Automatic Learning Context Tool for Effective Personal Document Indexing and Retrieval

Y.D. Jayaweera, Md Gapar Md Johar, S.N. Perera

1Phd Student, Management and Science University, Malaysia.
2Prof. Dato’, Management and Science University, Malaysia
3Dr.,University of Colombo, Department of Mathematics, Sri Lanka.

Abstract

Managing digital documents has become a time consuming process due to sheer scale. Most users manage their personal documents by creating logical hierarchical folder structures. This logical structure depends on the user’s assessment of the context of the document. Basic file structuring has not been changed for decades and hierarchical file structure remains the same. But there has been a surge in the usage of digital documents. The scale of use of digital documents has led to information overload where users struggle to process facts already encoded and stored on computers to produce on demand information requests. Document Indexing facilitates fast retrieval. Indexing can be field-based, full-text, or a combination of both. Field-based Indexing focuses on identifying and encoding key terms that can uniquely identify a document as much as possible. Automatic field-based indexers lack building semantic relationships between the terms and the document. Hence, the process is augmented by manual indexing for better results which, in turn, consumes time. On the contrary, full-text indexing stores the entire document in a database and users can search it using any of the term in the text. Full-text indexing often overloads the database and makes retrieval inefficient. This research presents an implementation of a tool that manages semi-structured file collections based on Formal Concept Analysis (FCA). Formal Concept Analysis has been applied in document retrieval to identify a coherent set of terms that can best classify documents in different contexts. The tool considers the context of a document and the user behavior.

Key words: Information Retrieval, Personal Document Management, Formal Concept Analysis (FCA), Context Aware Document Retrieval.

1. INTRODUCTION

Today, information, especially digital information, is not a scarce resource; information exists in abundance and human time and attention have now become the scarce resource [2]. Information overload is now a recognized problem as people struggle to manage the increasing quantities of information they need to deal with on a daily basis [1]. Personal document management is the activity of managing a collection of digital documents performed by the owner of the documents, and consists of creation/acquisition, organization, finding and maintenance. Document management is a pervasive aspect of digital work, but has received relatively little attention from researchers. The hierarchical file system used by most people to manage their documents has not conceptually changed in decades. Unfortunately, strict hierarchical structures can map poorly to user needs. The use of document locations as a fundamental organizing principle, and the restriction that documents appear in only one location at a time, force users to create strict categorizations of document types and organization [3].

Although revolutionary prototypes have been developed, these have not been grounded in a thorough understanding of document management behaviour and, therefore, have not resulted in significant changes to document management interfaces [4]. Many personal document management tools exist but none of the tools considers the user context to store and manage documents. Most of the tools equipped with inverted index used for indexing, use a bag of keywords to link terms to documents. The main drawback of the inverted index is that it is flat and does not count semantic structure of terms and relationships existing between terms used in a context [5]. This is due to the fact that in most contexts they lack a clear division in words or phrases, a prerequisite for inverted indexes to function properly.

The results of this research culminated in the development of a personal document management tool using Formal Concept Analysis (FCA) which incorporates activities of a Personal Document Management system.

The rest of the paper is organized as follows. Section 2 gives an overview of the current status of Personal Document Management systems and application of FCA in Information Retrieval. Section 3 presents the discussion of the proposed Personal Document Management Tool. Section 4 describes the system architecture and implementation details. Finally Section 5 concludes the work with topics for future direction.
2. LITERATURE REVIEW

Personal Document Management (PDM) has three key areas: Filing, Organizing and Finding [4]. Filing refers to creating an appropriate folder structure and storing of files. With a proper filing strategy users can locate the file without consuming much time. Organizing means creating meaningful and deep document structures similar to a mind map. Related folders are grouped together to create more meaningful structures and to hide complexities. Folders are arranged in such a manner that one can extract the context through the folder structure. Finding implies locating files required by the User. The key areas are highly interconnected. Previous research shows that there is a tradeoff between filing and finding. The higher filing effort yields lesser finding effort [4]. But due to sheer volume of files and time constraints Users sometimes may not adhere to a proper filing strategy and this makes locating a file very hard at a later stage. There are a number of tools developed to assist Users in Personal Document Management activities which yield to manage and find files on their personal desktops. The tools can be categorized as PDM Suites, Mind mapping tools, tools for Note taking, Desktop search tools and Semantic PDM tools [6]. PDM Suites such as MS Outlook, Lotus Notes and the like focus mainly on storage and they lack context [6]. In such tools files are stored somewhere but they do not come out when the User needs them. Mind mapping tools like MindManager and Freemind allow Users to connect structures and build a cognitive model but such tools lack searching and navigation. Note-taking tools like OneNote help pillars more than filers. Such tools can capture information quickly but have weak support for structured data, making it difficult to map User’s context [7]. Desktop search tools like Google Desktop Search is a local search engine that facilitates file search and background indexing.

More advanced searching features are inbuilt into modern Operating systems. Windows Vista, Windows 8 and 7 to facilitate search as an inbuilt feature in the OS [8]. The search result generated by the OS tools can be saved for later reference. The stored file saves the query which was used to search the files in XML format. When the user runs the stored XML file, the operating system re-runs the query on the Windows search subsystem. Their primary objective is to locate the file fast but no context, no hierarchy, no organization and no ranking. There are systems that remember files a user has visited and allows a user a quick access to the files recently visited. A related approach under this is the “Stuff I’ve Seen” system [7], which simply remembers all entities including files, Web pages, emails, contacts, etc. that a User uses on a computer. But like in the previous methods this lacks semantics which has no context. On the other hand, revolutionary semantic PDM tools like Nepomuk-KDE connects words with a meaning and associate documents [9]. Semantic PDM tools leverage PDM activities by superior managing and locating files due to semantic rich data associated with it. seMouse is a another semantic desktop tool [10]. It introduces the notion of a knowledge folder as a coarse set of documents bound together by a common ontology. These semantic PDM tools require manual tagging in order to build and tune the knowledgebase which consumes an additional amount of time and effort for the Users.

Considering the features and limitations of the existing tools for Personal Document Management it is observed that to bridge the gap of connecting words with meaning and to build association with documents automatically, a knowledge layer is needed. To build a knowledge layer on top of the existing file structure Formal Concept Analysis is proposed due to its inherent hierarchical structure similar to User’s cognition. This study focuses on building a personal document management tool to facilitate document acquisition, organization, finding and maintenance of semi-structured files in a personal document collection. The proposed tool facilitates logical structuring of semi-structured file collection incrementally into a concept lattice to assist filing and facilitate finding of documents through the search API that uses an index built automatically.

2.1 Formal Concept Analysis for Information Retrieval

Formal Concept Analysis (FCA) is based on a mathematical framework for analyzing and structuring a domain of interest [11, 13]. FCA can serve as a guideline for context building because it allows the identification of concepts by factoring out their commonalities while preserving concept specialization relationships. Due to this relationship between concepts in a concept lattice, it provides scaling of context at different levels of granularity. FCA transforms a formal context into a concept lattice. A formal context is a representation of the relation between objects and their attributes. Formally, a context is a triple $k = (G, M, I)$, where $G$ is a set of objects, $M$ is a set of attributes, and $I$ is a binary relation $I \subseteq G \times M$. Given a set of objects $A \subseteq G$, the shared image of $A$ in $M$ is defined as:

$$A \uparrow = \{m \in M | (g, m) \in I \lor g \in A\}$$  \hspace{1cm} (1)$$

Similarly, for a set of attributes $B \subseteq M$, its shared image in $G$ is:
B ⊥ = \{g ∈ G | (g, m) ∈ I \forall m ∈ B\} \quad (2)

A ⊆ G, B ⊆ M and A = B ↓, B = A ↑. A is called the extent of the concept and B is called the intent of the concept [11, 13]. In other words, equation 1 defines the collection of all attributes shared by all objects from A, and Equation 2 defines the collection of all objects sharing all the attributes from B. A partial ordering can be defined over the concepts of a context. Specifically,

\[(A1, B1) \sqsubseteq (A2, B2) ↔ A1 \subseteq A2\]

Equivalently

\[(A1, B1) \sqsubseteq (A2, B2) ↔ B1 \sqsupseteq B2\]

The set of all formal concepts of a given context with the sub concept-super concept relation (⊆) is always a complete lattice, called the concept lattice.

The significant advantage of applying FCA in information retrieval is that the mathematical formulas of FCA can construct the conceptual structure which has generalization and specialization relationships among the concept nodes automatically [11, 13]. In this approach, a document is annotated with a set of keywords which is then used for lattice generation. Concept on each node defines the context which associates a bag of words (Intents) with the documents (Extents). This lattice structure allows User to reach a group of documents via one path, but then rather than going back up the same hierarchy and guessing another starting point (random seeker model), one can go to other parents of the present node improving the problem of category mismatch. To develop accurate Lattice it is required to have good representation of keywords (bag of words) of the document collection and another problem of FCA is the memory consumption. Since most of the FCA algorithms are memory based the scalability of FCA to support large collection is limited. In this study the topics are extracted from semi-structured word documents (memoranda) and used for concepts generation.

2.2 Incremental Construction of the Concept Lattice

Once the concept lattice is built it is required to maintain it with the changes. Specially, in the context of a Personal Document Management system there will be lots of new documents added into the collection. Considering its dynamic nature, it is necessary to generate only the completed pairs of the lattice without regenerating it from scratch. Incremental algorithms outperform most of the batch algorithms [12]. AddIntent algorithm is an approach to update formal concepts and concept lattice incrementally. It was done by adding a new document with a set of keywords or by refining the keywords for an existing document. AddIntent uses a similar methodology based on Propositions 1 and 2 below to avoid the consideration of old concepts and non-canonical generators in the lattice in a bottom-up search for canonical generators and modified concepts [12].

Proposition 1. If (B', B) is a canonical generator of a new concept (F', F), while (D', D) is a non-canonical generator of (F', F) – in this case, B ⊆ D – then any concept (H', H) such that H ⊆ D and H ⊈ B is neither modified nor is a canonical generator of any new concept.

Proposition 2. If (D', D) is an old concept and D ∩ g' = B – in this case, (B', B) ∈ Li is modified – then any concept (H', H) such that H ⊆ D and H ⊈ B is neither modified nor is a canonical generator of any new concept.

AddIntent algorithm relies on a two-phased iterative approach to first discover canonical generators and in the second phase to update the lattice. AddIntent combines these two phases. In the proposed framework a similar kind of approach is adopted due to its novelty. The proposed framework uses a relational database management system to store, build and retrieve concepts to facilitate parallel insert and fast search. The system uses stored procedures and triggers to track and maintain the changes at the database level. Hence, the entire concept lattice is stored and maintained by the relational database management system.

3. THE PROPOSED DESIGN

This section starts with the rationale behind designing a new Personal Document Management tool and then explores core algorithms the proposed tool is based on with the basic strategy each algorithm follows.

The tool facilitates Personal Document Management in a dynamic environment. The activities supported by the tool consist of document acquisition, organization, finding and maintenance. The tool has the following features notably;

- Documents are indexed as concepts
- The index is a hierarchy of Concepts which are linked to one another
- The index is stored in a database
- Supports dynamic addition of documents to the collection (AddIntent)
- Incorporates document deletions and updates (DeleteExtent)
- Supports Query refinement based on support (SearchExtent)
- Allows pruning of search space by using minimum support

The tool uses algorithms; Next Closure algorithm to build initial lattice [11, 13], AddIntent algorithm to add new concepts and maintain the initially built lattice
incrementally [14]. DeleteExtent algorithm to delete documents from the collection, UpdateIntent algorithm to update the parent concepts after a deletion and SearchExtent algorithm to retrieve documents with their support. The algorithms Next Closure and AddIntent were introduced to facilitate activities associated with document management system. The following section outlines the detail of each algorithm used in the personal document management tool.

3.1 The Next Closure Algorithm

The algorithm is used to build the initial lattice from the subset of documents already available for indexing. The limitation of the algorithm is, it shows no additive result in rebuilding the entire lattice from the scratch.

Algorithm: Next Closure
Input: A closure operator \( X \mapsto X'' \) on a finite set \( M \), and a subset \( A \subseteq M \).
Output: \( A \) is replaced by the lexically next closed set.

begin
i := largest element of \( M \);
i := succ(i);
success := false;
repeat
i := pred(i);
if \( i \notin A \) then begin
A := A \cup \{i\};
B := A;
if \( B \setminus A \) contains no element \( < i \) then begin
A:= B;
success := true;
end;
end else A := A \setminus \{i\};
until success or \( i \) = smallest element of \( M \).
end.

Next Closure algorithm takes a closed set as input and outputs the next closed set according to a particular lexicographical order, which is a linear extension of the subset-inclusion order. Assuming a linear order \( \prec \) on attributes in \( M \), we say that a set \( A \subseteq M \) is lexically smaller than a set \( B \subseteq M \) if

\[ \exists b \in B \setminus A \forall a \in A (a < b \Rightarrow a \in B) \]

In other words, the lexically largest among two sets is the one containing the smallest element in which they differ [13].

3.2 AddIntent Algorithm

The problem of generating the set of all concepts of a formal context to incorporate changes is costly. The cost is in terms of computation time and memory consumption. The AddIntent function being incremental, it relies on the lattice constructed from the subset of objects of the context to integrate the next object into the lattice [14]. Therefore, its use is most appropriate in those applications that require an incremental approach to manage documents in a dynamic environment.

Function AddIntent(intent, GeneratorConcept, L)

GeneratorConcept := GetMaximalConcept(intent, GeneratorConcept, L)
If GeneratorConcept.Intent = intent
Return GeneratorConcept
End If
GeneratorParents := GetParents(GeneratorConcept, L)
NewParents := \( \emptyset \)
For each Candidate in GeneratorParents
If Candidate.Intent \( \not\subseteq \) GeneratorConcept.Intent
Candidate := AddIntent(Candidate.Intent \( \cap \) intent, Candidate, L)
End If
End For
If addParent := true
For each Parent in NewParents
If Candidate.Intent \( \not\subseteq \) Parent.Intent
addParent := false
Exit For
Else If Parent.Intent \( \not\subseteq \) Candidate.Intent
Remove Parent from NewParents
End If
End For
End If
End For
Add Candidate to NewParents
End If
End For
NewConcept := (GeneratorConcept.Extent, intent)
L := L \cup \{NewConcept\}
For each Parent in NewParents
RemoveLink(Parent, GeneratorConcept, L)
SetLink(Parent, NewConcept, L)
End For
SetLink(NewConcept, GeneratorConcept, L)
Return NewConcept

The parameters of the function AddIntent(intent, GeneratorConcept, L) are the intent of a new concept to be placed into the concept lattice \( L \) and a precomputed GeneratorConcept, such that intent is a subset of the intent of GeneratorConcept. AddIntent returns a concept whose intent corresponds to intent a new concept will be created if there was no such concept before or an existing one will be returned otherwise. First, the algorithm finds the most general concept whose intent is a superset of intent and assigns it to GeneratorConcept. If the intent of this concept is equal to intent, then the desired concept is already in the lattice and the algorithm terminates. Otherwise, GeneratorConcept is the canonical generator of the new concept, which has to be created and linked to other concepts in the lattice. To find the parents of the new concept in the diagram graph, it examines all parents of
GeneratorConcept. If the intent of such a parent, called Candidate, is a subset of intent, then Candidate is modified. Otherwise, a recursive call to AddIntent ensures that the lattice contains a concept whose intent is equal to the intersection of intent and the intent of Candidate. This concept is assigned to Candidate. Then, Candidate is added to the (initially empty) NewParents list if it is minimal among its current elements (that is, has a maximal intent). At the same time, if some concept in NewParents is more general than Candidate, this concept is removed from the list. Thus, the NewParents list always contains incomparable (w.r.t. being more general) concepts. Moreover, in the end, it contains precisely the parents of NewConcept that is to be inserted.

3.3 DeleteExtent Algorithm

The DeleteExtent function uses the mathematical foundation based on lectic order where concepts are arranged in a totally ordered relationship. The purpose of the algorithm is to identify the Candidate Concept where document removal took place.

Function DeleteExtent (extent, GeneratorConcept, L)

intent := GetIntents(extent)
GeneratorConcept := GetMaximalConcept(intent,GeneratorConcept, L)
If GeneratorConcept.Intent = intent
    Return GeneratorConcept
Else
    Return Ø
End If

The parameters of the function DeleteExtent (extent, GeneratorConcept, L) are the extent of a concept to be deleted from the concept lattice L and a precomputed GeneratorConcept, such that extent's intent is a subset of the intent of GeneratorConcept. DeleteExtent returns a concept whose intent corresponds to intent of the candidate if there was no such concept that matches NULL will be returned.

3.4 UpdateIntent Algorithm

The UpdateIntent procedure is used to propagate the changes (deletions) made in the Candidate concept to all parent concepts.

Procedure UpdateIntent (extent, GeneratorConcept, L)

GeneratorParents := GetParents(GeneratorConcept, L)
For each Candidate in GeneratorParents
    RemoveObject(extent, candidate, L)
End For
RemoveObject(extent, GeneratorConcept, L)

The parameters of the procedure UpdateIntent (extent, GeneratorConcept, L) are the extent of a concept to be updated in the concept lattice L and a precomputed GeneratorConcept, such that extent’s intent is a subset of the intent of GeneratorConcept. UpdateIntent iteratively navigates through the parent concepts in order to update the concepts and finally it updates the GeneratorConcept.

3.5 SearchExtent Algorithm

The user’s search for an extent (Object) is supported through SearchExtent algorithm. The parameters represent search terms, root concept and the concept lattice L. The function returns the formal concept which matches the intent.

Function SearchExtent (intent, GeneratorConcept, L)

GeneratorConcept = GetMaximalConcept(intent,GeneratorConcept, L)
If GeneratorConcept.Intent = intent
    Return GeneratorConcept
Else If GeneratorConcept.Intent <> intent
    Return GetMaximalConcept(GeneratorConcept.Intent ∩ intent,GeneratorConcept, L)
Else
    Return BottomConcept
End if.

4. IMPLEMENTATION

4.1 The Framework

The proposed tool uses the framework proposed in earlier research work carried out by the author [15]. The framework in figure 1 functions in two ways. Firstly, the framework builds ordered concepts using the terms used in the Subject field of the semi-structured documents as part of filing process. The complete lattice is then constructed and stored in a MySQL database. Secondly the search interface, when user requests files through the Search API the framework returns recommended documents through searching the ordered concept’s Intent and retrieving its Extent. The proposed tool extracts concepts from structured documents automatically, so that documents which are similar will be clustered together and then map search queries to concepts to recommend the documents [15].
4.2 The Context Aware Personal Document Management Tool

The implementation of the tool is based on the Java Development Kit 1.8 and a number of Open-Source projects. For the indexing process semi-structured word document collection is used. The tool also extracts metadata from the file system such as file size and date of last change. The FCA related mapping from concepts to graph for visualization is done through ToscanaJ which is another Open Source project. All the concepts are stored in a MySQL database and all the maintenance of formal concepts are done through stored procedures. Figure 2 shows the main interface of the Context Aware Personal Document Management Tool.

The main interface in figure 2 facilitates automatic concept extraction of stored semi-structured word documents. The semi-structured documents are used to improve effectiveness of the semantic relatedness. Once the concepts are generated the concept lattice is stored in MySQL database. Figures 2 and 3 represent a snapshot of stored documents where related terms are identified.
Once the tool builds the formal concepts user can review the formal context for fine tuning or even manual tagging. This enables better document retrieval at a later stage. Figure 4 depicts a possible context of the document collection and related terms of the previous snapshot.

Figure 5 shows a visual representation of how the formal contexts are arranged. An open source project ToscanaJ is used to build module. The structured representation of the results allows removal over specified queries and support of the search result is improved using term classification. The changes to the file system are observed and incorporated into the stored concepts using AddIntent and UpdateIntent algorithms. This module is deployed as a windows service.

The approach equipped allows users to extract documents based on a bag of words which best represents each document. Since the keywords are organized as coherent concepts, over specified queries can be easily removed by using a threshold value. The searching process is fine-tuned by the use of WordNet which facilitates automatic term extraction based on similarity. In the initial prototype only the crisp set are supported. The Search interface supports Boolean AND, OR, NOT operations where user can use them to refine the search result. Figure 6 shows the search interface with a search result. Each concept retrieved has a support. The support indicates the abstract level of a concept.

In this paper a tool for an automatic learning context for effective personal document indexing and retrieval system was presented. The tool builds an automatic incremental concept lattice using a coherent set of terms that best describe a given document. As the document collection, a collection of semi-structured word documents were used to extract context of each object in the document collection. The tool receives on demand document search requests from users and it is able to return the relevant files through the Search Interface. The Search Interface uses a term base to identify synonyms. The generated output is validated through minimum support for concepts. The support of a concept in the lattice is also used for document filtering and ranking. The tool currently handles only semi-structured word documents since it was developed as a proof of concept. It needs pluggable file handlers to handle multiple file types and metadata to describe files. The search interface also

5. CONCLUSION AND FUTURE WORK

The existing hierarchical structure organizes documents and requires users to adhere filing to the system structure. This process makes it hard for users to file, locate and share documents. Moreover, the structure dictates document retrieval process and lacks flexibility. The Placeless Documents system introduces the concept of personalized document properties as a uniform mechanism for organizing, filing, grouping, retrieving and manipulating documents and document collections [3, 16].
requires fine tuning. The current implementation supports only crisp sets and only Boolean operators can be applied on the formal concepts. Storing full concept lattice in the database facilitates parallel insertions and concurrent access to the formal concepts. The clustering technique utilized shows the fact that a given document can be attached to many topics avoiding one leveled index where a document belongs to a single cluster. The hierarchical ordering of concepts facilitates a document to be attached in multiple clusters. Learning context from documents automatically and building a dynamic context lattice makes file structuring and indexing a step closer to human reasoning.

REFERENCES


