Abbreviation Expansion in Schema Matching and Web Integration. *

L. Ratinov, E. Gudes
Computer Science Department, Ben Gurion University.
{lev-arie,ehud}@cs.bgu.ac.il

Abstract

Schema matching is a problem of finding correspondences, particularly equivalence relationships across schemas. The problem has a particular significance in integrating web repositories, as distributed databases over the web becomes increasingly popular. Most of the existing prototypes use schema level lexical information for schema matching. However, most of them perform rather poorly on real-world problems due to the abundance of abbreviations in real-world schemas. For example, none of the lexical matchers we tested would recommend a mapping of 'cnum' to 'cid', while 'customer number' and 'customer ID' are matching entities. In this work we propose a method for abbreviation expansion in schemas that facilitates lexical schema matching.

1 Introduction

Schema matching [3, 1, 2, 4, 5] is a problem of finding correspondences, particularly equivalence relationships across schemas. Originally motivated by relational database integration problem, schema matching became increasingly significant in the last decade, having applications in E-business, semantic query processing and query interfaces integration on the web. For example, in [3], Schema Matching was used to integrate databases held by many independent contractors that worked for the U.S. Air Force. Many of the contractor development-teams no longer existed, the databases were poorly documented, and there were several different data models in the database collection. Some databases were relational, some were extended entity-relational, and some were hierarchal. This lead to problems of redundant data capture and inconsistency, making the maintenance of the data increasingly expensive. Since the schema of the U.S. Air Force databases is constantly changing, an automatic integrating tool is clearly needed.

*Partially supported by the Lynn and William Frankel Center for Computer Sciences

More formally, schema matching is performed with a Match operator that finds similarity relationships between powersets of elements of the two schemas. For example the Match operator applied on the schemas: Cust(cnum, cname, address) and Customer(CustName, CustID, st., city) results in:

{Cust.cnum} ≃ {Customer.CustID},
{Cust.cname} ≃ {Customer.CustName}
{Cust.address} ≃ {Customer.st ,Customer.city}

Most of the existing prototypes ([3, 4, 5]) perform rather poorly on real-world problems due to the abundance of abbreviations in real-world schemas. For example, CUPID, a schema matching tool developed under Microsoft research [5], used the following methodology for lexical similarity. (1)*Tokenization:* the names were parsed into tokens using different heuristic methods, such as upper-lower case transitions. For example, the lexeme 'CustID' should be tokenized into {Cust,ID}. (2)*Expansion:* the abbreviation were expanded. For example, {Cust,ID} turned into {Customer, Identity}. (3)*Elimination:* 'stop tokens' are eliminated. (4)*Tagging:* tokens known to be related to one of the predefined categories were tagged with that category. For example, tokens such as 'Cost' and 'Value' were tagged with the concept 'Money'. The tagging scheme is used as a sort of thesaurus for measuring semantic similarity between the tokens.

Like most of the existing prototypes, CUPID relies on the availability of a complete thesaurus or a tool for abbreviation expansion. In the case of CUPID, the prototype assumes existence of an XML file with a list of abbreviations and their expansions. Likewise, it assumes a list of tokens mapped to concepts. For example, the small default thesaurus provided with the CUPID prototype tags 'e-mail', 'phone' and 'province' with the concept 'address'. Another assumption of CUPID is the existence of a thesaurus that measures semantic similarity between the concepts. For example, the default CUPID thesaurus contains the pair: [number≅code, similarity=0.8]. Clearly these assumptions are unrealistic for the real-world problems. In our experiments with real-world schemas CUPID was unable to expand (and indeed match) abbreviations such as 'MFC' for

Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence (WI'04)
0-7695-2100-2/04 $ 20.00 IEEE

IEEE COMPUTER SOCIETY

'Manufacturer Code' or 'FFP' for 'Frequent Flyer Plan' resulting in poor mapping. We also note that abbreviation expansion need not be unique. For example, 'POID' could be expanded to: 'Post Office ID', 'Police Officer ID' or to 'Purchase Order ID'.

As it became evident in the discussion above, there are two main challenges in using lexical information in schema matching. They are abbreviation expansion and disambiguation. To demonstrate these problems we consider a small example. Suppose we are given a schema: $Adm(pid, pd, date)$. The first challenge in processing this schema is to create a list of candidate expansions for every element in the schema. In our example, such expansions could be: 'Adm'$>\{$Admission,Administraction,Admiral$\}$, 'pid'$>\{$problem id, patent id, patient id$\}$, 'pd'$>\{$patient diagnosis, patent description, problem description$\}$, 'date'$>\{$date$\}$. The second challenge, disambiguation, is to select the most likely candidates for every abbreviation. For example 'Admission(patient id, problem description, date)' and 'Administration(patent id, patent description, date)' are such examples. On the other hand, the expansion 'Admiral(problem id, patent description, date)' clearly makes no sense. Figure 1 demonstrates the global view for schema expansion and disambiguation. This paper discusses the challenges and the methods for abbreviation expansion. Abbreviation disambiguation is discussed in [16].

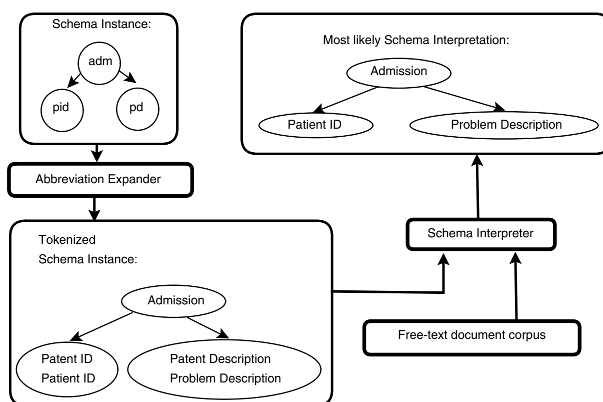

Figure 1. Global view on schema expansion.

2 Abbreviation expansion problem.

2.1 Abbreviation expansion in schemas

The problem of automatic abbreviation expansion has received a lot of attention ([7, 8, 6, 9, 10]) in the context of automatic construction of special-purpose dictionaries and other applications, such as information retrieval, text transformations, etc.). Static abbreviation and acronym dictionaries also exist, and are available from a number of sources, some dedicated to a particular domain [13, 14], some aim to be general purpose dictionaries, [11, 12]. Online portals for abbreviations containing a large number of links to different acronym dictionaries are also available [15]. However these tools, as we show later, do not provide a full answer to abbreviation expansion problem in the particular context of lexical schema matching. This section first discusses the general aspects of abbreviation expansion, then the particularity of this problem applied to lexical schema matching.

The problem of abbreviation expansion in the context of schema matching is: given an abbreviation abr as input, output a list of all possible expansions for abr. In abbreviation expansion, we cannot rely on some static abbreviation dictionary, since database schemas abbreviations are constructed very dynamically, and are unlikely to appear in any such dictionary. A popular approach to respond to this challenge ([7, 8, 6, 9]) is to crawl the Web (or a static text corpus), looking for patterns of the form : <long form, (abr. form)>. The intuition behind this approach is that when the abbreviations are used for the first time, their meaning is explained in brackets (or some other popular pattern) in the immediate proximity to the abbreviation. We cannot use this assumption, however, because unlike abbreviations in any other domain which denote important and repeating domain concepts, schema abbreviations are sometimes created only to save space, from phrases that would not be abbreviated in a normal context. We must also keep in mind that abbreviations in database schemas have different abbreviation rules, for example, they tend to be longer (use more letters of the long form) then abbreviations in the other fields. Thus we are unlikely to find the abbreviated form in any text corpus. On the other hand, we can sacrifice precision for recall because unlike the traditional abbreviation dictionaries, our abbreviation expansion tool does not have to give only the absolutely correct set of expansions. We should merely give (a possibly large) set of candidate expansions for the input abbreviation and later apply linguistic tools that choose the best expansion candidate for every schema element looking at the *global* schema interpretation. We call this stage *disambiguation* and propose solutions in [16].

3 Abbreviation Patterns

Our task is: given an abbreviation, to create a list of all of its possible expansions. We follow the intuition of [7, 8, 6, 9, 10] that schema elements' names come form meaningful and even popular phrases that are likely to appear in a text document. We further enhance this approach by performing shallow parsing and stop words elimina-

IEEE COMPUTER SOCIETY

tion on the corpus. For example, the following text: *"You can mail contribs to Danny Thomas, Post Office Box 7599, Chicago. So, if anybody possibly solicits by phone, make sure you mail the dough to the above."* is transformed to: *"[You] [mail] [contribs] [Danny Thomas], [Post Office Box], [Chicago]. [anybody] [possibly] [solicits] [phone], [make sure] [you] [mail] [dough] [above]."* To further increase precision, we could consider only the noun phrases as expansion candidates. To increase the recall, we could consider each candidate expansion phrase both with and without stop words. However the main goal of this paper is to show the feasibility of our approach, which performed satisfactorily even without these enhancements.

We also note that abbreviations tend to be created in patterns that repeat themselves. A list of several such patterns (borrowed from [9] and [10]) is given in Table 1. Before suggesting any concrete solution, we have to define a model for abbreviation creation from its long form (note that the abbreviation could be created in different ways from its long form, thus the process need not be unique). For example, [9] uses feature vectors to capture some important aspects of the abbreviation process. Each entry in the nine-dimensional vector denotes a manually selected feature believed important for the process of creating abbreviations. Some of such features are: 'percent of letters in abbreviation in lower case', 'percent of letters aligned on a syllable boundary', 'percent of letters in the abbreviation that are aligned', etc.... The probability of a phrase p being the long form of the given abbreviation abr, is scored based on the best induced feature vector $induce(p, abr)$ using statistical methods (linear regression).

[6] and [10] attempted to develop an 'abbreviation rule grammar'. Typical rules (or grammar derivations) are: 'take the first letter of a word', 'convert the word to a number', 'take some interior character of the word', etc.... Each rule is assigned a weight, and the probability of phrase p being the long form of the given abbreviation abr, is scored based on the best induced sequence of derivations from p to abr. (Scoring a sequence of derivations based on scores for single derivations is not a trivial problem without assuming rule application independence, another problem of this approach is the computational complexity of finding all the possible derivations.)

The main problem of these approaches is that the set of grammar rules, and the feature vector dimensionality are both hardcoded and fixed. We suggest a more flexible system, that can learn new abbreviation patterns using Neural Networks. We relax the feature vector approach to 'abbreviation pattern approach'. The idea behind the 'abbreviation patterns' is that we want to represent the abbreviation process, while on the one hand, keeping enough information for inducing previously unobserved abbreviation templates, and on the other hand, in a way that would allow us to generalize and avoid overfitting.

The notion of 'abbreviation pattern' is best described by an example. An abbreviation from 'Department' to 'Dpt' induces two abbreviation patterns: 'CxCxxxxxxC' and 'CxCxxCxxxx', which mean that there were three consonant letters in appropriate locations chosen for the abbreviation. This example also shows that a pair <abbreviation,phrase> can induce several different abbreviation patterns.

More formally, abbreviation pattern is a word over the alphabet {'C','V','_','x'}, where: '_' means 'space' (words are delimited with a spaces), 'V' means that a vowel letter was chosen, 'C' means that a consonant was chosen, 'x' means that the letter was filtered out from the phrase in abbreviation construction.

This representation, while avoiding overfitting, keeps a lot of information that can be inferred about the abbreviation process, such as: (*)What were the locations of the letters in the phrase (first letters of words, last letters of words ...), (*)Was the letter chosen for word representation a consonant or a vowel (consonants are used more often). (*)How long was each word of the abbreviated phrase, and how many characters it contributed to the abbreviation, etc.

We could further enhance the expressive power of the abbreviation patterns by: (*)Expanding the vocabulary to {'C','c','V','v','_','x'} capturing whether the mapped letter was capitalized in abbreviation. (*)Marking syllables boundaries. (*)Marking whether a filtered letter was a consonant or a vowel. (*)Including a part-of speech tag before the words. We could even represent the abbreviation process as a pair <long form, indices of letters chosen for abbreviation>, which would keep lossless information on abbreviation process. However, as we increase the expressiveness of abbreviation patterns, we increase the 'abbreviation rules' search space, make the learning process more difficult and risk overfitting.

Note that abbreviation patterns as they are defined now, do not handle the cases when the abbreviated form contains letters that do not appear in the long form. Neither can they handle the cases when words or letters are replaced by digits (see table 1). Extending the expressiveness of abbreviation patterns is subject to further research.

We define an abbreviation pattern induced by a pair <phrase,abbr> as 'legal' if the pattern indicates that the phrase could be the long form of the abbreviation. Else, the pattern is considered illegal. For example, abbreviation pattern 'Cxxxxxxx_Cxxxxx' is a legal pattern, because it indicates that the first consonant letters of every word were chosen for the abbreviated form. On the other hand, the pattern 'xVxxxxVxxx_xxxx_xxxx' is illegal pattern, because it appears unlikely that only the first word in the long form contributed two internal vowels to the abbreviated form. A pair <phrase,abbr> may induce several abbreviation pat-

Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence (WI'04)
0-7695-2100-2/04 $ 20.00 IEEE

IEEE COMPUTER SOCIETY

Abbrv.	Definition	Description
VDR	vitamin D receptor	The letters align to the beginnings of the words.
PTU	propylthiouracil	The letters align to a subset of syllable boundaries.
JNK	c-Jun N-terminal kinase	The letters align to punctuation boundaries.
IFN	interferon	The letters align to some other place.
ATL	adult T-cell leukemia	The long form contains words not in the abbreviation.
CREB-1	CRE binding protein	The abbreviation contains letters not in the long form.
beta-EP	beta-endorphin	The abbreviation contains complete words.
W3C	World Wide Web Consortium	Letters or words are replaced by digits

Table 1. Typical patterns for abbreviations

terns. If at least one of them is legal, we consider the pair legal. For example, the pair <Artificial Intelligence, AI> induces 7 abbreviation patterns. It is a legal pair, since besides illegal patterns, such as: *VxxVxxxxxx_xxxxxxxxxxxx (AxxIxxxxxx xxxxxxxxxxxx)* it contains a legal pattern: *Vxxxxxxxxx_Vxxxxxxxxxxx (Axxxxxxxxx Ixxxxxxxxxxx)*.

4 Recognizing Legal Abbreviation Patterns

We implemented and compared two tools for classifying abbreviation patterns to legal and illegal ones: **A hard-coded heuristic tool** that assigns weight to a pattern. For every mapped letter the weight is $0.5^{offset-from-word-start}$ and for every filtered letter, the score is $-0.5^{offset-from-word-start}$. This simple heuristic is based on the intuition that in schema abbreviations many of the letters of the long form are kept in the abbreviation, with priority given to the letters that start words. For example, a pattern 'CxVxxx CxCxx' would score $0.5^0 - 0.5^1 + 0.5^2 - 0.5^3 + 0.5^4 - 0.5^5 + 0.5^0 - 0.5^1 + 0.5^2 - 0.5^3 + 0.5^4 = 1.09$. A pattern scoring over 0.2 is considered to be legal. **A neural network (NN) Learning tool.** The other tool is a NN for pattern classification. The advantage of the hard-coded algorithm is that it's simple, fast and quite effective. The NN on the other hand can learn to recognize unpredicted patterns. In general, we reduce the problem of abbreviation expansion to supervised pattern classification. We chose NN for our machine learning method because it is a popular choice for pattern recognition learning with autonomous feature selection and good generalization capabilities.

5 Neural Networks classification.

We selected a simple feed-forward multilayered network structure, with backpropagation learning algorithm. With 60 input elements, the NN is able to classify patterns of length up to 60 letters. If the output neuron is activated, the pattern was considered legal. The network also contained a hidden layer of 15 elements. The positive and the negative training sets for the network learning were created using a hard-coded probabilistic algorithm that worked similarly to the hard-coded classification tool discussed in section 4.

The negative training set contains *random patterns, missing-word patterns* and *bad-segment patterns*. Random patterns are just random collections of symbols over V,C,_,x. They are highly unlikely represent abbreviations, so they are added to the negative training set. Examples of such patterns are: 'CVxx__CxxVC_Cx_V_' and 'xxCVC__C_Vxx_xx_CV'. Missing-word patterns represent phrases where one of the words of the phrase did not contribute letters to the abbreviation. These patterns are considered illegal, even though the rest of the phrase might have been a legal abbreviation. Examples of such patterns are: 'CVxxx_xxxxx_CCxxxx' and 'xxxxx_Cxxxxx'. Bad-segment patterns represent phrases where all the words in the phrase contribute letters to the abbreviation, but at least one of the words contributes letters in an illegal fashion. Even single such word is enough to make the pattern illegal. Examples of these patterns are: 'Cxxxxx_xxxxxC_Vxxxx' and 'xxxxxC_Cxxxxxx'.

The legal patterns training set was characterized by letters being mapped closer to words beginnings and the mapped letters being more often consonants then vowels. Examples of such patterns are: 'CCVxxxxx', 'Cxxxxx_CxCxxxx', 'CxVxxx_VVxxx_xCxxxx'. The next section discusses the results of our neural network and hard-coded expansion tool.

6 Results and Conclusions

To check the feasibility of our abbreviation-expansion tool, we expanded a given abbreviation by scanning the (shallow-parsed) Brown corpus[1] and comparing the lists of

[1] The Brown corpus consists of one million words of American English texts printed in 1961. The texts for the corpus were sampled from 15 different text categories to make the corpus a good standard reference. Today, this corpus is considered small, and slightly dated.

IEEE COMPUTER SOCIETY

candidate expansion phrases given by the hardcoded tool and the NN-learning tool. The neural network we worked with was a (60,15,1) perceptron network, trained on 1800 automatically created training samples.

The ad-hoc(hard-coded) expansion tool performed comparably to NN expansion tool (which is not surprising since the NN tool was trained with the ad-hoc heuristic tool). The NN tool has a slightly better recall, but a significantly lower precision. The method of expanding abbreviations using a corpus showed reasonable results whenever the corpus contained the original abbreviated phrases. The NN tool found the right expansion in all experiments we performed, producing however several hundreds of candidate expansions per abbreviation.

Our NN tool could be boosted by:(1)Extending the abbreviation patterns to capture additional information such as syllable boundaries in the abbreviations. (2)An improved training set based on several heuristics(possibly existing methods). We believe that trained on a large enough, more carefully built (containing more negative examples, using several heuristics, with manual extension) training set, the NN approach will considerably outperform the ad-hoc approach.

The main goal of our abbreviation expansion tool was to show the feasibility of the machine learning corpus-based abbreviation expansion approach. At this stage we do not require a great precision, since we later use disambiguation tool [16] that will consider a global interpretation of schema expansion. The main goal of this stage was to reduce the candidate expansions set from several millions (if we consider any word combination of fixed length) to several hundreds (with our methods), giving a reasonable input to the disambiguation tool. We also believe that selecting a context-specific corpus can significantly reduce the number of candidates.

Experiments that involving different network structures, checking corpus sensitivity and comparing our methods to other existing ones is subject to furute work.

References

[1] B.He, K. C.C. Chang. Statistical Schema Matching across Web Query Interfaces; *Proceedings of the 2003 ACM SIGMOD Conference (SIGMOD 2003)*

[2] Erhard Rahm, Philip A. Bernstein; A survey of approaches to automatic schema matching; *The VLDB Journal 10:* 334 − 350 *(2001).*

[3] C. Clifton, E. Housman, A. Rosental. Experience with a combined approach to attribute-matching across heterogeneous databases. *Proceedings of the 7th IFIP 2.6 Working conference on Database Semantics*, October 1997.

[4] A. Doan, P. Domingis, A. Halevy. Reconciling Schemas of Diseparate Data Source: A Machine-Learning approach. *In Proceedings of the ACM SIGMOD Conf*, pp. 509−520, 2001.

[5] Madhavan J., Bernstein P.A., Rahm E. Generic schema matching with Cupid. *In Proceedings of the 27th International Conference On Very Large Data Bases* pp. 49−58, 2001.

[6] Leah S Larkey, Paul Ogilvie, M Andrew Price, Brenden Tamilio. Acrophile: An Automated Acronym Extractor and Server. *Proceedings of the Fifth ACM International Conference on Digital Libraries, 2000.*

[7] Stuart Yeates. Automatic Extraction of Acronyms from Text. *New Zealand Computer Science Research Students' Conference, 1999.*

[8] Stuart Yeates, David Bainbridge, Ian H. Witten. Using Compression to Identify Acronyms in Text. *Data Compression Conference, 2001.*

[9] J.T. Chang, H Schutze, and R.B. Altman. Creating an Online Dictionary of Abbreviations from MEDLINE. 2002. *JAMIA*, to appear. Available at: *citeseer.nj.nec.com/chang02creating.html.*

[10] Youngja Park, Roy J. Byrd. Hybrid Text Mining for Finding Abbreviations and Their Definitions. *2001 Genetic and Evolutionary Computation Conference Late Breaking Papers, pp:* 317 − 324.

[11] Acronym Dictionary, at: *http://www.ucc.ie/cgi-bin/acronym.*

[12] Acronym Finder, at:*http://www.mtnds.com/af/*

[13] Human genome acronym list, at: *http://www.ornl.gov/hgmis/acronym.html.*

[14] The great three-letter abbreviation hunt, at: http://www.atomiser.demon.co.uk/abbrev/.

[15] Opaui guide to lists of acronyms, abbreviations, and initialisms on the world wide web, at: *http://www.opaui.com/acro.html.*

[16] L.Ratinov. Probabilistic semantics for schema matching. Dept. of Computer Science, Ben-Gurion University, Israel, 2004. *MS'c thesis.*

IEEE COMPUTER SOCIETY