CEM – Visualisation and Discovery in Email

Richard Cole[1], Peter Eklund[1], Gerd Stumme[2]

[1] Knowledge, Visualisation and Ordering Laboratory
School of Information Technology, Griffith University, Gold Coast Campus,
PMB 50, Gold Coast Mail Centre QLD 9726, AUSTRALIA;
r.cole,p.eklund@gu.edu.au
[2] Institut für Angewandte Informatik und Formale
Beschreibungsverfahren, Universitt Karlsruhe (TH)
D-76128 Karlsruhe, GERMANY; stumme@aifb.uni-karlsruhe.de

Abstract. This paper presents a lattice-based visual metaphor for knowledge discovery in electronic mail. It allow a user to navigate email using a visual lattice metaphor rather than a tree structure. By using such a conceptual multi-hierarchy, the content and shape of the lattice can be varied to accommodate any number of queries against the email collection. The system provides more flexibility in retrieving stored emails and can be generalised to any electronic documents. The paper presents the underlying mathematical structures, and a number of examples of the lattice and multi-hierarchy working with a prototypical email collection.

1 Introduction

Email management systems usually store email as a tree structure in analogue to the file management system. This has the advantage that trees are simple and easily explained to novice users as a direct mapping from the physical structure of the file system. The disadvantage is that at the moment of storing an email the user preempts the way he will later retrieve the email. The tree structure forces a decision about the criteria considered a primary and secondary indexes for the email. For instance, when storing email regarding the organization of a conference, one needs to decide whether to organise the email as `Komorowski/pkdd2000/program_commitee` where "Komorowski" is a primary index alternatively `conferences/pkdd/pkdd2000/organisation`. This problem is common when a user cooperates with overlapping communities on different topics with multiple viewpoints. Should we organise our email as a specialisation or a generalisation hierarchy, i.e. are we trying to give every email a unique key based on its content or group emails together broadly on category? There is no general answer except that it is context and query dependent.

In this paper, we profile the *Conceptual Email Manager* called Cem. This follows earlier work reported in [3, 2]. Cem is a lattice-based email retrieval and storage programme that aids in knowledge discovery by a number of flexible views over email. It uses as data structure for storing emails by a concept lattice rather than a tree. This permits clients to retrieve emails along different paths.

For the example above, the client need not decide which of the two paths to store the email. When retrieving the email later, he can consider any combination of the catchwords in the two paths. Email retrieval is totally independent of the physical organisation of the file system.

There are related approaches to the above problem. For instance, the concept of a *virtual folder* was introduced in a program called View Mail (VM)[6]. A virtual folder is a collection of email documents retrieved in response to a query. The virtual folder concept has more recently been popularised by a number of open-source projects, e. g. [8]. Our system differs from those projects in the understanding of the underlying structure – via formal concept analysis – and in its implementation.

Concept lattices are defined in the mathematical theory of Formal Concept Analysis [11]. A concept lattice is derived from a binary relation which assigns attributes to objects. In our application, the objects are all emails stored by the system, and the attributes catchwords like 'conferences', 'Komorowski', or 'organisation'. We assume the reader to be familiar with the basic notions of Formal Concept Analysis, and otherwise refer to [5]. In the next section, we describe the mathematical structures of the CEM. Requirements for their maintenance are discussed in Section 3. We also describe how they are fulfilled by our implementation.

2 Mathematical Structure

We assume the reader familiar with the two basic notions of Formal Concept Analysis: formal context and concept lattice. Definitions and examples can be found in [5]. In this section, we describe the system on a structural level; we abstract from implementation details. They are discussed in Section 3. We distinguish three fundamental structures:

1. a *formal context* that assigns to each email a set of catchwords;
2. a *hierarchy* on the set of catchwords in order to define more general catchwords;
3. a mechanism for creating *conceptual scales* used as a graphical interface for email retrieval.

2.1 Assigning Catchwords to Email

In the Cem, we use a *formal context* (G, M, I) for storing email and assigning catchwords. The set G contains all emails stored in the system, the set M contains all catchwords. For the moment, we consider M to be unstructured. (In the next subsection we will introduce a hierarchy on it.)

The relation I indicates emails assigned to each catchword. In the example given in the introduction, the client might want to assign all the catchwords 'Komorowski', 'pkdd2000, 'program_commitee', 'conferences', 'pkdd, and 'organisation' to a new email. The incidence relation is generated in a semi-automatic process: (i) an automatic string-search algorithm recognizee words within sections of an email and suggests relations between email attributes, (ii) the client may accept the suggestion of the string-search algorithm or otherwise modify it, and (iii) the client may attach his own attributes to the email. In Section 3, we will discuss how the user is supported in this assignment. At the moment, we suppose that the relation is given.

Instead of a tree of disjoint folders and sub-folders, we consider the concept lattice $\underline{\mathfrak{B}}(G, M, I)$ as navigation space. The formal concepts replace the folders. In particular, this means that emails can appear in different concepts. The most general concept contains all email. The deeper the client moves into the hierarchy, the more specific the concepts, and subsequently the fewer emails they contain.

2.2 Hierarchies of Catchwords

To support the semi-automatic assignment of catchwords to the emails, we provide the set M of catchwords with a partial order $\leq$. For this *subsumption hierarchy*, we assume that the following *compatibility condition* holds:

$$\forall g \in G,\ m, n \in M\colon \quad (g, m) \in I,\ m \leq n \ \Rightarrow\ (g, n) \in I \qquad (\ddagger)$$

i.e., the assignment of catchwords respects the transitivity of the partial order. Hence, when assigning catchwords to emails, it is sufficient to assign the most specific catchwords only. More general catchwords are automatically added.

For instance, the user may want to say that 'pkdd' is a more specific catchword than 'conferences', and that 'pkdd2000' is more specific than 'pkdd2000' (i. e., 'pkdd2000'≤'pkdd'≤'conferences'). Emails concerning the creation of this paper are assigned by the email client to 'pkdd2000' only (and possibly some additional catchwords like 'cole', 'eklund' and 'stumme'). When the client wants to retrieve this email, he is not required to recall the pathname. Instead, they also appear under the more general catchword 'conferences'. If 'conferences' provides too large a list of email, the client can refine the search, by choosing a sub-term like 'pkdd', or adding a new catchword, for instance 'cole'.

Notice that even though we impose no specific structure on the subsumption hierarchy $(M, \leq)$ it naturally splits three ways. One relates the contents of the emails, e.g., if an email is related to 'conference' (or not) or classified to 'organisation' etc. A second relates to the sender or receiver of the email. The third describes aspects of the emailing process (if it is inbound or outbound mail). An example of a hierarchy is given in Fig. 1.

The hierarchy displayed in Fig. 1 is a forest (i. e., a union of trees), but the resulting concept lattice — used as the search space — is by no means a forest. The partially order set is displayed both in the style of a folding editor and as a connected graph.

Consider for example the concept generated by the conjunction of the two catchwords 'PKDD 2000' and 'conference organisation'. It will have at least two incomparable super-concepts, namely the one generated by the catchword 'PKDD 2000' and the one generated by the catchword 'conference organisation'. In general, all we know is that the resulting concept lattice is embedded as a join-semilattice in the lattice of all order ideals (i. e., all subsets $X \subseteq M$ s. t. $x \in X$ and $x \leq y$ imply $y \in X$) of $(M, \leq)$.

Fig. 1. Partially ordered set of catchwords: as a folding editor and connected graph.

2.3 Conceptual Scales and Navigating Email

Conceptual scaling deals with many-valued attributes. Often attributes are not one-valued as are the catchwords given above, but allow a range of values. This is modelled by a *many-valued context.* A many-valued context is roughly equivalent to a relation in a relational database with one field being a primary key. As one-valued contexts are special cases of many-valued contexts, conceptual scaling can also be applied to one-valued contexts to reduce the complexity of the visualisation.

In this paper, we only deal with one-valued formal contexts. Readers who are interested in the exact definition of many-valued contexts and the use of conceptual scaling in this more general case are referred to [5]. Applied to one-valued contexts, conceptual scales are used to determine the concept lattice that arises from one vertical 'slice' of a large context:

Definition 1. *A* conceptual scale *for a subset $B \subseteq M$ of attributes is a (one-valued) formal context $\mathbb{S}_B := (G_B, B, \ni)$ with $G_B \subseteq \mathfrak{P}(B)$. The scale is called* consistent *wrt $\mathbb{K} := (G, M, I)$ if $\{g\}' \cap B \in G_B$ for each $g \in G$. For a consistent scale $\mathbb{S}_B$, the context $\mathbb{S}_B(\mathbb{K}) := (G, B, I \cap (G \times B))$ is called its* realized scale.

Conceptual scales are used to group together related attributes. They are determined as required by the user, and the realized scales are derived from them when a diagram is requested by the user. CEM stores all scales that the client has defined in previous sessions. To each scale, the client can assign a unique name. This is modelled by a mapping ($\mathcal{S}$).

Definition 2. *Let $\mathcal{S}$ be a set, whose elements are called scale names. The mapping*

$$\alpha: \mathcal{S} \rightarrow \mathfrak{P}(\mathcal{M})$$

defines for each scale name $s \in \mathcal{S}$ *a scale* $\mathbb{S}_s := \mathbb{S}_{\alpha(s)}$.

For instance, the user may introduce a new scale which classifies the emails according to being related to a conference by adding a new element 'Conference' to $\mathcal{S}$ and by defining $\alpha(\text{Conference}) := \{\text{CKP '96}, \text{AA 55}, \text{KLI '98}, \text{Wissen '99}, \text{PKDD 2000}\}$.

Observe that $\mathcal{S}$ and M need not be disjoint. This allows the following construction deducing conceptual scales directly from the subsumption hierarchy: Let $\mathcal{S} := \{m \in M | \exists n \in M \colon n < m\}$, and define, for $s \in \mathcal{S}$, $\alpha(s) := \{m \in M | m \prec s\}$ (with $x \prec y$ if and only if $x < y$ and there is no z s. t. $x < z < y$). This means all catchwords $m \in M$, neither minimal nor maximal in the hierarchy, are considered as the name of scale $\mathbb{S}_m$ and as a catchword of another scale $\mathbb{S}_n$ (where $m \prec n$). This last construction, first been presented in [9], defines a hierarchy of conceptual scales for a library information system [7].

3 Requirements of the Cem

In this section, we discuss requirements of the Cem based on the Formal Concept Analysis paradigm. In the following section we explain how our implementation responds to these requirements. The requirements may be divided along the same lines as the underlying mathematical structures defined in Section 2;

1. assist the user in editing and browsing a catchword hierarchy;
2. help the client visualise and modify the scale function α;
3. allow the client to manage the assignment of catchwords to emails;
4. assist the client search the conceptual space of emails for both individual emails and conceptual groupings of emails.

In addition to the requirements stated above, a good email system needs to be able send, receive and display emails: processing the various email formats and interacting with the current popular protocols. Since these requirements are already well understood and implemented by existing email programs they are not discussed further.

Editing and Modifying a Catchword Hierarchy The catchword hierarchy is a partially ordered set $(M, \leq)$ where each element of M is a catchword. The requirements for editing and browsing the catchword hierarchy are:

- graphically display the structure of the $(M, \leq)$. The ordering relation must be evident to the client.
- make accessible to the client a series of direct manipulations to alter the ordering relation. It should be possible to create any partial order to a reasonable size limit.

Visualising and Modifying the Scale Function α The user must be able to visualise the scale function, α, explained in Section 2. The program must allow an overlap between the set of scale labels, $\mathcal{S}$, and the set of catchwords M.

Fig. 2. Scale, catchword and concept lattice. The central dialogue box shows how α can be edited.

Managing Catchwords Assignment Section 2 introduced the formal context (G, M, I). This formal context associates email with catchwords via the relation I. Also introduced was the notion of the *compatability condition*,(‡).

The program should store the formal context (G, M, I) and ensure that the compatability condition is always satisfied. It is inevitable that the program will have to sometimes modify the formal context in order to satisfy the compatability condition after a change is made to the catchword hierarchy.

The program must support two mechanisms for the association of catchwords to emails. Firstly, a mechanism in which emails are automatically associated with catchwords based on the email content. Secondly, the user the should be able to view and modify the association of catchwords with emails.

Navigating the Conceptual Space The program must allow the navigation of the conceptual space of the emails by drawing line diagrams of concept lattices derived from conceptual scales [5]. This is shown in Fig. 2. These line diagrams should extend to locally scaled nested line diagrams[9] shown in Fig. 3. The program must allow retrieval and display of emails forming the extension of concepts displayed in the line diagrams.

4 Implementation of Cem

This section divides the description of implementation of the Cem into a similar structure to that presented in Section 3.

Fig. 3. Scale, catchword and nested-line diagram.

4.1 Catchword Hierarchy

Browsing the Hierarchy The user is presented with a view of the hierarchy, $(M, \leq)$ as a tree widget[1], shown in Fig. 2. The tree widget has the advantage that most users are familiar with its behaviour and it provides a compact representation (in the sense screen space) of a tree structure.

The catchword hierarchy, being a partially ordered set, is a more general structure than that of tree. Although the example given in Fig. 1 is a forest, no limitation is placed by the program on the structure of the partial order other than that it must be a partial order.

The following is a definition of a tree derived from the catchword hierarchy for the purpose defining the contents and structure of the tree widget. Let $(M, \leq)$ be a partially ordered set and denote the set of all sequences of elements from M by $< M >$. Then the tree derived from the catchword hierarchy is comprised by $(T, \text{parent}, \text{label})$, where $T \subseteq < M >$ is a set of tree nodes, $<>$ the empty sequence is the root of the tree, $\text{parent} : T/ <> \rightarrow T$ is a function giving the parent node of each node (except the root node), and $\text{label} : T \rightarrow M$ assigns a catchword to each tree node.

$$T = \{< m_1, \ldots, m_n > \in < M > \mid \; m_i \preceq m_{i+1} \text{ and } m_n \in \text{top}(M)\}$$

$$\begin{aligned} \text{parent}(< m_1, \ldots, m_n >) &:= < m_1, \ldots, m_{n-1} > \\ \text{parent}(< m_1 >) &:= <> \\ \text{label}(< m_1, \ldots, m_n >) &:= m_1 \end{aligned}$$

[1] A widget is a graphical user interface component with a well defined behaviour usually mimicking some physical object, for example a button.

Each tree node is identified by a path from a catchword to the top of the catchword hierarchy. The tree representation has the disadvantage that elements from the partial order occur multiple times in the tree and the tree can become large. If however the user keeps the number of elements with multiple parents in the partial order to a small number, the tree is manageable.

Modifying the Hierarchy $(M, \leq)$ The program provides four operations for modifying the hierarchy: insert & remove catchword and insert & remove ordering. More complex operations provided to the client, moving an item in the taxonomy, are resolved internally to a sequences of these basic operations. In this section we denote the order filter of m as $\uparrow m := \{x \in M \mid m \leq x\}$, the order ideal of m as $\downarrow m := \{x \in M \mid x \leq m\}$, and the upper cover of m as $\succ_m := \{x \in M \mid x \succ m\}$.

The operation of insert catchword simply adds a new catchword to M, and leaves the $\leq$ relation unchanged. The remove catchword operation takes a single parameter $a \in M$ for which the lower cover is empty, and simply removes a from M and $(\uparrow a) \times \{a\}$ from the ordering relation.

The operation of insert ordering takes two parameters $a, b \in M$ and inserts into the relation $\leq$, the set $(\uparrow a) \times (\downarrow b)$. The operation of remove ordering takes two parameters $a, b \in M$ where a is an upper cover of b. The remove ordering operation removes from $\leq$ the set $((\uparrow a / \uparrow (\succ_b /a)) \times (\downarrow b))$.

4.2 Visualisation of the Scale Function α

The set of scales S, according to the mathematisation in Section 2 is not disjoint from M, thus the tree representation of M already presents a view of a portion of S. In order to reduce the complexity of the graphical interface, we make S equal to M, i.e. all catchwords are scale labels, and all scale labels are catchwords.

Such an assumption is made possible by the definition of the default scale for a catchword given in Section 2. A result of this definition is that catchwords with no lower covers lead to trivial scales containing no other catchwords.

The function α maps each catchword m to a set of catchwords. The program displays this set of catchwords, when requested by the user, using a dialog (see Fig. 2 – centre). The dialog box contains all catchwords in the down-set of m an icon (either a tick, or a cross) to indicate membership in the set of catchwords given by $alpha(m)$. Clicking on the icon changes the membership of $\alpha(m)$.

By only displaying the down-set of m in the dialog box, the program restricts the definition of α to $\alpha(m) \subseteq \downarrow m$. This has an effect on the "remove ordering operation" defined on $(M, \leq)$. When the ordering of $a \leq b$ is removed the image of α function for attributes in $\uparrow a$ must be checked and possibly modified.

4.3 Associating Emails with Catchwords

Each member of $(M, \leq)$ is associated with a query term, in this application is a set of *section/word pairs*. That is: Let H be the set of sections found in the email documents, W the set of words found in email documents, then a function $\texttt{query}\colon M \to \mathfrak{P}(H \times W)$ attaches to each attribute a set of section/word pairs.

Let G be a set of email. An inverted file index stores a relation $R_1 \subseteq G \times (H \times W)$ between documents and section/word pairs. $(g, (h, w)) \in R_1$ indicates that document g has word w in section h.

A relation $R_2 \subseteq G \times M$ is derived from the relation R_1 and the function query via: $(g, m) \in R_2$ iff $(g, (h, w)) \in R_1$ for some $(h, w) \in \text{query}(m)$. A relation R_3 stores user judgements saying that an email should have an attribute m. A relation R_4 respecting the compatibility condition (‡) is then derived from the relations R_2 and R_3 via: $(g, m) \in R_4$ iff there exists $m_1 \leq m$ with $(g, m_1) \in R_2 \cup R_3$.

Maintaining the Compatability Condition Inserting the ordering $b \leq a$ into $\leq$ requires the insertion of set $(\uparrow a / \uparrow b) \times \{g \in G \mid (g, b) \in R_4\}$ into R_4. Such an insertion into an inverted file index is $O(nm)$ where n is the average number of entries in the inverted index in the shaded region, and m is the number of elements in the shaded region. The real complexity of this operation is best determined via experimentation with a large document sets and a large user defined hierarchy [1]. Similarly the removal of the ordering $b \leq a$ from $\leq$ will require a re-computation of the inverted file entries for elements in $\uparrow a$.

Processing new Email and Integrating user Judgements When new emails, G_b, are presented to Cem, the relation R_1 is updated by inserting new pairs, R_{1b}, into the relation. The modification of R_1 into $R_1 \cup R_{1b}$ causes an insertion of pairs R_{2b} into R_2 according to $\texttt{query}(m)$ and then subsequently an insertion of new pairs R_{4b} into R_4.

$$
\begin{aligned}
&R_{1b} \subseteq G_b \times (H \times W)\\
&R_{2b} = \{(g, m) \mid \exists\, (h, w) \in \text{query}(m) \text{ and } (g, (h, w)) \in R_{1b}\}\\
&R_{4b} = \{(g, m) \mid \exists\, m_1 \leq m \text{ with } (g, m_1) \in R_{2b}\}
\end{aligned}
$$

When the user makes a judgement that an indexed email should be associated with an attribute, m, then an update must be made to R_3, which will in turn cause updates to all attributes in the order filter of m to be updated in R_4. In the case that a client retracts a judgement, saying that an email is no longer be associate with an attribute, m, requires a possible update to each attribute, n, in the order filter of m.

4.4 Navigating the Conceptual Email Space

When the user requests that the concept lattice derived from the scale with name $s \in S$ be drawn, the program computes $\mathbb{S}_{\alpha(S)}$ from Definition 1 via the algorithm reported in [1]. In the case that the user requests a diagram combining two scales with names labels s and t, then the scale $\mathbb{S}_{B \cup C}$ with $B = \alpha(s)$ and $C = \alpha(t)$ is calculated by the program and its concept lattice $\underline{\mathfrak{B}}(\mathbb{S}_{B \cup C})$ is drawn as a projection into the lattice product $\underline{\mathfrak{B}}(\mathbb{S}_B) \times \underline{\mathfrak{B}}(\mathbb{S}_C)$.

5 Conclusion

This paper gives a mathematical description of the algebraic structures that can be used to create a a lattice-based view of electronic mail. The claim is that this structure, its implementation and operation, aid the process of knowledge discovery in large collections of email. By using such a conceptual multi-hierarchy, the content and shape of the lattice view is varied. An efficient implementation of the index promotes client iteratation.

Note for reviewers, the screen-shots and the text example introduced early in the paper need to be synchronised if the paper were acceptable to PKDD, however we ran out of time.

References

1. R. Cole, P. Eklund: Scalability in Formal Concept Analysis: A Case Study using Medical Texts. *Computational Intelligence*, Vol. 15, No. 1, pp. 11-27, 1999.
2. R. Cole, P. Eklund: Analyzing an Email Collection using Formal Concept Analysis. *Proceedings of the European Conf. on Knowledge and Data Discovery*, pp. 309-315, LNAI 1704, Springer, Prague, 1999.
3. R. Cole, P. W. Eklund, D. Walker: Using Conceptual Scaling in Formal Concept Analysis for Knowledge and Data Discovery in Medical Texts, *Proceedings of the Second Pacific Asian Conference on Knowledge Discovery and Data Mining*, pp. 378-379, World Scientific, 1998.
4. A. Fall: Dynamic taxonomical encoding using sparce terms. *4th International Conference on Conceptual Structures*. Lecture Notes in Artificial Intelligence 1115, 1996,
5. B. Ganter, R. Wille: *Formal Concept Analysis: Mathematical Foundations*. Springer, Heidelberg 1999 (Translation of: Formale Begriffsanalyse: Mathematische Grundlagen. Springer, Heidelberg 1996)
6. K. Jones: View Mail Users Manual. *http://www.wonderworks.com/vm*. 1999
7. T. Rock, R. Wille: Ein TOSCANA-System zur Literatursuche. In: G. Stumme and R. Wille (eds.): *Begriffliche Wissensverarbeitung: Methoden und Anwendungen*. Springer, Berlin-Heidelberg 2000
8. W. Schuller: *http://gmail.linuxpower.org/*. 1999
9. G. Stumme: Hierarchies of Conceptual Scales. *Proc. Workshop on Knowledge Acquisition, Modeling and Management*. Banff, 16.–22. Oktober 1999
10. F. Vogt, R. Wille: TOSCANA — A graphical tool for analyzing and exploring data. In: R. Tamassia, I. G. Tollis (eds.): *Graph Drawing '94*, Lecture Notes in Computer Sciences 894, Springer, Heidelberg 1995, 226–233
11. R. Wille: Restructuring lattice theory: an approach based on hierarchies of concepts. In: I. Rival (ed.): *Ordered sets*. Reidel, Dordrecht–Boston 1982, 445–470