RI08008, 26 May 2008 Computer Science

IBM Research Report

Interleaved Retrieval of Documents and Diagnostic Trees for Self Help Portals

Dinesh Garg, Nanda Kambhatla
IBM Research Division
IBM India Research Lab
EGL, Block D, 3rd Floor,
Bangalore - 560071, India
dingarg2@in.ibm.com, kambhatla@in.ibm.com

Gopal Pingali
IBM Research Division
IBM TJ Watson Research Center,
NY, USA
gpingali@us.ibm.com

IBM Research Division
Almaden - Austin - Beijing - Delhi - Haifa - T.J. Watson - Tokyo - Zurich

Interleaved Retrieval of Documents and Diagnostic Trees for Self Help Portals

Dinesh Garg
IBM India Research Lab
Bangalore, India
dingarg2@in.ibm.com

Nandakishore Kambhatla
IBM India Research Lab
Bangalore, India
kambhatla@in.ibm.com

Gopal Pingali
IBM T J Watson Research
Center, NY, USA
gpingali@us.ibm.com

ABSTRACT

Self help portals are online portals for product sales and customer support. Today, consumers can resolve most of their problems pertaining to a product by accessing information from these portals. The information on self help portals is often organized as solution documents and *diagnostic trees*. A diagnostic tree encodes the diagnostic steps for iteratively narrowing down the scope of user's problem and then eventually presenting the most relevant set of solution documents to the user. Typically, search is enabled only for solution documents. In this paper, we present algorithms for unified, interleaved retrieval of solution documents and diagnostic trees that can lead to more efficient resolution of user problems, especially when users' queries are imprecise. We show that interleaved retrieval of documents and diagnostic trees leads to a improvement up to 10% in precision for general help desk kind of queries about Microsoft Excel XP.

Categories and Subject Descriptors

H.5.2 [**Information Interfaces and Presentation**]: User Interfaces

General Terms

Documentation, Human Factors

Keywords

Diagnostic Trees, Solution Documents, Customer Support Portals, Search

1. INTRODUCTION

Self help portals are online portals for product sales, support, and customer service used by end users to access information and resolve problems. The end user requests could range from simple needs like downloading a user manual for a washing machine to complex issues like finding optimal plans for wealth management. Self help portals have become an integral part of 24×7 online customer support and Customer Relationship Management strategies of most companies.

A self help portal contains varied features like frequently asked questions (FAQ), search, etc. The goal is to enable users to resolve their queries quickly and efficiently with minimal human (agent) involvement that can lead to greater user satisfaction and lower costs for the company. For instance, a typical search feature lets users search for solutions relevant to their query from a corpus of solution documents, resulting in a ranked list of such documents displayed back to them.

A solution document describes a solution to a particular problem. Typical problems are 'how-to' questions about troubleshooting or accessing specific features and products; e.g. 'How do I add an attachment to an email?' or 'How do I reset my Internet password?'. Solution documents are drafted by the designers, experts, and support staff for the products and/or services. These documents are indexed and made available for the search by users.

In addition to solution documents, *diagnostic trees* are a mechanism to capture the diagnostic steps often followed in narrowing down users' problems and suggesting solutions to them. These trees contain user problems at their root nodes, solution documents at their leaf nodes and successive diagnostic or resolution steps at their intermediate nodes. Thus, each diagnostic tree captures knowledge about resolving a particular problem and finding relevant solution documents. A user can navigate the tree to find specific solution documents relevant to their issues.

Self help portals often contain both diagnostic trees and solution documents in order to enable users to find solutions quickly. However, typically, search is enabled only over solution documents. This can lead to an inefficient and frustrating experience for users in their quest for answers, especially when their queries are imprecise. Enabling unified retrieval of both diagnostic trees and solution documents can help alleviate this problem.

In this paper, we present algorithms for interleaved retrieval of diagnostic trees and solution documents for self help portals. We compute relevance of diagnostic trees for queries utilizing the relevance scores of indexed solution documents. We show that our algorithm satisfies a set of axioms. For relevence feedback data compiled for help desk queries about Microsoft Excel XP, we show that interleaved retrieval improves precision up to 10% over separate retrieval.

The rest of this paper is organized as follows. In the next

.

section, we explain our notion of diagnostic trees. Next, we discuss the need for interleaved retrieval of diagnostic trees and solution documents. We then delineate axioms for interleaved ranking and present a ranking algorithm that satisfying these axioms. Finally, we describe experimental results demonstrating the utility of interleaved retrieval and present our conclusions.

2. RELEVANT WORK

This paper is related to the area of *structured queries for the hierarchical structure* which is well described in [2]. This area addresses the problem of exploiting the hierarchical structure of text documents while querying them. In our case, diagnostic trees codify the hierarchical structure in documents. However, as we will describe later in the paper, unlike the situation for structured queries, for self help portals, the user queries typically describe user problems and not their causes. Thus, structured query search may not help much here.

There is a large amount of literature on *diagnostic trees*. Most of this research has focused on automatic creation and optimization of diagnostic trees [4] [3]. We were unable to trace any prior work about unified retrieval of trees and documents.

There is also literature available on axiomatic foundations of ranking system [1] but this literature is mostly concerned with the aggregating the preferences of the rational agents.

3. DIAGNOSTIC TREES

Diagnostic trees (DTs) are data structures that capture the iterative/hierarchical diagnosis of a problem or set of problems. DTs can be trees of many levels, where the leaf nodes are solution documents. An internal node of a DT comprise of the description of a problem whose solution can be found by navigating one of the descendants of the node. A user can navigate a DT to find solutions to a particular problem by starting from the root of the tree and ending at a solution document. When a user's initial query is imprecise, DTs can help the user navigate through the solution documents. Typically, as shown in Figure 1, DTs are created by people with expertise about the problem areas and they use the DTs to encode the knowledge in their heads. Figure 2

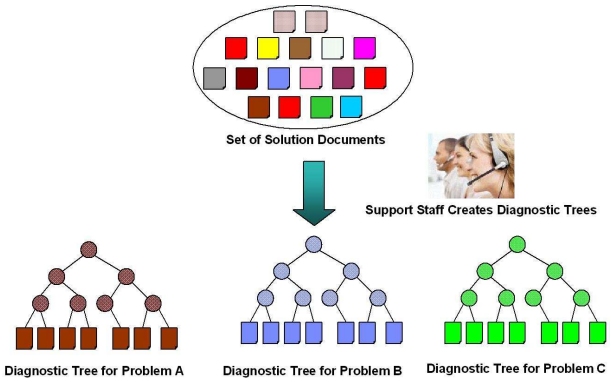

Figure 1: Creation of Diagnostic Trees

depicts a sample DT. This DT codifies the diagnosis for a user problem "**Unable to send email while using Lotus Notes 6.0**". Note that

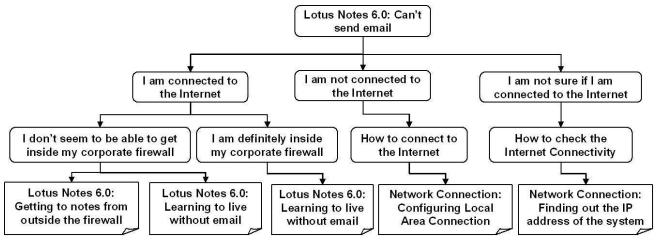

Figure 2: A Sample Diagnostic Tree

- The solution documents are always the leaf nodes of the DT.
- Internal nodes have text descriptions of (narrower) problems that help users navigate through the tree to narrow down their problem.
- A user can traverse the tree starting from the root to repeatedly narrow down the problem to reach a solution document most relevant to the problem.

4. INTERLEAVED RETRIEVAL OF DTS AND SOLUTION DOCUMENTS

In this section, we discuss the need for interleaved retrieval of DTs and solution documents in response to user search queries. Interleaved retrieval implies that when users enter search queries, they should get back relevant solution documents as well as DTs[1] in unified ranked lists. An example of such a unified interleaved list with both solution documents and DTs is shown in Figure 3. Here, the search query

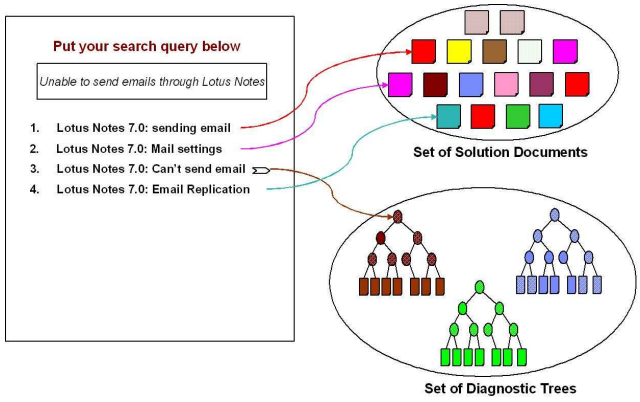

Figure 3: Illustration of the result of interleaved retrieval of DTs and solution documents

is "*Unable to send emails through Lotus Notes*", and the results consist of three solution documents (ranked 1, 2, and 4) and one DT (ranked 3). When displaying a DT, we always display the text associated with the root node of the tree. When the user clicks on the DT entry, it unfolds in a breadth first manner. Figure 4 illustrates a user's navigation through the DT in Figure 2.

Why do we need interleaved retrieval of DTs and solution documents? Will the retrieval of solution documents not

[1]The notion of *relevance* of a diagnostic tree is defined later in the paper.

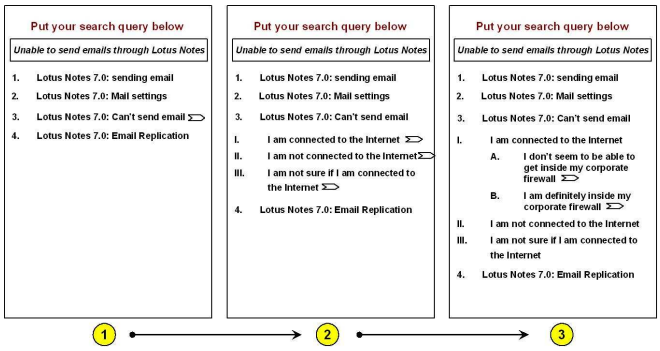

Figure 4: Illustration of navigation through a DT from an interleaved set of DTs and solution documents

suffice, especially since they are the leaf nodes of DTs and navigation through DTs culminate in solution documents anyway?

The answers to these questions are rooted in the fact that in customer support centers, users of products and/or services often only provide high level descriptions of problems they are facing while using the products or services. Typically, a user does not know or even want to know the cause behind a problem. She just wants to fix the problem without getting into the potential causes of the problem (for which there may be a great many possibilities).

Given the above pattern, search queries are often imprecise due to a lack of knowledge by users about the problem at hand. This, in turn, results in the relevant solution documents not being retrieved for such queries. In most such cases, search queries only describe problems at a high level without describing auxiliary information relevant to problem resolution. Such high level queries are often ineffective in retrieving solution documents describing resolution of the problems being experienced by the users.

DTs can effectively guide users from abstract high level problem descriptions to relevant solution documents by repeatedly prompting for and obtaining relevant auxiliary information necessary for problem resolution. Especially for imprecise, high level queries, DTs will often have higher relevance scores than solution documents, since intermediate nodes of DTs necessarily have abstract, high level information about problems and their solutions. For such queries, it is often the cause of the problem and *not the problem itself* that helps retrieve the solution documents most useful for the user. The logic behind this claim is simple: for a given problem there may be several potential causes and for each cause there may be one or many solution documents. When information about the root cause is absent in queries, search algorithms are unable to retrieve the right set of solution documents, since enough key words may not match the queries and documents.

For example, suppose a user is unable to send email while using Lotus Notes 6.0. She might submit search queries like "*Lotus Notes unable to send email*" or "*unable to send email while using Lotus Notes 6.0*". Since several different reasons might have led to the user facing this problem, such queries are unlikely to match the solution documents without more (auxiliary) information from the user. For this query, a higher level DT (e.g. the one in Figure 2) might match well with the queries and also let the user find relevant solution documents by providing more information while navigating the DT.

In general, retrieving both DTs and solution documents for user queries might work better than separately retrieving both. Having a unified ranked list ensures that the user can find the most relevant information at rank 1, which can be a DT if the query is more abstract or high level. Thus, users can express queries at the level of abstraction or problem knowledge suitable to them and expect to find relevant information (in the form of DTs or solution documents) at the top of the result set.

5. ALGORITHMS FOR INTERLEAVED RETRIEVAL

Interleaved retrieval necessitates a combined ranking of diagnostic trees and solution documents together in a single ranked list. In this section, we present algorithms for ranking trees and documents together.

Since a diagnostic tree is not a document, but a logical collection of (solution) documents addressing the same problem, we do not index them directly as we would index documents. We derive the ranking of diagnostic trees using the ranks of the solution documents contained by the trees.[2] Our ranking scheme is based on simple concept that if we know how to derive rank of a *one level* tree by using ranks of its leaves then we can derive the rank of any complex tree by using the ranks of its leaves. Therefore, we treat one level tree as building block for our purpose and develop the axioms for ranking using this. For this, we first define the concept of *canonical form* of a one level tree as follows. Suppose we have a one level tree with n leaves and a score vector of $(r_1, r_2, \ldots, r_n)$ for these leaves. The canonical form of this tree is another one level tree which also has the same set of n leaves but the score of each leaf is equal to the average score of all the leaves in the given tree (that is, $\frac{1}{n}\sum_{i=1}^{n} r_i$. Figure 5 illustrates this idea. By using this

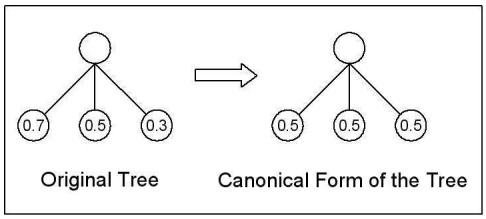

Figure 5: Canonical form of a tree

definition of canonical form, we put forward 5 (heuristic) axioms that need to be satisfied by any ranking scheme for computing relevance of a diagnostic tree given the relevance scores of solution documents contained by the tree. The 5 axioms are as follows:

- **Boundary Condition**: For a diagnostic tree T with just two nodes - i.e. a root node and a leaf node containing a solution document, the score of the tree T must be equal to the score of the solution document r.

[2]In this context, by ranking of a solution document, we mean the relevance ranking score of a solution document assigned by the usual search algorithm that underlies the search application.

- **Score Monotonicity**: Consider the case of two diagnostic trees A and B with the same number of leaves but with different sets of solution documents belonging to these leaves. For a given search query q, we compute the relevance score of all solution documents and convert the trees into their respective canonical forms as shown in Figure 6. In this scenario, tree A must receive a higher relevance score than tree B. In other words, the relevance score of a diagnostic tree monotonically increases with the scores of leaves in its canonical form.

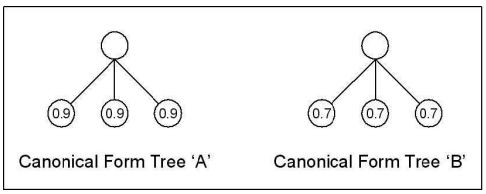

Figure 6: Illustration of *Score Monotonicity* axiom

- **Size Monotonicity** Consider the case of two diagnostic trees A and B with different number of leaves and different sets of solution documents belonging to these leaves. For a given search query q, we compute the relevance scores of all the solution documents and convert the trees into their respective canonical forms as shown in Figure 7. In this scenario, the tree A must receive a higher relevance score than tree B. In other words, the relevance score of a diagnostic tree monotonically increases with number of leaves in its canonical form.

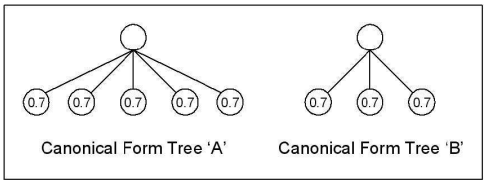

Figure 7: Illustration of *Size Monotonicity* axiom

- **Distribution of Relevance Scores** Consider the case of two diagnostic trees A and B with the same number of leaves but with different sets of solution documents belonging to these leaves. For a search query q, suppose the relevance scores of solution documents in tree A are more uniformly distributed than the relevance scores of solution documents in tree B, then the tree A must receive a higher score than tree B. This idea is explained in Figure 8 where we have shown trees A and B along with their canonical forms (which is the same for both A and B). The assumption is that the information contained in A will be more useful in resolving user problems, since there is a greater uncertainty (similar scores) among all the solution documents at the leaf nodes for A and letting users iteratively choose from among these might be effective.

- **Identical Scales** This axiom says that the scale on which the relevance ranking of the diagnostic trees are being computed must be same as the scale on which the relevance ranking score of the solution documents is being computed by the usual search algorithm.

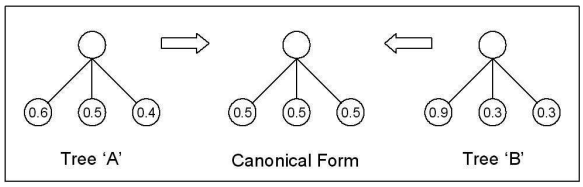

Figure 8: Illustration of *Distribution of Relevance Scores* axiom

5.1 A Ranking Scheme

In this section, we describe an instance of a ranking scheme which will rank the solution documents corresponding to a search query along with the diagnostic items (the root nodes of the tree shown in Figure 3) in a single ranked list while satisfying all the axioms discussed earlier. The high level idea behind this scheme is as follows. The ranking scheme iteratively starts from leaves of a diagnostic tree and proceeds upwards towards the root. In each iteration, it computes the relevance scores of only those internal nodes which themselves are not leaves and whose children are all leaves. In the next iteration, all such internal nodes become the leaves of the tree and the process repeats. Thus, in each iteration, it computes the relevance score of only those nodes which are roots of one level trees. Also, while computing the the relevance scores of one level trees, the algorithm ensures that all the 5 axioms are satisfied. In what follows, we discuss this scheme in more detail.

Let us assume that we have already run some standard search algorithm and retrieved a ranked list of the solution documents in response to the search query. Let us assume that in this list, the solution documents are arranged in decreasing order of the relevance score (which is number in between 0 and 1). Now for each diagnostic tree assign a score to each of its leaf node. Recall that the leaf node of a diagnostic tree is a solution document. Therefore, the score of the leaf will be the same as the relevance score of this solution document if it is present in the ranked list retrieved earlier otherwise it will be zero. See Figure 3 for an example. Use the relevance scores of all the leaves to compute a relevance score of the root node which will be the relevance score of the diagnostic tree. Now use this score of the diagnostic tree to rank it along with the other solution documents. Thus, the crux of whole algorithm lies in designing the scheme for computing the relevance score of the root node given the relevance scores of all the leaf nodes.

The **Diagnostic-Tree-Relevance** algorithm shown as Algorithm 5.1 below is one possible algorithm for computing the relevance score of a diagnostic tree root from the give relevance scores of its leaf nodes. In this algorithm following are the input quantities.

1. The diagnostic tree T
2. The relevance scores of all the leaf nodes, say - $r_1, \ldots, r_k$, where $0 \leq r_i \leq 1 \forall i = 1, \ldots, k$
3. Weight Factor - β

This algorithm scans the given tree in a bottom up fashion starting from leaves and moving towards the root. At each

Algorithm 1 Diagnostic-Tree-Relevance

Procedure Diagnostic-Tree-Relevance $(T, r_1, \ldots, r_k, \beta)$

1: $n \leftarrow$ total number of the nodes in the tree T
2: **if** $n > 1$ **then**
3: $l \leftarrow$ a leaf node of tree T
4: $S \leftarrow$ the set of all the sibling nodes of l
5: **if** (all the nodes in set S are leaf nodes) **then**
6: $p \leftarrow$ parent node of node l
7: $r_l \leftarrow$ relevance score of node l
8: $R_S \leftarrow$ relevance scores vector of all nodes in S
9: $r_p \leftarrow$ **One-Level-Tree-relevance** (r_l, R_S, β)
10: delete all the nodes in S and node l from tree T
11: make node p as leaf node
12: $n \leftarrow (n - |S| - 1)$
13: **else**
14: $l \leftarrow$ a node in S which is not a leaf node
15: Go to line number 4
16: **end if**
17: **else**
18: Return the relevance score of the remaining root node
19: Stop
20: **end if**

iteration, the algorithm identifies a subtree within the given tree which is just one level tree. The algorithm computes the relevance score of this one level subtree by making use of another subroutine **One-Level-Tree-Relevance(.)** given in Algorithm 5.1. It is easy to verify that the expression

Algorithm 2 One-Level-Tree-Relevance

Procedure One-Level-Tree-Relevance $(r_1, \ldots, r_m, \beta)$

1: **if** $m = 1$ **then**
2: return r_1
3: **else**
4: $R \leftarrow \sum_{i=1}^{m} r_i$
5: Compute the entropy E of the tree as follows

$$E \leftarrow \left(-\sum_{i=1}^{m} \frac{r_i}{R} \log \frac{r_i}{R} \right)$$

6: Normalize the entropy as follows: $E \leftarrow E / \log m$
7: Compute the average score A of a node as follows $A \leftarrow R/m$
8: Compute the relevance score ϕ of the tree as follows $\phi = A + (1 - A)(\beta E + (1 - \beta)(1 - \frac{1}{2^m}))$
9: return ϕ
10: **end if**

used for computing the relevance score ϕ in Algorithm 5.1 satisfies all the axioms discussed earlier. For example,

1. The **if** statement takes care of the *boundary condition* axiom.
2. The first term of the expression for ϕ (i.e A) satisfies the requirements of the *score monotonicity* axiom.
3. The quantity $(1 - \frac{1}{2^m})$ satisfies the requirement of the *size monotonicity* axiom.
4. The quantity E satisfies the requirements of the *distribution of relevance scores* axiom.
5. The way we have developed the formula for ϕ satisfies the requirements of the *identical scale* axiom.

It is also easy to verify that the time complexity of the Algorithm 5.1 is $O(n)$.

6. EXPERIMENTS

In this section, we present results comparing interleaved retrieval of diagnostic trees and solution documents with retrieval of only solution documents.

For our experiments, we used a commercially available knowledge content database as our corpus. This database consists of 56,537 solution documents related to variety of software (e.g. Microsoft Excel, Microsoft Office, etc.) as well as hardware related problems that can be faced by typical IT users. Each document consists of the title of the problem and a step-by-step procedure to fix that problem. In our experimental setup, we first indexed all these 56,537 documents by making use of *Apache Lucene* APIs. [3]. Note that this indexing is a one time operation.

We manually created 34 diagnostic trees using 1,333 documents out of 1,762 solution documents in the corpus that address problems related to using Microsoft Excel XP software. Each tree was designed to address a specific kind of problem. For simplicity, we just created one level trees where the root of the tree had a text description of an abstract problem and the leaf nodes were the solution documents from the corpus which addressed some aspect of that problem. A few sample text descriptions of the roots of the trees are given below.

- Excel XP - How do I ? - Adding/Removing
- Excel XP- How do I ? - Viewing/Displaying

Total number of leaves across all the trees were 1,366. On an average, each diagnostic tree contained 40 solution documents as leaf nodes.

Finally, we developed an interface for searching the solution documents in the corpus. In response to user queries, the following steps get executed in the background - (1) Search is performed using Lucene APIs and the relevant solution documents are retrieved in the form of a list. In this list, each solution document consists of the title and a URL of the document. (2) The program iterates over the retrieved list and for each solution document, it finds all the trees (which we have annotated in the previous step) that have this document as one of the leaf. The program assigns the relevance score of this document (as given by Lucene) to each of these leaf nodes. (3) For each tree, if there is a leaf which does not appear in the retrieved list then its score is assigned to as 0. (4) The relevance score of each tree is computed using Algorithm 5.1 with $\beta = 0.9$.

Once the background calculation is over, we present the search results to users as two ranked lists placed next to each other. In left hand side list, we just present the solution documents returned by Lucene in the decreasing order of their relevance scores. In the right hand side list, we present the list of interleaved diagnostic trees and solution documents where we rank the diagnostic trees along with solution documents based on their relevance scores computed earlier. We asked users to select the documents (and trees) relevant to

[3]Apache Lucene is an open source text search engine library. URL - http://lucene.apache.org/java/docs/

their queries in each of these two lists. When users submit their feedback, we log the feedback in a database. Note that in the case of right hand side list, if an entry is a tree then we don't provide the option for the user to specify which leaf nodes are relevant and which are irrelevant. Instead, we just provide the option for the user to specify whether the tree as a whole is relevant or not.

We collected user relevance feedback from 7 different users as described above. These users were instructed to search for problems related to Microsoft Excel XP. For each search query, we presented 10 results in both left hand and right hand lists. Each entry in the result set had a mention of the title and a link to the document. For trees, each entry has a textual description of its root that unfolds into a view showing the leaf nodes when a user clicks on it. We asked users to give feedback about the relevance of each document/tree in both the result lists.

The objective of this experiment was to determine the extent of improvement, if any, that can be obtained by retrieving diagnostic trees along with solution documents. We computed the following measures for both interleaved retrieval of trees and documents and standard retrieval of only solution documents:

- Mean Precision (MP) at position i where $i = 1, 2, \ldots, 10$
- Mean Average Precision (MAP)
- Mean Reciprocal Rank (MRR)

The mean precision at position i is the same as the mean of (single query precisions at position i) across all the queries. MAP is the mean of (single query average precisions) across all the queries. MRR is the mean of (1/rank of first relevant document for the single query) across all the queries.

We collected user relevance feedback for 65 queries from a set of 7 users on the topic of Microsoft excel XP as described above. For this data set, we computed the above three performance metrics for both the cases separately - when we allow interleaving of the diagnostic trees and when we do not allow interleaving. The results are summarized in Tables 1 and 2.

Position	MP without Trees	MP with Trees	Gain
1	0.461538	0.492307	+06.66%
2	0.376923	0.415384	+10.20%
3	0.343589	0.364102	+05.97%
4	0.315384	0.326923	+03.65%
5	0.283076	0.292307	+03.26%
6	0.264102	0.271794	+02.91%
7	0.243956	0.252747	+08.25%
8	0.236538	0.253846	+07.31%
9	0.222222	0.237606	+06.92%
10	0.210769	0.232307	+10.21%

Table 1: Mean Precision (MP) at each position

MAP without Trees	MAP with Trees	Gain
0.314041	0.315341	+0.41%
MRR without Trees	MRR with Trees	Gain
0.562222	0.578504	+2.89%

Table 2: Mean Average Precision (MAP) and Mean Reciprocal Rank (MRR)

7. CONCLUSIONS AND FUTURE DIRECTIONS

In this paper, we propose algorithms for uniform, interleaved retrieval of solution documents and diagnostic trees for self help portals. Diagnostic trees are often used to codify the diagnostic steps in narrowing down an imprecise query to a specific solution document. Uniform interleaved retrieval can be especially beneficial when user queries are imprecise or vague.

We described an axiom based approach to compute the relevance scores of diagnostic trees by using the scores of solution documents. Our initial experiments with a corpus of documents on Microsoft Excel XP show that interleaved retrieval of solution documents and diagnostic trees improves the precision up to 10% over retrieval of only solution documents.

8. REFERENCES

[1] A. Altman and M. Tennenholtz. On the axiomatic foundations of ranking systems. In *19th International Joint Conference on Artificial Intelligence - IJCAI05*, pages 917–922, 2005.

[2] R. A. Baeza-Yates and B. A. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press / Addison-Wesley, 1999.

[3] W. J. Hopp, S. M. Iravani, and B. Shou. Performance improvement diagnostic tree for serial manufacturing environments. In *2005 NSF DMII Grantees Conference*, Scottsdale, Arizona, 2005.

[4] D. Tong, C. Jolly, and K. Zalondek. Diagnostic tree design with model-based reasoning. In *IEEE Automatic Testing Conference - AUTOTESTCON'89*, pages 161–167, Philadelphia, PA, USA, 1989.