MINING SPARSE TEXTS IN BUILDING PRODUCT MODELS TO ASSESS RELEVANCE FOR REUSE

Peter Demian[1] and Renate Fruchter[2]

ABSTRACT

This paper presents research on the use of text analysis to measure relevance or similarity between objects in building product models. This relevance measure is used to support design reuse from an archive of product models created during previous construction projects. First, standard information retrieval techniques are applied: the text vector model and its refinement latent semantic indexing. Next, we attempt to improve the performance of the relevance measure by considering contextual building model elements. When comparing any two building components, we attempt to improve data mining performance by considering not only the two components being compared but also components related or linked to them. The paper concludes with an evaluation and discussion of these techniques. It concludes that good retrieval results can be achieved even with sparse models.

KEY WORDS: Data mining, information retrieval, text analysis, design reuse.

INTRODUCTION

Increasingly, building models are composed of more than just the geometry of the building, but include symbolic representations in the form of attributes or relationships to other objects. Most of this additional information is textual, such as names of building components. In this paper, we investigate the use of text analysis techniques to measure relevance between objects in product models. This relevance measure is used to support design reuse from an archive of previous product models (a *corporate memory*). It is a part of the CoMem (Corporate Memory) prototype (Fuchter and Demian 2002). CoMem (Figure 1) provides an overview of the entire repository in the form of a *map of the corporate memory*, which supports the *finding* of reusable items. Once the user identifies a reusable item on the map, CoMem enables him/her to *understand* this item in context by providing details-on-demand about this item's evolution in the Evolution History Explorer and project context in the Project Context Explorer.

The Overview provides a succinct "at a glance" view of the entire corporate memory. CoMem employs the *squarified treemap* technique to display all items in the corporate memory as a series of nested rectangles. Each rectangle is colored to indicate its relevance according to the relevance measure discussed in this paper.

[1] Lecturer, Department of Civil and Building Engineering, Loughborough University, Leicestershire LE11 3TU, UK, Phone +44 (0) 1509 228541, FAX: +44 (0) 1509 223945, P.Demian@lboro.ac.uk

[2] ASCE Member, Director of Project Based Learning Laboratory, Department of Civil and Environmental Engineering, Stanford University, Stanford, CA 94305-4020, fruchter@ce.stanford.edu

The Project Context Explorer supports the designer to explore the project context of any item selected CoMem Map and related items in that project. The Evolution History Explorer enables the designer to explore the version history of any item selected from the Overview.

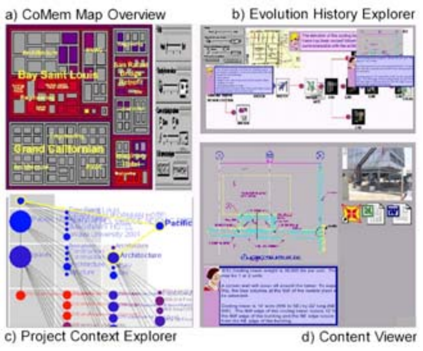


Figure 1: CoMem Modules: (a) The Overview; (b) The Evolution History Explorer; (c) The Project Context Explorer; (d) The Content Viewer displays the item being reused.

Our observations of the reuse process as it occurs in current practice show that it consists of two steps: (1) *finding* reusable items, and (2) *understanding* these items in the context in which they were originally created. CoMem supports reuse in both steps. This paper addresses this question: how can the relevance between any two objects in the corporate memory be quantified? Based on the tasks of *finding* and *understanding* that this measure supports, *relevance* can be defined as follows:

For any two corporate memory objects A and B, object B is relevant to object A if:

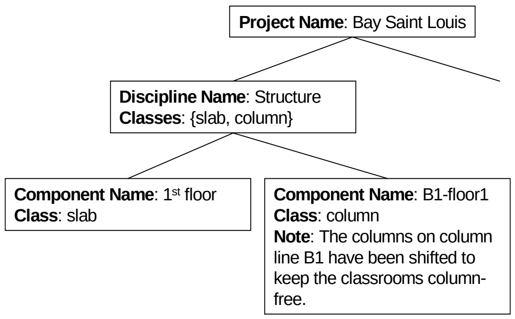
- The designer is working on object A and object B is potentially reusable. Or:
- The designer is considering reusing object A and object B is related to object A, such that knowledge about object B helps the designer to understand object A.

The approach taken in generating relevance measures is to use text analysis. This is effective because the product model objects are annotated with text strings but have otherwise little formal data. On the other hand this is challenging because the text strings are *much shorter* than those normally used for text analysis and retrieval.

CONVERTING COMEM OBJECTS TO DOCUMENTS

The CoMem schema is a three-level hierarchy composed of *projects*, *disciplines*, and *components*. As team members collaboratively develop the CAD model for the *project*, they communicate and collaborate by creating *discipline* and *component* objects, and linking them to geometrical entities from the CAD model. A *discipline* object encapsulates a portion of the design from a particular point-of-view, a discipline (e.g. architecture), a subsystem (e.g. HVAC) or a general issue (e.g. cost). A *component* object is a design feature over which the design team collaborates. Component objects can also be linked to notes exchanged by the designers or to external files or documents.

Each *project* object has a name. This becomes the text of the project document. Each *discipline* has a name and list of classes that constitute the vocabulary or ontology of that particular design perspective. The user is free to use any text string as a class; there is no universal vocabulary from which the class list is drawn. Each *component* object has a name, and belongs to/is an instance of one of the classes in its parent discipline object. A component object has one or more notes linked to it. Figure 2 gives examples of CoMem objects. Each object is converted into a document by concatenating all of its text elements.


Project Name: Bay Saint Louis
Discipline Name: Structure
Classes: {slab, column}
Component Name: 1st floor
Class: slab
Component Name: B1-floor1
Class: column
Note: The columns on column line B1 have been shifted to keep the classrooms column-free.

Object Type	Object Text	Document Text
Project	Project Name: Bay Saint Louis	bay saint louis
Discipline	Discipline Name: Structure Classes: {slab, column}	structure slab column
Component (without note)	Component Name: 1st floor Class: Slab	1st floor slab
Component (with note)	Component Name: B1-floor1 Class: Column Note: The columns on column line B1 have been shifted to keep the classrooms column free.	b1-floor1 column the columns on column line B1 have been shifted to keep the classrooms column free

Figure 2: Typical CoMem objects.

EVALUATING THE MEASURES OF RELEVANCE

Tests were conducted on a pilot corporate memory consisting of 10 project objects, 35 discipline objects, 1036 components, giving a total of 1081 objects that were converted into documents. Common words that add little meaning (*the*, *you*, etc.) were filtered out prior to indexing. The experiments were repeated with and without stemming. Stemming is the process whereby distinct terms are reduced to their common grammatical root. Stemming was performed using Porter's algorithm (Baeza-Yates and Ribeiro-Neto 1999).

Evaluation of relevance measures was carried out in the classic information retrieval manner. Given a set of queries, and a set of documents for each query judged to be relevant by a human expert, the results returned by the relevance measure were compared to those provided by the expert using measurements of recall versus precision. In the case of CoMem, a query is a specific object from the corporate memory, and the "hits" returned are other objects that are relevant to the query object. For each query, precision was measured at 11 standard recall levels from 0 to 1.0 in increments of 0.1 using the interpolation rule described in section 3.2.1 of Baeza-Yates and Ribeiro-Neto (1999). The precision measurements at those recall levels were averaged for entire sets of queries.

Three sets of queries were considered separately: queries where the query object was a project (8 queries), a discipline (18 queries), and a component (6 queries). Those were considered separately because those three types of documents differ in how much text they contain, and how representative that text is of the actual content of the object.

COMPARING DOCUMENTS USING THE VECTOR MODEL AND LSI

The starting point is to use the text vector model (Salton and Buckley 1998). For a collection of N documents and a total of n index terms across the entire collection, we build a document matrix of size $N \times n$. For each document-term element in this matrix, we compute a weight $w_{i,j}$ which represents the occurrence of term k_i in document d_j. Each document is represented as a vector in the high-dimensional space of index terms. The similarity or relevance between two documents is quantified by calculating the cosine of the angle between the two document vectors. Experiments were conducted with three different term-weighting systems (see Demian and Fruchter 2005 for detailed descriptions): (1) Binary weights, (2) Tf-idf weights, and (3) Log-entropy weights

Table 1 gives measurements of precision averaged over the 11 standard recall levels for each of the query sets and term-weighting schemes (with and without stemming).

Table 1: Mean precision over the 11 standard recall levels using vector model comparisons.

		No Stemming (mean precision)	**Stemming (mean precision)**
PROJECT QUERIES	**Binary**	0.31	0.31
	Log-entropy	0.31	0.31
	Tf-idf	0.31	0.31
DISCIPLINE QUERIES	**Binary**	0.39	0.45
	Log-entropy	0.40	0.40
	Tf-idf	0.37	0.41
COMPONENT QUERIES	**Binary**	0.49	0.49
	Log-entropy	0.56	0.57
	Tf-idf	0.60	0.63

We can make the following observations from the results of

Table 1:

- **Comparison of project, discipline, and component queries.** Overall, the vector model gives the best results for component queries and the worst results for project queries. Both project and component documents contain only a few terms. In the case of component objects, these few terms are fairly representative of the content of the object.
- **Term-weighting.** For discipline queries binary weighting performs the best. This is because the other two term-weighting systems reduce the weights of the class terms since they occur frequently over the entire collection, even though the class terms give a better indication of the content of the discipline than the discipline name.
- **Stemming.** For project queries stemming makes little difference. For discipline queries stemming consistently gives a considerable improvement. This is because some discipline documents use singular nouns for the class list (beam, column, slab) whereas others use plural nouns (beams, columns, slabs).

Latent semantic indexing (LSI) is a refinement of the simple vector model that addresses the problems of synonymy (using different words for the same idea), polysemy (using the same word for different ideas). LSI uses singular value decomposition to give an approximation of the document matrix. The *number of factors* indicates the amount of approximation introduced into the model. For large numbers of factors, the LSI model converges to the exact vector model. The claim is that this approximation models the implicit higher order structure in the association between terms and concepts (Landauer and Dumais 1995).

For example, if the two terms *lift* and *elevator* frequently co-occur within documents in the collection or if they frequently occur in the same contexts, then an LSI query for *lift* would also return documents with only the term *elevator*, an association that would be overlooked by the simple vector model.

We conducted several tests with LSI to determine whether it could offer an improvement over the performance obtained with the simple vector model. The data from the LSI runs for component queries is shown in Figure 3 (the "No Corpus" lines). Each line shows the mean precision over the 11 standard recall levels obtained by running LSI using tf-idf weights and varying the number of factors.

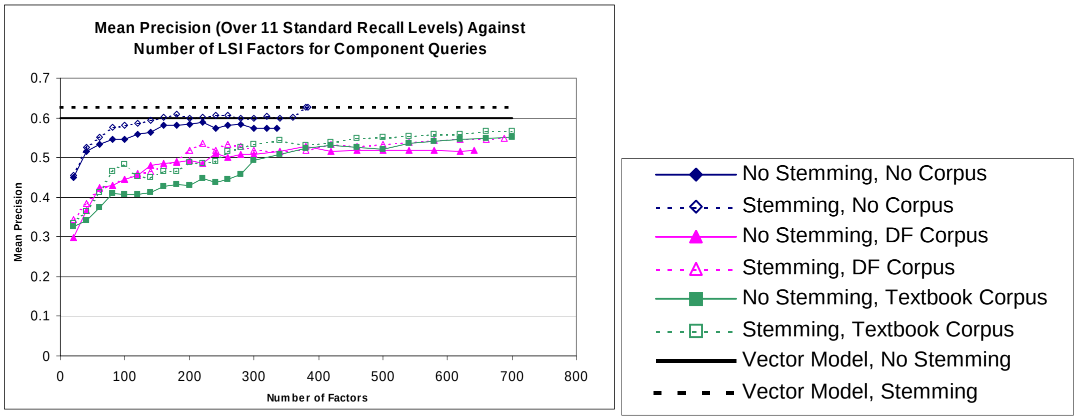


Figure 3: Mean precision over 11 standard recall levels using LSI for component queries.

For discipline and component (Figure 3) queries, LSI performs worse than the vector model for small numbers of factors, but gradually converges to the performance of the vector model as the number of factors is increased. For project queries (not shown here), LSI gives a modest improvement over the simple vector model when 300-350 factors are used. As with the vector model runs, stemming consistently improves performance.

It is not surprising that LSI does not offer any significant improvement over the simple vector model. LSI is claimed to work best when the collection of documents is large and the documents are rich in keywords, which helps to infer relationships between terms based on their co-occurrence. In our case, the document matrix is sparse, with many of the documents consisting of only two or three terms.

One way to address this problem is to add a set of "rich" documents with the CoMem documents. The rationale behind this is that if these additional documents are numerous enough and rich enough (i.e. contain many keywords), then LSI should be able to infer relationships between terms because they frequently co-occur in the additional documents. These inferred relationships should in turn improve data mining performance when comparing CoMem objects. We tested this approach with two sets of additional documents:

- A collection of discussion forum messages exchanged by the design teams working on the projects in our experimental corporate memory. Each individual message was treated as a single document.
- A set of articles from reference handbooks for professional structural designers and construction managers. Each paragraph was treated as a single document.

It can be seen from the "corpus" lines in Figure 3 that this was not successful. In both cases, adding the corpus further reduced the performance of the LSI runs.

CONTEXT-SENSITIVE COMPARISONS: CONCATENATING DOCUMENTS

As noted above, CoMem documents do not belong in a flat collection but are hierarchically structured. It would make sense therefore to consider an object's relatives within the hierarchy when making comparisons involving that object. Before we try a more sophisticated approach involving tree matching, we will test whether the approach of simply concatenating documents offers any improvement in data mining performance. When converting an object to a document, we will include the text from that object's ancestors and/or descendants. Specifically, we will try the following options:

- **Concatenating descendants**: The retrieval performance for project queries is fairly weak. It has already been noted that the short label given to a project object is usually a poor indication of the content of the project. Concatenating the texts of all the project's descendants (disciplines and components) to the project document will result in a much longer text, which may improve data mining performance. The same is also true of discipline objects.
- **Concatenating ancestors**: For component and discipline objects, retrieval performance might be improved because objects belonging to similar parents (to the query item) will be ranked above those coming from unrelated parents.
- **Concatenating both descendants and ancestors**: Only discipline objects have both descendants (components) and ancestors (a project).

The results (mean precision over the 11 standard recall levels) from repeating the simple vector model analysis with concatenated documents are shown in Figure 4. Separate results are shown for the project, discipline, and component query sets.

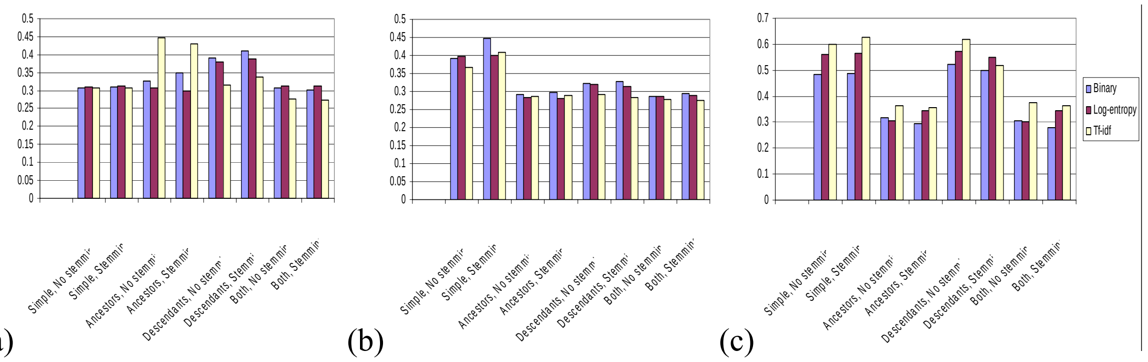


Figure 4: Mean precision over 11 standard recall levels for (a) project queries, (b) discipline queries, and (c) component queries using different forms of concatenation.

For project queries, concatenating descendants gives improved performance, particularly when stemming is combined with binary weights. For discipline queries, any form of concatenation causes a decrease in performance. As noted earlier, the text of discipline objects (which consists of a name and a list of classes) already provides a good representation of the content of the discipline object. For component queries, concatenating ancestors and concatenating both descendants and ancestors both cause a decrease in data mining performance. This can be explained by the fact that the ancestors of component objects (their disciplines and projects) were not given much consideration by the human experts when making human judgments of relevance.

CONTEXT-SENSITIVE COMPARISONS: TREE MATCHING

Above, we discussed the relatively simple method of concatenating the text from ancestors and descendants to take account of contextual objects. Here, we try a more elaborate method inspired by the concept of tree isomorphism. Two graphs *G* and *G'* are isomorphic if we can label the vertices of *G* to be vertices of *G'*, maintaining the corresponding edges in *G* and *G'*. The basic idea is that when comparing two objects, we try to find the best possible one-to-one match between those objects, their ancestors and their descendants. The closeness of this match becomes our relevance measure. We will use the vector model measurements of relevance between any two nodes as a *simple* measure of relevance and aggregate those into *compound* measures of relevance that take account of the relatives (ancestors and descendants) in the tree. There are six possible types of comparisons: (1) Component-Component comparisons, (2) Discipline-Discipline comparisons, (3) Project-Project comparisons, (4) Project-Component comparisons, (5) Discipline-Component comparisons, and (6) Project-Discipline comparisons

For brevity, only the second type of comparison will be described here. The remaining comparisons are described in more detail elsewhere (Demian and Fruchter 2005). To compare two disciplines, the *simple relevance* between disciplines d_i and d_j is taken as r_{d_i,d_j}, where r_{d_i,d_j} is the simple vector model relevance between them.

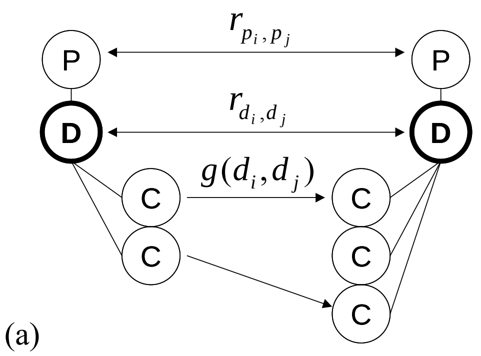
The *compound relevance* $\hat{r}_{d_i,d_j}$ between disciplines d_i and d_j is:

$$\hat{r}_{d_i,d_j} = w_c g(d_i, d_j) + w_d r_{d_i,d_j} + w_p r_{p_i,p_j}$$

where r_{p_i,p_j} is the simple relevance between d_i's parent project p_i and d_j's parent project p_j, and $g(d_i, d_j)$ is some aggregated function of the simple relevancies between discipline d_i's m component children and discipline d_j's n component children. We can say without loss of generality that $m \leq n$.

The best way to think of $g(d_i, d_j)$ is as providing some aggregated measure of relevance between d_i and d_j based on simple relevancies between their children components. There are $2mn$ possible directed edges between the set of d_i's m component children and the set of discipline d_j's n component children such that each edge spans the two sets. Each edge has a relevance value associated with it, which is the simple vector model relevance between the two components connected by the edge. We would like to find some subset of those edges which best represents the relevance between those two sets of components, and calculate the mean relevance associated with this subset. This becomes the value of $g(d_i, d_j)$.

In the spirit of isomorphic tree matching, we might choose a subset of edges such that each component in the smaller set has one outgoing edge and each component in the larger set has no more than one incoming edge. In other words, we will try to find the best one-to-one mapping from the components in the smaller set to the components in the larger set. There are P_m^n possible mappings. Each possible mapping can be represented by a set of m edges, and can be evaluated by taking the mean relevance of those edges. The value of $g(d_i, d_j)$ is the mean relevance of the best possible mapping. This is shown in Figure 5 (a).



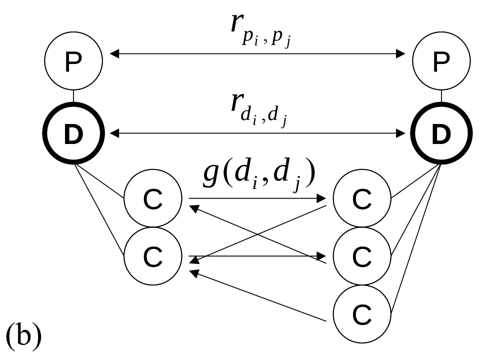
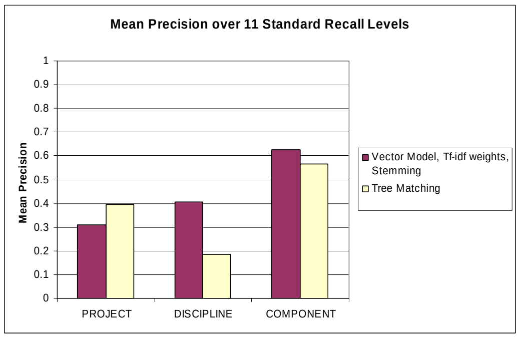


Figure 5: Compound discipline-discipline comparisons. (a) More accurate method but more complcomputationally-demanding (b) Less accurate method but simpler

This method is too computationally demanding for large sets of components. Finding the best mapping means evaluating every possible mapping. For *m* and *n* approximately equal, the number of possible mappings is of the order of *n!*. Larger CoMem disciplines can have more than 20 components; $20! = 2.4 \times 10^{18}$. Therefore, for larger sets of components, an alternative method is used. According to this method, a subset of edges is chosen such that the highest-relevance outgoing edge for each component in both sets is included. Subsets of this type will have *m*+*n* edges, and can be evaluated by calculating the mean relevance of those edges, which is taken as the value of $g(d_i,d_j)$ (Figure 5 (b)) The main advantage of this method is that it finds a local optimum with very little search. The main disadvantage is that it does not enforce a one-to-one mapping and so is arguably less accurate.

Figure 6 (a) shows the retrieval performance for both the simple vector model using stemming and tf-idf weights and the tree matching method where those same vector model comparisons are aggregated into compound relevance measures that take account of contextual objects.

(a) 

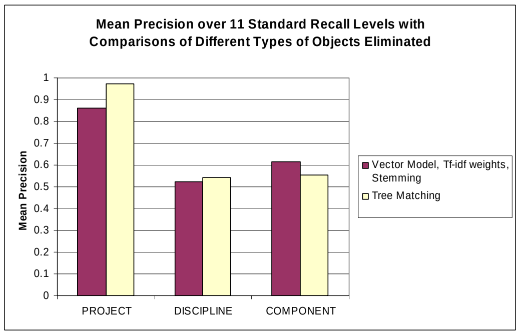
(b) 

Figure 6: (a) Mean precision over 11 standard recall positions for project, discipline, and component queries using simple vector model and tree matching, (a) for all comparisons in query sets (b) with comparisons between different types of objects eliminated.

Tree matching out-performs the vector model for project queries, but not for discipline or component queries. As noted earlier, project documents consist only of the project name, which is not highly indicative of the type of project, and so comparing a project to other objects based on that project's disciplines and components makes sense. However the retrieval performance achieved for project queries using the tree matching method is comparable to that achieved by concatenating documents.

On the other hand, discipline documents include the discipline name (e.g. “structural system”) as well as a list of classes (e.g. beam, column, slab). In this case, direct vector model comparisons using that discipline document are adequate. A similar argument can be made for component queries. The text of a component document is a good representation of that component and so there is not much need to compare contextual objects.

One major disadvantage of the tree matching method is that when comparing two objects of two different types, those two objects are never directly compared to one another. For example, when comparing a discipline with the text “slabs {precast, post tensioned, composite}” to a component with the text “first floor slab”, those two texts are never directly compared using the vector model. Instead the discipline’s children are compared to the component, and the component’s parent is compared to the discipline. To investigate the extent to which the poor performance of the tree matching method is due to this effect, we reran the evaluation of the results with all such comparisons eliminated. Only comparisons between objects of the same type are included in the evaluation. Figure 6 (b) shows the retrieval performance of the two methods when this restriction is enforced. The performance of the tree matching method is comparable to that of the vector model for all three types of queries. For discipline queries the tree matching method fares much better than before.

DISCUSSION AND CONCLUSIONS

The most striking outcome is that there is no clear strong winner amongst the various studied alternatives. Furthermore, more complex relevance measures do not necessarily give better results than simpler ones.

- When comparing different term-weighting systems, the simplest system (binary weights) often performed just as well as the most complicated (log-entropy).
- When taking context into account, the simple method of concatenating documents performed well and often better than the more complicated tree matching method.
- When attempting to address the problem of synonyms, latent semantic indexing doesn’t perform better than the simple vector model, whereas the relatively primitive dimensionality reduction achieved by stemming consistently performed better than both LSI and the unstemmed vector model.

The best overall performance is achieved using tf-idf weights in conjunction with stemming, and this is what is used by CoMem. For component and discipline queries, simple vector model comparisons are used. For project queries, the context (i.e. the discipline and component objects belonging to the project) needs to be taken into account, and concatenating the descendants of the project object onto the text of the project document is a simple and effective way of achieving this.

The tree matching approach as implemented here does not provide sufficiently improved performance to justify the additional computation it entails. It is worth developing in future research. In particular the choice of weights, w_p, w_d, and w_c, needs further investigation. Currently, the weights are chosen based on simple heuristics.

LSI did not deliver the improvement in retrieval performance usually reported by its supporters. The situations for which LSI has been shown to be effective are significantly different than ours. In published cases where LSI out-performs the simple vector model the mean document size is significantly larger than in the CoMem corpus. It has been noted earlier that LSI makes statistical associations between synonyms because they repeatedly co-occur within the same document or they repeatedly occur in similar contexts. The smaller the number of documents, and the more sparse the documents, the thinner the statistical sample from which LSI can make such associations. In the case of CoMem, the documents were simply too short, even with the addition to the corpus of discussion forum messages or technical articles. Wiemer-Hastings (1999), using sentences as units of discourse (i.e. a single sentence is a document), also reports poor performance of LSI. Rehder et. al. (1998), using LSI to grade student essays, report that if only the first 60 words or less of the student essay are used then LSI performs poorly.

Finally, a comment may be made about the generalizability of the results. CoMem is hierarchical, as are most information systems: from the ubiquitous file systems of modern computers to more specialized information schemas in the construction industry such as IFC, AECXML, and so on. The short names given to CoMem objects are comparable to the names given to files and folders, or the names of objects in schemas such as IFC. Caldas et. al. (2002) have shown that information retrieval techniques and data mining such as the ones described here can be used to automatically link construction documents to IFC components.

The fact that relatively good quality retrieval results were achieved with such short texts (albeit not with LSI) is encouraging. Applying the vector model to such short texts and showing it to be effective is an important contribution of this research. Attempts to address this problem of short undescriptive texts by concatenating related documents or using tree matching are further contributions.

REFERENCES

Baeza-Yates, R., Ribeiro-Neto, B., (1999), Modern Information Retrieval, Addison-Wesley, Harlow, England.

Caldas, C. H., Soibelman, L., and Han, J., (2002). “Automated Classification of Construction Project Documents.” Journal of Computing in Civil Engineering, 16(4), 234-243.

Demian, P. and Fruchter, R., 2005, “Measuring Relevance in Support of Design Reuse from Archives of Building Product Models.” ASCE Journal of Computing in Civil Engineering, volume 29, issue 2, pp. 119-136

Fruchter, R., and Demian, P. (2002). “CoMem: Designing an interaction experience for reuse of rich contextual knowledge from a corporate memory.” Artificial Intelligence for Engineering Design, Analysis and Manufacturing (AIEDAM), 16(3), 127–147.

Landauer, T. K., and Dumais, S. T. (1995). “A Solution to Plato’s Problem: The Latent Semantic Analysis Theory of Acquisition, Induction and Representation of Knowledge.” Psychological Review, 104, 211-240.

Rehder, B., Schreiner, M. E., Wolfe, M. B. W., Laham, D., Landauer T. K., Kintsch, W., (1998). “Using Latent Semantic Analysis to assess knowledge: Some technical considerations.” Discourse Processes, 25, 337-354.

Reiner, K., and Fruchter, R. (2000). “Project Memory Capture in Globally Distributed Facility Design.” In Computing in Civil and Building Engineering, Proceedings of the Eight International Conference (ICCCBE-VIII), 820-827.

Salton, G., and Buckley, C., (1988). “Term-weighting approaches in automatic text retrieval.” Information Processing & Management, 24(5), 513-523.

Wiemer-Hastings, P., (1999). “How Latent is Latent Semantic Analysis?” In *Proceedings of the 16th* International Joint Conference on Artificial Intelligence, Stockholm, Sweden, pp 932-937.