multi
3	0	7741	0.59144	intermediate	3	0	7741	0.59144	intermediate
3	0	15587	0.23029	language	3	0	15587	0.23029	language
---					---				
3	0	17482	0.13395	data	3	0	17482	0.01443	data
3	0	10420	0.16351	grammar	3	0	10420	0.01178	grammar
3	0	16508	0.11745	structure	3	0	16508	0.00502	structure
3	0	5152	0.13717	program	3	0	5152	0.01322	program
3	0	17338	0.15217	system	3	0	17338	0.01292	system
3	0	13148	0.17502	level	3	0	13148	0.03232	level
3	0	4967	0.17271	1	3	0	4967	0.01128	1
3	0	18563	0.18423	paper	3	0	18563	0.01026	paper
3	0	19661	0.20496	problem	3	0	19661	0.01200	problem
3	0	2981	0.17041	input	3	0	2981	0.01957	input
3	0	18900	0.12436	general	3	0	18900	0.00896	general
3	0	5376	0.13798	programming	3	0	5376	0.01331	programming
3	0	18948	0.1819291	2	3	0	18948	0.01188	2

Table 5.1.5 Query file (Partial) after expansion

Experiment 3.2

In experiment 3.2 we tried different values of *A* and *B* in the normalization formula. Considering the results of experiment 1 and experiment 2, we use the result of strategy 4 in experiment 3.1 as the base for comparison. The result is shown in Table 5.1.6.

System Performance change		B					
		0	0.1	0.2	0.3	0.4	0.5
A	0.5	0.0	1.9	1.1	0.5	1.6	-0.2
	0.6	-0.9	1.9	1.2	0.5	1.8	-0.9
	0.7	0.2	1.8	1.2	0.0	1.6	-0.9
	0.8	0.2	1.8	1.2	0.1	1.6	-1.0
	0.9	0.3	1.8	1.2	0.1	1.2	-1.0
	1.0	0.6	1.8	1.2	-0.1	1.2	-0.9

Table 5.1.6 Result of experiment 3.2

From the result we found that when *A=0.6, B=0.1* (i.e. all support values distribute in [0.1, 0.6]) the system performance can be best improved.

5.5 The impact of the threshold values

Experiment 4

There are 671 keywords in total in the original query file. In the first three experiments we have 2056 keywords in query file after query expansion. We think that too much new keywords may introduce much more noise. So we tried to reduce the number of new keywords.

Experiment 4.1

In experiment 4.1 we give a threshold value of rule confidence (i.e. we ignore those rules with low confidence). We use the result of strategy 4 in experiment 3.2 as the base for comparison (*A=0.6, B=0.1*).

	Lower threshold of rule confidence							
	0.5	0.55	0.6	0.65	0.7	0.75	0.8	0.9
System Performance change	1.9	0.6	0.9	1.5	1.5	1.1	0.5	-0.6

Table 5.1.7 Result of experiment 4.1

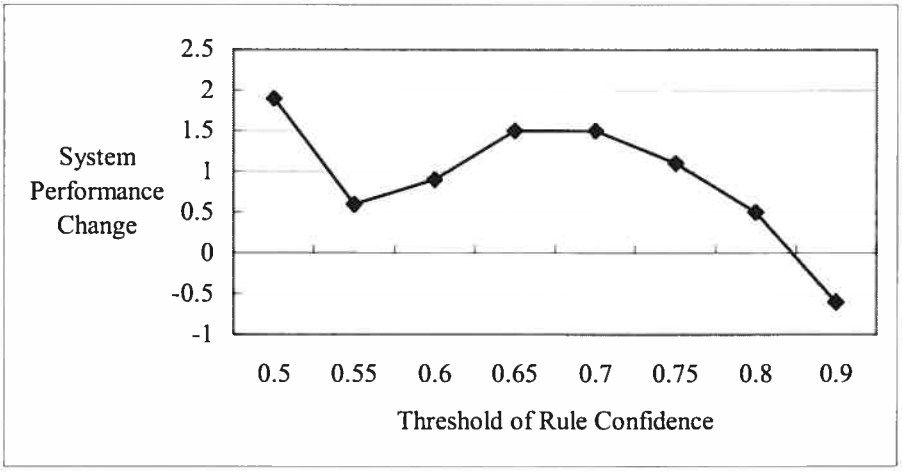

Figure 5.1.4 System Performance Change of Experiment 4.1

We can see from Figure 5.1.4 that the system performance became worse as the threshold of rule confidence increased. It seems that rules with lower confidence have more contribution to the system performance improvement. This is reasonable since generally a rule with high confidence value will have a low support value.

Experiment 4.2

In experiment 4.2 we ignore those rules with high confidence. Other parameters are the same as in experiment 4.1. The result is shown in Table 5.1.8.

	Upper threshold of rule confidence						
	0.9	0.8	0.75	0.7	0.65	0.6	0.55
System Performance change	1.9	1.6	1.7	1.6	1.5	1.0	0.6

Table 5.1.8 Result of experiment 4.2

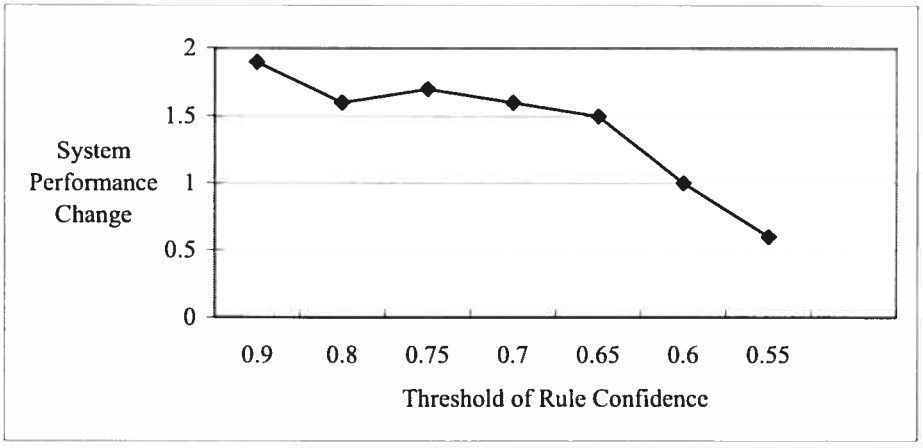

Figure 5.1.5 System Performance Change in Experiment 4.2

The results of experiment 4.1 and 4.2 show us that it's not beneficial to use restrictions on rule confidence to reduce the number of new keywords. This technique can not improve the system performance.

Experiment 4.3

The problem of experiments 4.1 and 4.2 is that the values of the support are fixed. So in experiment 4.3 we fix only the value of B as 0.1 and make the value of A vary from 0.5 to 1. At the same time we make the threshold value of the confidence vary from 0.5 to 0.9. The result is shown in Table 5.1.9.

System Performance		Lower threshold of rule confidence							
		0.5	0.55	0.6	0.65	0.7	0.75	0.8	0.9
A	0.5	1.9	0.5	1.0	1.5	1.5	1.2	0.5	-0.6
	0.6	1.9	0.6	0.9	1.5	1.5	1.1	0.5	-0.6
	0.7	1.8	0.6	1.0	1.5	1.7	1.1	0.4	-0.6
	0.8	1.8	0.5	1.0	1.4	1.7	1.1	0.5	-0.6
	0.9	1.8	0.6	0.4	1.4	1.7	1.0	0.5	-0.6
	1.0	1.8	0.6	0.4	0.8	1.0	1.0	0.5	-0.6

Table 5.1.9 Result of experiment 4.3

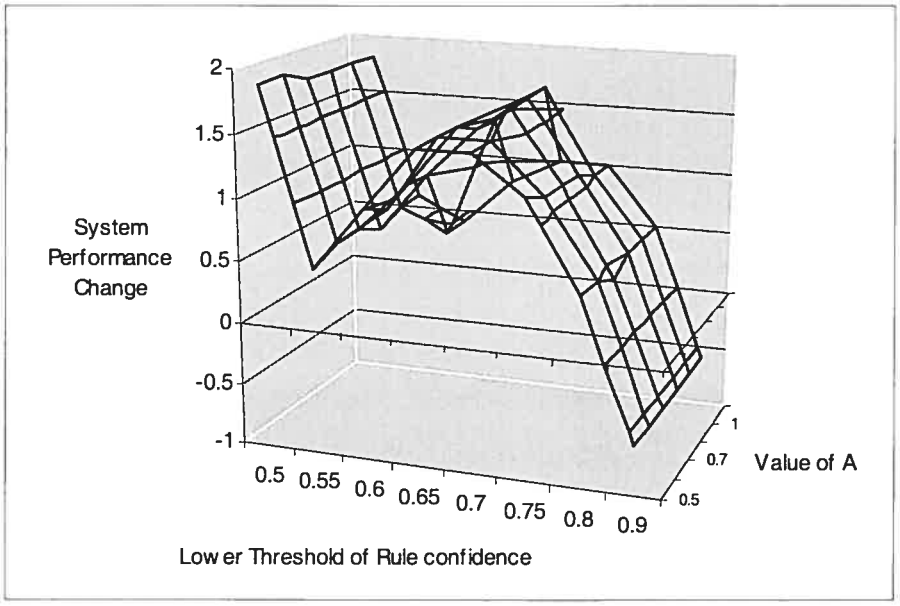

Figure 5.1.6 System Performance Change in Experiment 4.3

Comparing the result with experiment 4.1 we see that no other combination of confidence and support can get better result.

Experiment 5

In this experiment we ignored those keywords in the original query with low weights when inducing new keywords from association rules. In this experiment we give a threshold value of the weight. When we find a matching rule, we will ignore this rule if a keyword that appears in antecedent of the rule has a low weight (i.e. lower than the threshold value). We use the result of strategy 4 in experiment 4.1 as the base for comparison. The threshold is changed from 0.1 to 0.26.

Lower Threshold value of Weight	System Performance change	Threshold value of Weight	System Performance change
0.0	1.9	0.18	1.1
0.10	1.7	0.19	1.0
0.11	1.7	0.20	1.0
0.12	1.8	0.21	0.4
0.13	1.2	0.22	0.6
0.14	1.2	0.23	-0.2
0.15	1.2	0.24	0.3
0.16	1.2	0.25	0.1
0.17	1.2	0.26	-0.3

Table 5.1.10 Result of Experiment 5

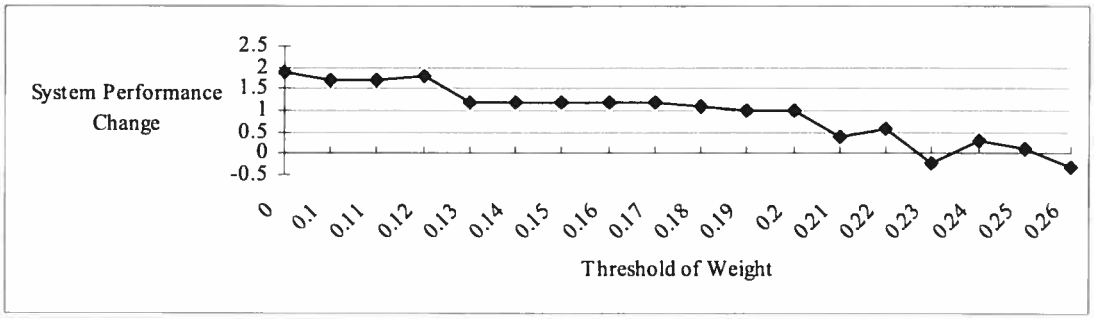

Figure 5.1.7 System Performance change of Experiment 5

As we can see in Table 5.1.10 and Figure 5.1.7, there's no noticeable improvement. When the threshold is too high it will even make the system perform worse.

Experiment 6

In this experiment we try another way to reduce the total number of new keywords. We give a threshold value for the weight and if a new keywords appears whose weight value is lower than the threshold we do not add it to the query. We still use the result of strategy 4 in experiment 3.1 as the basis for comparison. According to the new query file in experiment 3 as shown in Table 5.1.4 we choose the threshold range from 0.008 to 0.030. The result is shown in Table 5.1.11 and Figure 5.1.8.

Lower Threshold value of Weight of new keyword	System Performance change	Threshold value of Weight	System Performance change
0.0	1.9	0.015	0.6
0.008	1.8	0.016	0.4
0.009	1.8	0.017	0.5
0.010	1.9	0.018	0.3
0.011	1.3	0.019	1.1
0.012	0.5	0.020	1.1
0.013	0.4	0.025	0.7
0.014	0.6	0.030	0.4

Table 5.1.11 Result of Experiment 6

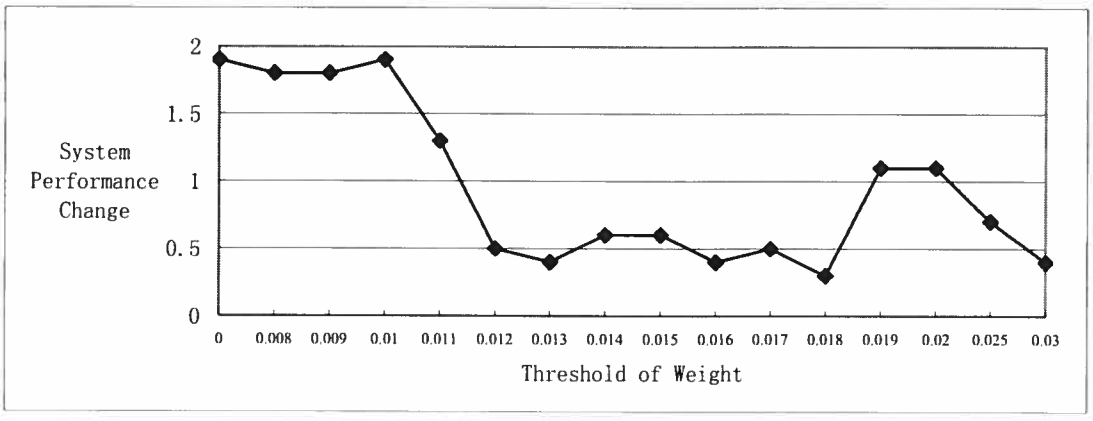

Figure 5.1.8 System Performance change of Experiment 6

From the result we know that the use of a unified threshold value for all queries is not reasonable. It just makes the system perform worse.

Experiment 7

In this experiment we try to reduce the total number of new keywords by giving threshold value of the support.

Experiment 7.1

In experiment 7.1 we give a threshold value of rule support (i.e. we ignore those rules with low support). We use the result of strategy 4 in experiment 3.2 as the base for comparison (*A=0.6, B=0.1, threshold of confidence is 0.5*). The result is shown in Table 5.1.12 and Figure 5.1.9.

	Lower threshold of rule support (%)							
	0.0040	0.0045	0.0050	0.0055	0.0060	0.0065	0.007	0.0075
System Performance change	1.9	1.4	0.6	1.4	0.9	0.9	0.4	0.4

Table 5.1.12 Result of Experiment 7.1

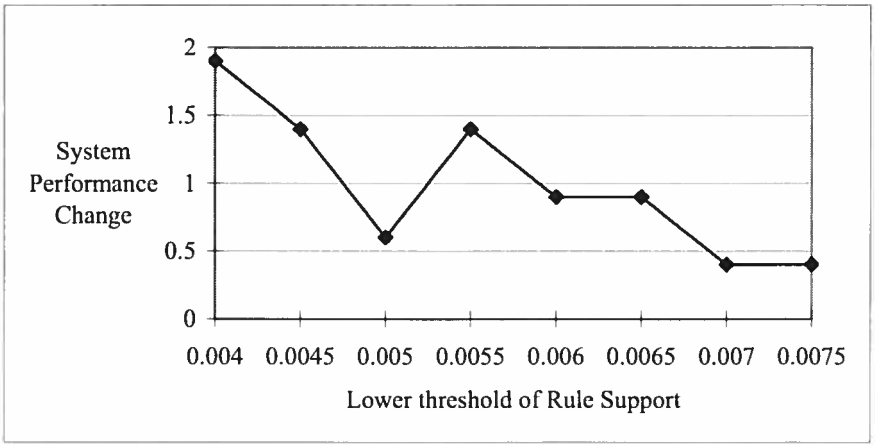

Figure 5.1.9: System Performance change of Experiment 7.1

Experiment 7.2

In experiment 7.2 we ignore those rules with high support. Other parameters are the same as in experiment 7.1. The result is shown in Table 5.1.13 and Figure 5.1.10.

	Upper threshold of rule support							
	0.0090	0.0085	0.0080	0.0075	0.0070	0.0065	0.0060	0.0055
System Performance change	-0.1	0.7	1.4	0.6	0.9	0.9	1.5	1.1

Table 5.1.13 Result of Experiment 7.2

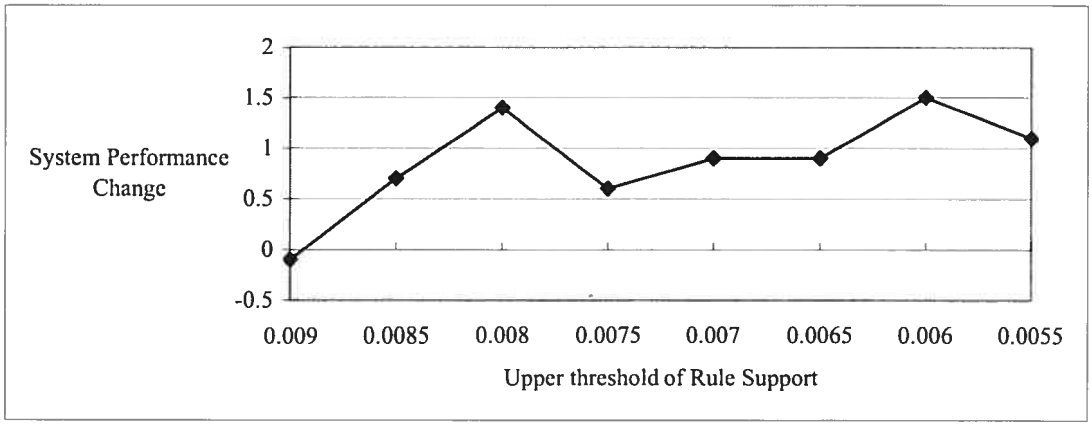

Figure 5.1.10 System Performance change of Experiment 7.2

Experiment 7.3

In experiment 7.1 and 7.2 the threshold value of confidence is fixed to 0.5. In experiment 7.3 we will vary the threshold value of confidence from 0.5 to 0.9. The result is shown in Table 5.1.14.

System Performance change		Lower threshold of rule confidence							
		0.5	0.55	0.6	0.65	0.7	0.75	0.8	0.9
Lower threshold of rule support	0.004	1.9	0.6	0.9	1.5	1.5	1.1	0.5	-0.6
	0.0045	1.7	0.6	0.4	1	0.8	0.5	-0.7	-1.1
	0.005	0.8	0.6	0.9	1.1	1.1	-0.1	-0.9	-0.9
	0.0055	1.5	1.1	1.1	0.8	1.1	-0.1	-1	-0.9
	0.006	1.1	0.9	0.4	0.6	0.6	-0.2	-0.1	-0.2
	0.0065	1.1	0.7	0.4	0.3	0.6	-0.1	0	-0.2
	0.007	-0.2	-0.2	0.2	0.2	-0.5	-0.9	-0.7	-0.2
	0.0075	0	0	-0.4	-0.4	-0.5	-0.9	0	-0.2

Table 5.1.14 Result of Experiment 7.3

The result of experiment 7 shows us that it is not reasonable to use restriction on rule support to reduce the number of new keywords. This technique can not improve the system performance.

5.6 Interactive Query expansion

Experiment 8

In this experiment we actually use interactive query expansion. It needs the user's intervention during the process of expansion. That is say after we get the new query file we look through it and remove some new keywords according to our own judgment. The purpose of this experiment is to see the possible upper bound when association rules are used in query expansion.

Query 5 Original keywords					Query 5 Original keywords				
5	0	80	0.26938	manager	5	0	9655	0.21901	editing
5	0	502	0.14039	user	5	0	10134	0.23277	essential
5	0	545	0.41144	interface	5	0	12200	0.33835	window
5	0	2673	0.21418		5	0	12513	0.20582	human
effectiveness					5	0	13461	0.17278	efficiency
5	0	4183	0.31976	satisfaction	5	0	15418	0.20939	design
5	0	4346	0.21573	issue	5	0	15605	0.22074	view
5	0	5912	0.13686	implementation	5	0	17481	0.20456	command
5	0	6117	0.19263	improvement	5	0	18563	0.09073	paper
5	0	7551	0.23760	interpreter					

Table 5.1.15 Keywords in Query 5 before expansion

5	0	17482	0.01534	data	5	0	19661	0.01224	problem
5	0	5152	0.01179	program	5	0	18900	0.01205	general
5	0	1811	0.01391	computer	5	0	1090	0.01131	time
5	0	17338	0.01889	system	5	0	5376	0.01633	programming
5	0	12266	0.00834	algorithm	5	0	18948	0.00760	2
5	0	15587	0.01504	language	5	0	12568	0.01623	text
5	0	4967	0.00784	1					

Table 5.1.16 New Keywords in Query 5 after expansion

For example, in Table 5.1.16 there are 13 new keywords in query 5 after the expansion. We think keyword '1' and '2' are not reasonable so we remove it from the query. Keywords 'problem', 'algorithm' and 'system' are too general in CACM collection. They also need to be removed.

We go this way all through the query file and get a new query file. Using this file, we test the system performance. According to our experiment, the system performance improvement is between 0.5 and 1.1 compared to the query without expansion.

This result shows that even used in an interactive way, the association rules can not make a larger improvement in IR effectiveness.

5.7 Experimental results analysis

From the experimental results we can conclude that the assignment of weights for the new keywords is a crucial aspect in query expansion. Different strategies for this task may lead to totally different results. Unfortunately there's no principle to guide us. In our experiments we tried many possibilities in the search for a good strategy to assign weight. The system performance improves slightly in some experiments. According to the results, the application of association rules in query expansion did not bring the improvement that we would expect. But we do not think it proves that we made efforts in the wrong direction. In fact, several factors may affect the result.

The first is the number of rules. As we can see in table 5.1.2 we use very low support value (only 0.004%) to generate rules and still got a large set of rules (7760). In the CACM collection there are 52 queries and 671 keywords. After applying the association rules, we obtain 2036 keywords. The average size of a new query is more than three times of the original one. Inevitable, this introduces noise to the query and it is difficult to reduce such noise. We think we should introduce some other analysis to the query process in order to filter the association rules.

The second factor is the document collection itself. CACM is a small collection and most documents in this collection are very short. Thus the weight of single keyword may have significant impact on the query result. This makes it more difficult to assign weight to the new keywords. Furthermore, the documents in this collection focus on only few fields (namely, all about computer science). This makes some frequent keywords present in almost every document as shown by Experiment 8. Since these frequent keywords do not belong to the stoplist, they will not be discarded in the indexing process. Also, since they are very frequent, they have a high probability to co-occur with other keywords. So, they may be present in many rules. After the expansion process, they may appear in every query, and may have no contribution to the precision but just introduce noise. As we observe in experiment 8, it is difficult to discard these words manually. It may be an important task of further research to find the guidelines to do this automatically.

The third factor is the rule extraction process. In fact, the document-term matrix used for rule extraction is treated as a binary value matrix in the process of rule generation. This means we only consider whether a term is present in a document or not. But actually these terms are weighted and the matrix should be a multi-valued. In the current approach it is not reasonable to calculate the co-occurrence of a term with high weight and a term with low weight. So we may scale the context to avoid such problem. Scaling aims to transform multi-valued matrix to single value matrix. (See appendix A.2 for the concept of scaling). Although scaling is already implemented in the Galicia platform, it introduces other problems. A problem is that after scaling a term may generate several binary features which correspond to different weights of that term in different documents. This makes it difficult to combine rule generation with the indexing process and it also makes the indexing process complex. The second problem

is after scaling the size of the context may grow quickly, which may lead to serious computational problems.

The fourth factor is the co-occurrence of terms. At present we consider that two terms co-occur if they appear in the same document. Actually, we may consider the distance of the two terms in the text as a factor. For example, if these two words are not present in the same paragraph, we should not consider them as co-occuring. The selection of the distance is also a potential research field.

The last possible reason is that, as both the document and the queries are in a very specific area (computer science), the queries may be already well expressed, and they do not need to be further expanded. It would be interesting to test query expansion using association rules on a different and non-specialized collection, in which the queries may be less well expressed (for example, a TREC collection).

Chapter 6 Conclusion and future work

6.1 Conclusion

The goal of this thesis is to apply association rules to query expansion in information retrieval. To generate association rules between terms a Galois lattice based-method is used. The key problem here is to efficiently generate the iceberg lattice. We used two algorithms for this purpose. Bordat algorithm is a classic top-down batch algorithm. We modified it so it can generate iceberg lattice according to the given support value. We also developed an incremental algorithm to construct the iceberg lattice. The increment can be either object-wise or attribute-wise. It makes sense to use an incremental algorithm since in many cases, especially in the context of information retrieval, the data set is dynamic. New documents will be fed into the collections periodically. An incremental algorithm provides a way to generate new lattice from the old lattice and the new data. Hence, it saves both time and computational resources.

When applying association rules to query expansion, the main problem is the assignment of weights to the new keywords that will be added to the queries. We have followed several alternative tracks and we did some experiments in a search for a good strategy to assign weights. The system performance improved slightly in some cases. With respect to our experiment results, we may conclude that the association rules can help query expansion to some extent. However, in order to take full advantage of the data mining mechanisms, a set of problems need to be solved. In particular, more effective weight assignment mechanism should be designed.

6.2 Future work

We have already pointed out some future research avenues in section 4.4. These may be summarized into five axes:

1. Statistical document analysis. The analysis should be carried out on two levels. First, one should analyze the terms in a single document to find more information about the ways those terms are related, such as the distance between them. Second we should take the whole collection into consideration to extract the global relationships between those terms. This analysis focuses on statistical information and does not consider the semantic aspects of the term-document or term-term relations.
2. The semantic analysis of the association rules. We think this is important to reduce the noise introduce by the rules. A considerable challenge lays in the automation of the analysis.
3. Indexing process. As we mentioned above if we scale the multi-valued document-term context, the frequency of a term in a document can be integrated into the mining process in a natural way. Thus the weight assignment could be benefit from this valuable information which is currently missing. However, the indexing process may require substantial modifications to accommodate the new scaling process.
4. The expansion process may be combined with pseudo-relevance feedback. In the process of pseudo-relevance feedback, the system assumes the top n documents in the first search result as relevant to the query. Then the system takes the terms from these documents to expand the query. We may apply association rules on the new query generated by the pseudo-relevance feedback process.
5. Finally, it would be interesting to experiment association rule extraction on a larger document collection. In order to do this, the complexity of the rule extraction process should be greatly reduced.

Reference

[AIS1993] Rakesh Agrawal, Tomasz Imielinski, and Arun Swami

Mining association rules between sets of items in large databases.

In Proceedings of the 1993 International Conference on Management of Data (SIGMOD 93), 1993, pp. 207-216.

[AIS1994] Rakesh Agrawal, Tomasz Imielinski and Ramakrishnan Srikant

Fast algorithms for mining association rules in large databases.

In Proceedings of the 20th International Conference on Very Large Databases (VLDB'94), Santiago, Chile, 1994, pp.487-499.

[B1986] Bordat, J.

Calcul pratique du treillis de Galois d'une correspondance.

Math. Sci.Hum. 96:31–47, 1986

[BR1999]Ricardo Baeza-Yates, Berthier Ribeiro-Neto

Modern information retrieval

ACM press, 1999

[CM1995] Richard H. Connelly and F. Lockwood Morris

A Generalization of the Trie Data Structure,

Math. Struct. in Comp. Science 5 ,1995, pp.381-418

[CY92] Crouch, C.J., Yong, B.,

Experiments in automatic statistical thesaurus construction,

SIGIR'92, 15th Int. ACM/SIGIR Conf. on R&D in Information Retrieval, Copenhagen, Denmark, June 1992, pp.77-87.

[G1971]G. Salton,

The SMART Retrieval System: Experiments in Automatic Document Processing,

Prentice-Hall, 1971.

[G1992] Grefenstette, G.,

Use of syntactic context to produce term association lists for retrieval

15th Int. ACM/SIGIR Conf. on R&D in Information Retrieval, SIGIR'92, Copenhagen, Denmark, June 1992, pp.89-97.

[GW1989]B. Ganter and R. Wille

Conceptual scaling.

In F. Roberts, editor, Applications of combinatorics and graph theory to the biological and social sciences, Springer-Verlag, New York, 1989, pp.139--167.

[GW1999] B. Ganter, R. Wille,

Formal Concept Analysis, Mathematical Foundations

Springer, Berlin, 1999.

[HGN2000] Jochen Hipp, Ulrich Guntzer, Gholamreza Nakhaeizadeh

Algorithms for Association Rule Mining-- A General Survey and Comparison.

ACM SIGKDD Explorations 2(1) : pp.58-64, June 2000.

[JCSB2002] Hideo Joho, Claire Coverson, Mark Sanderson, Micheline Beaulieu

Hierarchical presentation of expansion terms.

Proceedings of the ACM symposium on Applied computing, 2002, pp.645 - 649

[JKNSK1996]Kalervo Jervelin, Jaana Kristensen, Timo Niemi, Eero Sormunen, Hiekki Keskustalo

A deductive data model for query expansion.

Proc. of the 19th annual Intl ACM SIGIR Conf. on Research and development in information retrieval 1996, pp.235-243

[JMP1999] Myung-Gil Jang Sung Hyon Myaeng Se Young Park

Using mutual information to resolve query translation ambiguities and query term weighting

Proc. of the 37th conf. on Association for Computational Linguistics, 1999, pp. 223-229

[KO2001] Sergei O. Kuznetsov, Sergei A. Obiedkov

Algorithms for the Construction of Concept Lattices and Their Diagram Graphs

Principles of Data Mining and Knowledge Discovery: 5th European Conference, PKDD 2001, Freiburg, Germany, September 3-5, Proceedings, 2001, pp.289-300

[LBEM1998] Loupy, C., Bellot, P., El-Beze, M., Marteau, P.F.

Query expansion and classification of retrieved documents.

Proceedings of the Seventh Text REtrieval Conference (TREC-7) 1998, pp.382-389.

[LP2001] Dekang Lin, Patrick Pantel

Discovery of inference rules from text.

Proc. of the 7th ACM SIGKDD Intl. conf. on Knowledge discovery and data mining, 2001, pp.323-328

[MSB1998] M. Mitra, A. Singhal, and C. Buckley,

Improving Automatic Query Expansion,

Proc. 21st Ann. Int'l ACM SIGIR Conf. Research and Development in Information Retrieval, 1998, pp. 206-214,

[MTT1999] Rila Mandala, Takenobu Tokunaga, and Hozumi Tanaka.

Combining multiple evidence from different types of thesaurus for query expansion.

Proc. of the 22nd Annual Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval, August 1999, pp.15-19.

[P2000] Uta Priss.

Lattice-based Information Retrieval.

Knowledge Organization, 27(3): pp.132-142, 2000.

[PBTL1999] N.Pasquier, Y.Bastide, R.Taouil, and L.Lakhal.

Discovering frequent closed itemsets for association rules.

In Proceedings of the 7th International Conference on Database Theory, 1999, pp.398-416.

[PBTL1999-1] N.Pasquier Y.Bastide, R.Taouil and L.Lakhal

Efficient Mining of Association Rules Using Closed Itemsets lattices.

Information systems, 24(1): pp.25-46, 1999.

[PHM2000] J. Pei, J. Han and R. Mao

Closet: An efficient algorithm for mining frequent closed itemsets.

In ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, 2000, pp.21-30.

[PW91] Peat, H.J., Willett, P.,

The limitations of term co-occurrence data for query expansion in document retrieval systems,

Journal. of the ASIS, 42(5): pp.378-383, 1991.

[QF1993] Qiu Y. and Frei H.P..

Concept Based Query Expansion.

In Proc. of the 16th Int. ACM SIGIR Conf., ACM Press, June 1993, P.160-169

[S1994] Amanda Spink

Term relevance feedback and query expansion: relation to design

Proc. of the 17th annual Intl. ACM SIGIR conf. on Research and development in information retrieval 1994, pp.81-90

[SM1988]G. Salton, M.J. McGill,

Introduction to modern information retrieval,

McGrawHill, New York, 1988.

[STBPL2000] Gerd Stumme, Rafic Taouil, Yves Bastide, Nicolas Pasquier, Lotfi Lakhal

Fast computation of concept lattices using data mining technics.

In Proc. of the 7th International. Workshop on Knowledge Representation Meets Databases, Berlin, 2000, pp.21-22.

[STBPL2002]Gerd Stumme, Rafik Taouil, Yves Bastide, Nicolas Pasquier, Lotfi Lakhal

Computing iceberg concept lattices with TITANIC.

Data & Knowledge Engineering 42: pp.189-222, 2002.

[SWW1998]Tomek Strzalkowski, Jin Wang, Bowden Wise

Summarization-based query expansion in information retrieval

Proc. of the 17th Intl. conf. on Computational linguistics – 2:pp.1258-1264,1998.

[V1994] Ellen M. Voorhees

Query expansion using lexical-semantic relations

Proc. of the 17th annual Intl. ACM SIGIR conf. on Research and development in information retrieval, 1994, pp.61-69

[VHM2003] Petko Valtchev, Mohamed Rouane Hacene, and Rokia Missaoui

A generic scheme for the design of efficient on-line algorithms for lattices

Proceedings of the 11th Intl. Conference on Conceptual Structures (ICCS'03), Dresde (DE), Springer Verlag (LNAI), (21-25 July) 2003, pp.282-295

[VGRH2003] Petko Valtchev, David Grosser, Cyril Roume, Mohamed Rouane Hacene,

Galicia: an open platform for lattices.

Using Conceptual Structures: Contributions to the 11th Intl. Conference on Conceptual Structures (ICCS'03), Dresde (DE), Shaker Verlag, (21-25 July) 2003, pp. 241-254

[VML2002] Petko Valtchev, Rokia Missaoui, Pierre Lebrun

A partition-based approach towards constructing Galois (concept) lattices.

Discrete Mathematics 256(3):pp.801-829, 2002.

[W1982] R.Wille

Restructuring lattice theory: an approach based on hierarchies of concepts.

I.Rival (ed.). Ordered sets. Reidel, Dordrecht-Boston, 1982, pp.445-470

[HPD2000] Hersh W, Price S, Donohoe L.

Assessing thesaurus-based query expansion using the UMLS Metathesaurus.

Proc AMIA Symp. 2000, pp.344-348.

[Z2000] Mohammed J. Zaki

Conference on Knowledge Discovery in Data

Proc. of the 6th ACM SIGKDD Intl. conf. on Knowledge discovery and data mining, 2000, pp.34 – 43,

[ZH2002] Mohammed J. Zaki, Ching-Jui Hsiao

CHARM: An Efficient Algorithm for Closed Itemset Mining.

Proc. of 2nd SIAM Intl Conf on Data Mining, Arlington, Virginia April 2002, pp.457-473

Appendix

A.1 Bordat algorithm

First we show the original description of Bordat algorithm. In this description, an auxiliary tree is used to construct the line diagram. It is implemented by the sets **Ch. Ch((A,B))** is the set of children of the concept (A,B) in this tree.

0. $L := \varnothing$
1. $Process((O,O'),O')$
2. *L is the concept set.*

$Process((A,B),C)$
1. $L := L \cup \{(A,B)\}$
2. $LN := LowerNeighbors((A,B))$
3. *For each* $(D,E) \in LN$
   3.1. *If* $C \cap E = B$
      3.1.1. $C := C \cup E$
      3.1.2. $Process((D,E),C)$
      3.1.3. $\mathrm{Ch}((A,B) := \mathrm{Ch}((A,b)) \cup \{(D,E)\}$
   3.2. *Else*
      3.2.1. $Find((O,O'),(D,E))$
   3.3. (A,B) *is an upper neighbor of* (D,E)

$Find((A,B),(C,D))$
1. (E,F) *is the first concept in* $\mathrm{Ch}((A,B))$ *such that* $F \subseteq D$
2. *If* $F \neq D$
   2.1. $Find((E,F),(C,D))$
3. *Else* (E,F) *is the desired concept*

LowerNeighbors$((A,B))$

0. $LN := \varnothing$
1. $C := B$
2. *g is the first object in A such that* $\neg(\{g\}' \subseteq C)$*; if there is no such object, g is the last element of A*
3. *While g is not the last element of A*
   - 3.1. $E := \{g\}$
   - 3.2. $F := \{g\}$
   - 3.3. $h := g$
   - 3.4. *While h is not the last element of A*
     - 3.4.1. *h is the next element of A*
     - 3.4.2. *If* $\neg(F \cap \{h\}' \subseteq C)$
       - 3.4.2.1. $E := E \cup \{h\}$
       - 3.4.2.2. $F := F \cap \{h\}'$
   - 3.5. *If* $F \cap C = B$
     - 3.5.1. $LN := LN \cup \{(E,F)\}$
   - 3.6. $C = C \cup F$
   - 3.7. *g is the first object of A such that* $\neg(\{g\}' \subseteq C)$*; if there is no such object, g is the last element of A*
4. *LN is the set of lower neighbors of* (A,B)

A.2 Scaling in Formal Concept Analysis

Conceptual scaling was developed by Ganter and Wille.

A formal notation and definition of the Conceptual scaling is as follows [GW1989]:

Definition A.2.1: A ***many-valued context*** *(G, M, W, I)* consists of sets *G, M*, and *W* and ternary relation *I* between *G, M* and *W* (i.e. $I \subseteq G \times M \times W$), where the following holds:

$(g,m,w) \in I$ and $(g,m,v) \in I$ imply $w = v$.

The elements of *G* are called *objects*, the elements of *M* are called *many-valued*

attributes and the elements of *W attribute values.*

Definition A.2.2: A *conceptual scale* for attribute *m* of the many-valued context is a (onevalued) context $S_m := (G_m, M_m, I_m)$ with $m(G) \subseteq G_m$. The attributes of a scale are called *scale values*, the attributes ***scale attributes***.

Definition A.2.3: For a many-valued context *(G, M, W, I)* and scale contexts $S_m, m \in M$, the *derived context* is *(G, N, J)* with

$$N := \bigcup_{m \in M} M_m \text{ and } g\ J(m,w) :\Leftrightarrow m(g)\ I_m w$$

The many-valued context *(G, M, W, I)* together with the family of scale $S_m, m \in M$ is called a *plainly scaled context*