

Query Reformulation and Refinement Using NLP-Based Sentence Clustering

Frédéric Roulland, Aaron Kaplan, Stefania Castellani, Claude Roux, Antonietta Grasso, Karin Pettersson, and Jacki O'Neill

Xerox Research Centre Europe, Grenoble, France

Abstract. We have developed an interactive query refinement tool that helps users search a knowledge base for solutions to problems with electronic equipment. The system is targeted towards non-technical users, who are often unable to formulate precise problem descriptions on their own. Two distinct but interrelated functionalities support the refinement of a vague, non-technical initial query into a more precise problem description: a synonymy mechanism that allows the system to match non-technical words in the query with corresponding technical terms in the knowledge base, and a novel refinement mechanism that helps the user build up successively longer and more precise problem descriptions starting from the seed of the initial query. A natural language parser is used both in the application of context-sensitive synonymy rules and the construction of the refinement tree.

1 Introduction

In order to reduce service costs and downtime, users of complex electronic devices are increasingly being encouraged to solve problems for themselves. In order to do this, users need tools that help them identify appropriate solutions. A number of troubleshooting systems are currently available. Some consist of searchable textual descriptions of problems and associated solutions; searchable documents are relatively easy to create and maintain, but in order to search effectively in such a knowledge base the user must be familiar with the content and the terminology used in it. These systems are therefore more suited to expert troubleshooters than to non-technical users. Other approaches such as expert systems and decision trees provide more guidance, asking the user questions rather than simply responding to a query, but such systems are more expensive to build and maintain.

We have developed a text retrieval system called Pocket Engineer (PE) that is tailored to searching in troubleshooting knowledge bases. When a user's initial query returns too many results, PE automatically proposes a list of expressions that contain either some of the words of the query, or synonyms thereof (as listed in a thesaurus). When the user selects one of the expressions, the system proposes a new list of sentence fragments that are possible extensions of the selected expression. This process can be repeated indefinitely, until the number of results is small enough for the user to read through them conveniently.

The refinement choices are constructed on the fly, using the output of a natural language parser that has analyzed the knowledge base in a preprocessing step. Since

the refinement choices are generated automatically from natural language documents, the cost of maintenance is lower than for knowledge-rich approaches such as expert systems or decision trees; yet by iteratively proposing sets of possible refinements, it provides a kind of step-by-step guidance. Particularly when combined with a thesaurus that maps vague, non-technical terms to more precise terms found in the knowledge base, this guidance can help users with little prior knowledge of the terminology or content of the knowledge base to find the information they need.

In [13], we described an ethnographic study of professional troubleshooters mediating between non-technical users and a collection of problem descriptions and associated solutions, and how the findings from that study informed the design of PE. In the current paper, after recalling the type of interaction that PE provides, we present technical details of the various steps of processing that support this interaction, and explain why we believe this mechanism is particularly appropriate, compared to other query refinement mechanisms that have been described in the literature, for helping non-technical users search in a troubleshooting knowledge base.

2 The Query Refinement Process as Seen by the User

Figure 1 shows the PE screen. Initially only the query formulation section at the top of the screen is visible. The user begins a session by typing a query (“lines across page” in the example) in the search box. Subsets of the query words for which there are hits appear underneath the search box; one of these subsets is preselected, but the user can choose a different one (here the user has chosen “lines”).

In the matched problems area, a list of results that match the query is displayed. Each result contains a problem description and possibly a short excerpt from an associated solution description. To the left of this list, in the refinement area, is a navigation tree. Each node of the tree is labeled with an expression from the retrieved documents. Clicking on a node reduces the list of results to those that contain all of the expressions on the path from the root of the tree to the selected node. In the example, clicking on the node “in image area when making copies” causes the matched problem list to be reduced to those results containing both the expressions “white lines” and “in image area when making copies” in the same sentence. Clicking on a node also causes the children of that node to be displayed, to allow further refinement. When the matched problem list has been reduced enough for the user to see a problem description that corresponds to his or her problem, the user can click on the problem description to see a list of solutions.

In its visual aspect, the PE interface resembles those of clustering search engines, which help the user navigate a large set of results by factoring them into groups. Indeed, the term “clustering” is an appropriate description of what the PE refinement tree does to the retrieved documents; but as will be explained in the next section, the hierarchical structure of the PE refinement tree is derived from the structure of individual retrieved sentences, rather than on similarity of vocabulary over entire documents, as is the case in other document clustering methods.

The screenshot displays the Pocket Engineer interface. At the top, there is a search bar with the text "lines across page" and a "New Search" button. To the right, there are dropdown menus for "Select Model:" (set to DC220ST) and "Select your country:" (set to United Kingdom). Below the search bar, there are radio buttons for "across page (1 hits)", "lines (109 hits)" (which is selected), and "page (119 hits)".

Below the search options is a yellow bar with the text "Select/Deselect Suggested Refinements to find your issue in the list of Matched Problems".

The main interface is divided into two columns:

- Suggested Refinements:** A tree view showing a hierarchy of search terms. The root is "All Problems", which is expanded to show "lines", "white lines", "in image area when making copies", "from the document feeder", "from the glass only", "from the document feeder only", "when printing", and "wavy lines on copies prints faces after a new print cartridge has been".
- Matched Problems:** A list of three results:
 - 1 - White Lines in Image Area When Making Copies From the Document Feeder and Document Glass
 - 2 - White Lines in Image Area When Making Copies From the Glass Only
 - 3 - White Lines in Image Area When Making Copies From the Document Feeder Only

The "Matched Problems" column also shows "3 of 109" results.

Fig. 1. Pocket Engineer interface

3 Implementation and Rationale

Having given an overview of the main components of the PE refinement process, we will now present some details of the implementation, and explain the motivations for certain decisions in the context of troubleshooting search. The first subsection presents the underlying objects that we build from the content of the knowledge base, and the second describes how we use these objects to implement the refinement process seen by the user.

The knowledge base accessed by PE consists of problem descriptions, each of which is a single sentence, and solutions, which range from a single sentence to roughly a page of text. Each problem description is associated with one or more solutions, and each solution with one or more problem descriptions. Both problems and solutions are searched for query keywords, but the objects displayed in the “matched problems” area of the interface are problem descriptions; if query words are matched in a solution, then all problems associated with that solution are retrieved, and each is considered to be a separate result. Henceforth, we use the terms “result” and “document” interchangeably to mean a single problem, possibly paired with one of its associated solutions.

3.1 Preprocessing

Indexing: Lemmatization and Vocabulary Expansion Each document in the knowledge base is segmented into sentences, and each sentence into words. Each document is indexed under the lemmata of the words that occur in it, as well as synonyms and related terms, as listed in a thesaurus, in order to improve the recall of initial queries. This expansion is performed at indexing time, rather than at query time, in order to allow synonymy rules that include constraints on a word’s context in the knowledge base. A rule can include lexico-syntactic constraints that a sentence must satisfy in order to be indexed under a given synonym. Each sentence of the knowledge base is parsed, and the constraints are verified against the results of this analysis. For example, a rule can state that the synonym “replace” should be added to the index for any sentence in which the word “change” appears with a direct object whose head is “cartridge,” but not for

sentences where “change” occurs with other direct objects (for example, “change fax settings”).

Segmentation of Sentences into Refinement Expressions As part of the process of indexing a knowledge base for publication via PE, each sentence is segmented into a series of expressions of a granularity appropriate for presentation as refinement choices. This segmentation is performed by rule-based natural language parser [1], using the parser’s general-purpose grammar plus a layer of PE-specific rules. Grammars for English, French, Spanish, Portuguese, Italian, and German are in various stages of development, with the English and French grammars being the most mature and well-tested.

In writing the rules for identifying refinement expressions, we used the following rule of thumb: a refinement expression should be a sequence of words that is as short as possible while containing enough detail and context that users will be able to judge whether or not the expression is relevant to the problems they are experiencing.

The extent of the refinement expressions identified by our rules corresponds in large part to the notion of *chunk* as introduced by Abney [2], though the structure of verbal expressions differs from that typically constructed by shallow parsers. Refinement expressions do not have recursive structure—while the general-purpose grammar identifies nested constituents, the rules for identifying refinement expressions perform a flat segmentation based on that structure. When multiple nested expressions in a sentence fulfill the criteria, the maximal such expression is chosen as a refinement expression. The rules for English can be summarized as follows:

- A simple noun phrase consisting of a head noun and any preceding determiners and modifiers, but not including postmodifiers such as prepositional phrases, is a possible refinement expression. For example,
 - *lines*
 - *white lines*
 - *white and black wavy lines*
 are possible refinement expressions, but
 - *white lines on copies*
 - *white lines when printing*
 would each be divided into two refinement expressions.
- A prepositional phrase consisting of a preposition and a simple noun phrase as defined above is a possible refinement expression. For example,
 - *on paper*
 - *from the document feeder*
- An intransitive verb is a possible refinement expression, as is a transitive verb combined with its direct object (more precisely, the simple noun phrase that heads its direct object). Any auxiliary verbs and adverbs related to a main verb are made part of the verb’s refinement expression. For example,
 - *will not open*
 - *release toner particles*
- A form of the verb “be” used as a main verb, combined with its subject complement, constitutes a possible refinement expression. For example,
 - *is blank*

- In a passive construction, the subject (the logical object) and the verb are combined to form a refinement expression. For example.
 - *when the document feeder is used*

Rules for other languages differ in the details but follow the same general principles.

Normalization of Refinement Expressions In order to make the refinement tree more compact and the refinement choices easier to apprehend, PE attempts to group together refinement expressions that are very similar in meaning, and uses a single node in the refinement tree to represent all the members of such a group. The grouping is performed by mapping each expression to a canonical form; expressions with the same canonical form are grouped together. Two types of transformations are applied: morpho-syntactic and lexical. The morpho-syntactic normalization consists of dropping function words such as determiners and auxiliary verbs, transforming passive constructions to active, and lemmatization, which in English consists of replacing plural nouns with their singular forms and conjugated verbs with their infinitive. For example, the normalization process makes the following transformations:

- *white lines* → *white line*
- *when the document feeder is used* → *when use document feeder*

The lexical normalization uses a thesaurus in which equivalence classes of words and multi-word expressions are listed; each class has a distinguished synonym which is used as the canonical form. The system thus uses two distinct thesauri: one lists groups of equivalent terms for use in normalization of refinement expressions, and the other lists looser relationships used to enrich the index in order to improve the recall of initial queries. (The second thesaurus in fact contains the first—equivalences used in normalization are also used for query expansion, but not the reverse.)

3.2 Query-Time Processing

Subquery Selection If the query contains more than one word, then there may be hits that contain some but not all of the query words. For each retrieved document, we identify the sentences that contain one or more query keywords, and we identify *query matches* within these sentences. A query match is a contiguous sequence of expressions (as defined above) each of which contains at least one word of the query.

Subsets of the query words (henceforth “subqueries”) for which there are query matches are listed below the query box. The subqueries are ranked first by the number of words they contain, and secondarily by the average frequencies of the words. The list is truncated to the *n* best (five in the current prototype) to avoid overwhelming the user with an exponential number of subqueries in the (relatively rare) case of a long query. The top-ranked subquery (which is typically the entire query if there are hits for it) is preselected so that results deemed most likely to be relevant are visible immediately after the initial query, but the user can change this selection.

Typically, when an information retrieval system determines that no documents match the user’s entire query, it either returns zero results (in the case of a boolean retrieval

system), or displays a ranked list of results that match parts of the query (in the case of systems that use a measure of query-document similarity). Our approach is different: PE gives the user an explicit choice of degraded queries. It is standard practice in information retrieval to give less weight to query words that are very frequent in the document collection, and therefore less discriminative. PE does this, by factoring the number of hits into the ranking of subqueries, but unlike systems that present a single ranked list of all of the results, it gives explicit feedback about which keywords it is considering as most important, and allows the user to modify that decision. We chose this mechanism based on two factors. First, before designing PE, we observed users of its predecessor, called OSA. When no matches for the entire query are found, OSA displays a ranked list of results that match various parts of the query. In such cases, many of the displayed results may be irrelevant, and we observed that users were often confused about why these results were being presented. The PE subquery selection is a way of explaining to users how the results they are viewing were chosen, as well as allowing them to override the frequency-based weighting if they see fit. The second reason for the subquery mechanism is related to our particular clustering method, which will be described next. If we were to build a single refinement tree for all documents that match any part of the query, the tree could end up with an inordinately large number of top-level nodes, which would make it difficult to use.

3.3 The Refinement Tree

The root of the refinement tree is labeled “All problems.” This is the node that is selected when the tree is first displayed, and selecting it has the effect of selecting all documents that match the current subquery.

The children of the root are expressions that contain one or more of the words of the subquery, or synonyms thereof. These expressions are extracted from the matching documents. Expression boundaries are determined as previously described in Section 3.1. This level of the tree serves to disambiguate query words that are ambiguous, in two ways.

First, if a query word is ambiguous, and the ambiguity has not already been resolved implicitly by the presence of other query words, then the expressions containing that query word often serve as disambiguation choices. For example, the documents in our knowledge base refer to two different photocopier parts called “document feeder” and “high-capacity feeder.” (the former handles originals, the latter handles blank paper). If the user’s query is simply “feeder,” then the expressions “document feeder” and “high-capacity feeder” will both appear as choices in the first level of the refinement tree. It has often been observed that users of standard search interfaces pick up terms seen in the results of an initial query to formulate a subsequent, more precise query. By presenting multi-word terms that contain one of the user’s query words as refinement choices, the PE interface anticipates subsequent queries that the user might want to make and presents them in a concise format, hopefully reducing the time and effort involved in finding the appropriate term and refining the query.

Secondly, the inclusion of expressions containing synonyms of the query words involves the user explicitly in query expansion. The fact that synonyms are offered as choices rather than added automatically to the query allows us to use a more extensive

thesaurus: while more aggressive synonym expansion increases recall at the expense of overall precision, grouping the results for a given synonym in a single subtree allows users easily to ignore results for synonyms they consider inappropriate. Ours is certainly not the first system to propose synonyms of query words in a query expansion step (see Section 5), but we believe we have arrived at an elegant integration of interactive query expansion into a more general query refinement mechanism.

Nodes in subsequent levels of the tree are also labeled with expressions extracted from the results, but unlike in the first level these expressions need not contain query words or their synonyms. Instead, if node n of the tree is labeled with expression e , then the children of n are labeled with expressions that appear as continuations of e in the documents associated with n . In the example of Figure 1, “white lines” has the children “in image area when making copies” and “when printing” because these are the continuations of “white lines” found in the results.

Recall that clicking on node n selects those results that contain the expressions of n and all its ancestors (except for the root, whose label is not an expression). Children are generated to cover all of these results; that is, each result associated with n is associated with at least one of n 's children. If n 's expression e is the last expression in one of the result sentences, *i.e.* e has no continuation in that sentence, then we choose the rightmost expression in that sentence that is to the left of all expressions already chosen. For example, the query “fax” yields results including “black bands on faxes” and “blurred image on faxes.” These results will both be associated with a first-level node labeled “on faxes,” and each will be associated with a second-level node labeled “black bands on faxes” or “blurred image on faxes.”

Each node represents a unique normalized expression, which may be realized by different surface forms in different sentences (see Section 3.1). Since the normalization process sometimes results in expressions that are ungrammatical or unnatural, we use surface forms as the labels displayed in the refinement tree. When there are multiple surface realizations in the knowledge base of a single normalized expression, we choose one of them arbitrarily to serve as the label.

We hope ultimately to use the syntactic relationships between expressions, rather than simply their linear order, to define which expressions will be chosen as refinements for a given node. In principle this could lead to more appropriate choices for results such as “White lines when printing and when copying.” Whereas the current strategy presents “when printing” as a refinement of “white lines” and then “and when copying” as a refinement of “white lines when printing,” it would be more appropriate to have both “when printing” and “when copying” as refinements of “white lines.” The method we currently use for building a refinement tree from linear sequences of refinement expressions does not generalize in a straightforward way to sentences with a branching structure; making this adaptation, particularly in a way that remains comprehensible in the face of the inevitable errors and ambiguities in parsing, is a subject of ongoing work.

Note that while the procedure described here could be used to generate a refinement tree from the results of a query over an arbitrary document collection, the tree generated for a heterogeneous corpus such as the web would be less useful than that generated for troubleshooting documents. In generating the refinement tree, we exploit the recurrent vocabulary and structure that is typical in troubleshooting documents; if the documents

in the collection do not exhibit this recurrent structure, the tree will likely contain many nodes that represent only one document each.

4 Experimental Results

We evaluated PE in a comparison test against the system it was designed to replace, called OSA. The testing involved both quantitative measurements and ethnographic observation. We summarize the results here.

OSA provides access to the same troubleshooting knowledge base as PE. When a user submits a query to OSA, the query words are lemmatized, synonyms are applied, and documents that contain one or more of the query words are returned in a ranked list. There is a limited refinement mechanism in the form of a few hard-coded multiple-choice questions (*e.g.* “Select when the image quality problem occurs: copy, fax, print”), but we observed that users rarely avail themselves of this mechanism (we hypothesize that this is related both to the layout of the page and to the choice of questions, which often seems inappropriate relative to a given query). In these tests OSA thus serves chiefly as a representative of “classical” information retrieval systems that accept a keyword query and return a ranked list of results.

Each of fourteen users was asked to play out four predefined troubleshooting scenarios, two judged to be simple and two more complicated. The scenarios were selected after discussion with professional troubleshooters about problems their users frequently have. Only problems for which the knowledge base contains a solution were chosen. (We hypothesize that the PE refinement mechanism would also help users determine more quickly that no solution to their problem is available, since it factors a potentially large number of search results into a smaller number of refinement choices that can be scanned quickly to determine whether any are appropriate. The experiment described here did not test this hypothesis, but some anecdotal evidence supports it.) As much as possible the scenarios were presented pictorially (showing what the copies looked like, what the machine and its interface looked like, etc.) to avoid biasing users’ choice of query terms. Each user attempted to solve troubleshooting scenarios with first one system and then the other. The order in which the systems were presented to the user and of which scenario was used with which system was varied, to avoid learning and ordering effects. In addition to recording the number of problems users solved, we administered a usability questionnaire.

PE showed a statistically significant advantage over OSA in solve rate and user preference. 71% of the sessions (20 out of 28) using PE ended with the user finding the right solution, compared to 50% of the sessions (14 out of 28) using OSS¹. In terms of preference, out of the 14 participants, 10 preferred PE, 3 OSA and 1 was indifferent.²

In addition to the quantitative measurements, qualitative observation allowed us to confirm that many users understood the intended purpose of the navigation tree, and to identify areas for improvement.

¹ Two-tailed paired samples t-test: $p=0,047$

² Two-tailed test for difference between proportions: $p = 0,008$; Kolmogorov-Smirnov test for assessing distribution normality: $p = 0,01$.

From these initial tests we conclude that compared to a flat list of results, the PE refinement tree makes it significantly easier for a user to find the document that describes his or her problem. Due to the complexity and expense of performing this type of user testing, the number of scenarios used in the first round of testing was necessarily small; further testing against OSA is planned to confirm the initial results. In addition, comparative testing against a general-purpose clustering information retrieval system is planned, in order to test our hypothesis that the PE refinement mechanism is particularly well-suited to the troubleshooting task.

5 Comparison of PE with Other Troubleshooting and Information Retrieval Tools

Existing approaches to the design of on-line troubleshooting systems can be grouped roughly into two kinds: “knowledge-rich” approaches that make use of domain knowledge *e.g.* in the form of decision trees or models of the machine, and “knowledge-poor” approaches based on information retrieval techniques that do not use extensive domain knowledge. Systems that use knowledge-rich approaches [3, 10, 14] can prompt the user with relevant questions, rather than simply processing queries as the user formulates them, and thus have the potential to be more helpful for naive users, but the knowledge they require can be very expensive to build and maintain over the lifetime of the system. In Pocket Engineer we have implemented a knowledge-poor approach, but one that provides more guidance than a conventional information retrieval interface.

In IR-based troubleshooting systems, *e.g.* Eureka [7], the standard keyword search paradigm is applied to searching in a collection of solution descriptions. This type of interface is very familiar to most users, and can be effective for someone who is already familiar with the contents of the knowledge base and the way important concepts are expressed therein, but it does not meet the needs of a non-technical user who has a problem but is unable to generate spontaneously a sufficiently precise description of it.

The AI-STARS system [4] addressed the mismatch between user vocabulary and the vocabulary of a troubleshooting knowledge base by automatically expanding queries with synonyms. PE also uses synonyms to bridge the vocabulary gap, but with a refinement mechanism that gives the user more control. We believe that this approach will make the system more robust to contextually inappropriate synonyms. As in any system, there is a precision vs. recall balance to be maintained, but in cases where an inappropriate synonym is added to the query, the inappropriate results will be grouped together in a single branch of the refinement tree, and thus more easily ignored in favor of more appropriate results. AI-STARS included a number of tools for automatically identifying potential synonymy relationships by analyzing the text of the knowledge base and logs of user queries; we are currently developing tools along these lines.

Outside of the troubleshooting domain, there has been a great deal of work on mechanisms for helping users view and explore the results of an initial search in ways that are more useful than a simple ranked list. One early approach was relevance feedback [15], in which the user is asked to indicate whether each of the top-ranked results is relevant or not, and new words chosen automatically from the relevant documents are added to the query. Relevance feedback is designed for retrieval scenarios in which

there are many relevant documents, which is typically not the case in troubleshooting. A variation is to ask the user to choose candidate keywords directly [6].

Another class of interaction mechanisms partitions search results into groups and allows the user to narrow down the selection to members of a particular group. Early attempts based on a fixed clustering of the entire document base failed to improve access to relevant documents [8]. The Scatter/Gather technique introduced the possibility of inducing clusters among only the documents retrieved for a given query, with better results [8]. Since the Scatter/Gather paper, a host of alternative algorithms for clustering search results have been proposed, with the goals of improving the speed and/or the quality of the clustering. Whereas Scatter/Gather and many subsequent systems used “polythetic” clustering methods, in which documents are defined to be similar to the degree that their entire vocabularies overlap, a few more recent systems [5, 16, 12, 9] have used “monothetic” clustering methods, in which a cluster is defined by a single term, and all documents in the cluster contain that term. As Sanderson and Croft argue [16], monothetic methods yield clusters in which it is clear what the documents have in common with each other, and finding appropriate names for the clusters is not a problem as it can be in polythetic clustering. In addition, a monothetic method is particularly well-suited to the troubleshooting scenario, in which the user is looking not for a collection of documents about a general topic, but the unique document that describes a particular problem. By using a monothetic clustering, PE generates a refinement tree that, despite being built by knowledge-poor methods, often helps the user construct a coherent problem description, rather than simply a collection of more-or-less related keywords. For example, in Figure 1, from the initial query “lines,” the system proposes the refinement “white lines,” and from there “white lines when printing.”

Among the systems based on monothetic clustering, that of Edgar et al [9] is perhaps the most similar to ours. The root of their hierarchy is the initial query term, and the children of each phrase are longer phrases that contain the parent phrase; for example, *forest* → *sustainable forest* → *sustainable forest management*. Their notion of phrase is more general than ours, so that what PE treats as a sequence of phrases might be considered a single long phrase in [9]. They define phrase boundaries using some simple heuristics based on stop words and punctuation; we hope that the more sophisticated NLP used to define phrase boundaries in PE leads to more linguistically appropriate phrases (this remains to be tested). During the user testing reported in [9], users complained about the fact that Edgar et al.’s system accepted only a single query term, the term to be used as the root of the refinement tree. In PE, we allow the user to specify any number of query terms. Only one can be used as the root of the tree, but the others are taken into account in determining the result set, and for ranking results.

In summary, PE provides a troubleshooting interface that provides searchers with some guidance, yet doesn’t require hand-crafted knowledge resources. PE provides a unique integration of a keyword-based query interface with a refinement tree that groups documents based on both the words and the syntactic structure of individual sentences.

6 Summary and Future work

We have built a system that enables lay users to find solutions to problems they experience with a machine. This system reuses a knowledge base that was initially designed to support expert users. Since lay users are less familiar with the vocabulary used in the knowledge base, and have less understanding of what sorts of information might be important for troubleshooting, they often need help formulating useful descriptions of their problems.

Knowledge-rich approaches to helping users navigate a space of problems and solutions would have required the creation of a new knowledge base with additional content, *e.g.* a decision tree with sequences of questions that can guide a user to the relevant problem description. This additional content would be expensive to create and maintain; by generating refinement choices automatically from the original problem descriptions, our approach allows a similar type of interaction with lower maintenance costs. The main cost of adapting our system to a new document collection would be the cost of adapting the grammar, and we expect this cost to remain relatively small once a general-purpose grammar for the language in question has been written. In our first trials, the grammar developed for a particular knowledge base proved to be sufficient for processing several other knowledge bases that cover the same subject matter. Adapting the grammar to a different domain would probably require somewhat more work; collection-specific grammar adaptations typically involve additions to the lexicon and rules for specific typographical conventions.

Document clustering approaches are easier to put in place and to maintain than knowledge-rich approaches, but the type of clustering most information retrieval systems perform is not well suited to the troubleshooting domain, where there is typically at most one document in the collection that addresses the user's problem, not an entire class of recognizably similar documents. We have thus developed a method based on finding common expressions among individual sentences, rather than on comparing entire document vocabularies, and makes use of syntactic structure identified by a natural language parser. The parser can apply lexical and syntactic transformations in order to discover concepts that are common to multiple sentences even when they are expressed using different surface forms. This method takes advantage of the particular structure of a troubleshooting knowledge base, namely the fact that when a query in such a knowledge base returns many results, the retrieved sentences often have a significant degree of structural and terminological similarity. Our refinement mechanism helps a user iteratively build up a more and more detailed description of a problem, without requiring prior familiarity with the structure and terminology of the knowledge base.

PE includes infrastructure that supports the expansion of queries with context-sensitive synonymy rules intended to help with the mismatch between user vocabulary and the vocabulary of the knowledge base, but the thesaurus needed for this functionality has not yet been systematically populated. We expect this functionality to be particularly effective in combination with the refinement tree, which is structured to make it easy to ignore results based on inappropriate synonyms. Work is currently ongoing to automate parts of the thesaurus development process.

Initial user testing indicated that users have more success at finding solutions with PE than with a classical search mechanism that simply answers a query with a list

of results. Further testing is planned to compare our clustering method with a more traditional clustering based on similarity of document vocabularies, and to evaluate the contribution that individual components of the PE functionality, namely the synonymy mechanism and the novel clustering method, make to the overall performance.

References

1. S. Ait-Mokhtar, J.-P. Chanod and C. Roux: Robustness beyond shallowness: incremental dependency parsing. *NLE Journal*, 2002.
2. S. P. Abney: Parsing by Chunks. In R. C. Berwick, S. P. Abney, and C. Tenny, eds., *Principle-Based Parsing: Computation and Psycholinguistics*, pp. 257–278. Kluwer Academic Publishers, Boston, 1991.
3. D. W. Aha, Tucker Maney, and Leonard A. Breslow: Supporting Dialogue Inferencing in Conversational Case-Based Reasoning. In *Proc. of EWCBR'98*.
4. P. G. Anick: Adapting a full-text information retrieval system to the computer troubleshooting domain. In *Proceedings of the 17th Annual international ACM SIGIR Conference on Research and Development in Information Retrieval*, Dublin, Ireland, July 1994. Springer-Verlag, NY, 349-358.
5. P. G. Anick and S. Tipirneni: The paraphrase search assistant: terminological feedback for iterative information seeking. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, 1999, 153–159.
6. N. J. Belkin, C. Cool, D. Kelly, S. Lin, S. Y. Park, J. Perez-Carballo, and C. Sikora: Iterative exploration, design and evaluation of support for query reformulation in interactive information retrieval. *Information Processing Management*, Vol. 37, Num 3 (2001), 403-434.
7. D. G. Bobrow and J. Whalen: Community knowledge sharing in practice: the Eureka story. In *Journal of the Society for Organizational Learning*, Vol. 4 Issue 2, 2002.
8. D. R. Cutting, D. R. Karger, J. O. Pedersen, and J. W. Tukey: Scatter/Gather: a cluster-based approach to browsing large document collections. *Proceedings of the 15th Annual international ACM SIGIR Conference on Research and Development in information Retrieval*, Copenhagen, Denmark, June 1992. ACM Press, New York, NY, 318-329.
9. K.D. Edgar, D.M. Nichols, G.W. Paynter, K. Thomson, and I.H. Witten: A user evaluation of hierarchical phrase browsing. In *Proc. European Conference on Digital Libraries*, Trondheim, Norway, 2003.
10. F. V. Jensen, C. Skaanning, and U. Kjaerulff: The SACS0 system for Troubleshooting of Printing Systems. In *Proc. of SCAI 2001*, pp. 67-79.
11. B. H. Kang, K. Yoshida, H. Motoda, and P. Compton: Help Desk System with Intelligent Interface. In *Applied Artificial Intelligence*, Vol. 11, Num. 7, 1 Dec. 1997, pp. 611-631(21).
12. D. Lawrie and W. B. Croft and A. Rosenberg: Finding topic words for hierarchical summarization. In *SIGIR '01: Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, 2001, 349–357.
13. J. O'Neill, A. Grasso, S. Castellani, and P. Tolmie: Using real-life troubleshooting interactions to inform self-assistance design. In *Proc. of INTERACT*, Rome, Italy, 12-16 Sep. 2005.
14. B. Peischl and F. Wotowa: Model-based diagnosis or reasoning from first principles. In *IEEE Intelligent Systems* Vol. 18, Num. 3, 2003, pp. 32-37.
15. J. Rocchio: Relevance feedback in information retrieval. In G. Salton (ed.) *The SMART Retrieval System—Experiments in Automatic Document Processing*. Prentice-Hall, 1971.
16. M. Sanderson and B. Croft: Deriving concept hierarchies from text. In *SIGIR '99: Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, 1999, 206–213.