Content lens **520** first eliminates stop words, such as "a", "an", "the", "to", etc., and then tries to partition the documents by common sentences, phrases or words. As an example, content lens **520** finds 40 documents that contain the term "Home allendale bayonne belleville bergenfield bloomfield butler" corresponding to a cluster in cell **522**. In cell **524**, content lens **520** finds 20 documents that contain the term "top business and economy companies restaurants organizations". In cell **526**, content lens **520** finds 20 documents that contain the term "cape may county". Contact lens **520** finds 20 documents that have no patterns in cell **528**. Edges **554** indicate that documents clustered in cell **512** are exactly the same documents corresponding to cell **522**. Edges **556** indicate that the documents found in cell **514** are exactly the same as the documents clustered in cell **524**.

FIG. **5** also shows Age legs **530**. Age lens **530** clusters on the documents' date of last update. Age lens **530** partitions the 500 documents found into 4 clusters corresponding to cells **532**, **534**, **536**, and **538**. Cell **532** shows 40 documents which were updated on Mar. 4, 1997. Cell **536** finds 30 documents updated on 1997. Cell **536** finds 20 documents updated on 1996. Cell **538** finds 10 documents updated on 1995.

FIG. **5** shows the window of clusters results from which the user may select any number of cells from Lenses **500**, **510**, **520** and **530**. The system, then, displays the documents that satisfy the conditions in all the selected cells. For example, as shown in FIG. **6** which is a graphical display of a selected search result of FIG. **5**. In FIG. **5**, the user chooses cell **526** corresponding to "Cape May County", cell **534** corresponding to the year 1997 and cell **536** corresponding to year 1996. As a result, the system displays 15 documents that contain "Cape May County" and were last updated in 1996 or 1997. The documents found are clustered and displayed in cell **600**. Cell **602** indicates that 10 of the documents found were updated in 1997. Cell **604** indicates that 5 out of the 15 documents found were last updated in 1995.

In another embodiment of the invention, the set of keywords from the search query are used to rank the documents returned by the search engine. The more keywords that appear in a document, the higher the document is ranked. The results organizer outputs results from the highest ranked to the lowest ranked.

The benefit of clusters to the user is that the clusters may contain all items of interest or duplicate items. In the first case, only items in the cluster need be reviewed by the user. While in the second case, only one item from the cluster needs to be reviewed by the user prior to rejecting all the other documents in the cluster. The "Vote" column in the display allows the user to indicate the relevancy of a cluster to his informational needs. If the user votes positively on a cluster, the system can use the documents in the desired cluster (referred to hereafter as good documents) to recluster the remaining documents, giving a higher weight to documents that are similar to the good documents. If the user votes negatively on a cluster, the system can use the documents in the undesired cluster (referred to hereafter as bad documents) to recluster the remaining documents, giving a lower weight to documents that are similar to the bad documents.

All clusters which receive a yes vote are saved along with the query in a search context folder. A user has the ability to find a query and its results by either browsing the search context folders or doing a keyword based search for them among all the search context folders.

Often times, it is desirable to filter the output of search engines, to prevent information from being displayed. If the search engine returns a large set of URLs, one may want to restrict it to pages that were visited only last week, or to pages that have been bookmarked, or to a smaller set of URLs that are relevant, based upon some specific criteria not previously captured in the search engine index. Sometimes it is desirable to filter the output to exclude pages to which a user should not have access. Referring to FIG. **2**B, there is shown the details of the dynamic filter processor in step **24** of FIG. **1**A. The dynamic filtering, based on a dynamic set of URLs, is used to restrict the results of a search query. In the exemplary embodiment of the invention, the dynamic set of URLs can be determined explicitly by the user from a user profile as shown in step **57** of FIG. **2**B, or, in general, from information stored in other information management systems. Once the profile is accessed, then in step **59** the URL's may be filtered and the results are displayed in step **61**.

FIG. **7** is a Venn diagram which illustrates a dynamic filtering operation. In FIG. **7**, area **710** defines **S1** as the set of URLs of pages and their summaries, returned by the search engine, in response to a query for text in its own information management system. Let n be the number of URLs in **S1**. There are also n summaries corresponding to each URL of **S1**. Area **712** defines **S2** as the set of URLs that is dynamically generated by the user or by a query external to the search engine. Let m be the number of URLs in **S2**. There are no summaries associated with these URLs. **S1** and **S2** are likely to have URLs in common. The filter returns the URLs in the intersection of **S1** and **S2** and the respective summaries corresponding to area **714**.

The dynamic filtering of the present invention may improve upon the poor performance of other filtering techniques which typically involve multiple disk fetch and store operations and several sorting steps. The performance of a fetch and store filtering technique of this type would be O((m+n)log(m+n)). In contrast, the dynamic filtering of the present invention uses hashing and associated arrays in an intelligent fashion and has a performance of O(m+n).

First, for set **S2**, an associative array is setup as shown in Table 1.

Table 1

```
Flag[url_1]=1;
Flag[url_2]=1; . . .
Flag[url_m]=1;
```

In table 1, "url_n" represents a hash address generated from a particular URL. This process takes m steps. The Flag array indicates the URLs to be included from **S1**.

As the search engine starts returning URLs of set **S1**, a hash index into the associative array is generated from the URL and a check is performed to see if the corresponding Flag is set. Only the URLs with the corresponding Flag set and their summaries are provided as the output URL's of the dynamic filter. Everything else is ignored. After the URL is provided as an output URL, its Flag is reset to 0, to ensure that the same URL is not presented again. This is performed sequentially and hence takes n steps. The entire algorithm is therefore O(m+n) since lookup using a hash table in an associative array is O(1). The result of this process is the intersection of **S1** and **S2**, with the associated summaries from **S1**. Duplicate URLs, if returned by the search engine are eliminated.

FIG. **8** shows the architecture of an exemplary implementation according to the third embodiment of the present invention. The implementation uses commercially available tools such as a Harvest search engine, an Informix database and a dynamic filter implemented using JavaScript and Perl.

As an example, a particular implementation is the one of a student wanting to search pages of an Internet course that he is taking, but limiting it to the pages he visited last week.

The Student is presented with a query page **810**. Query page **810** is a form where the student user inserts the text to search and specifies to limit the search only to pages he visited last week. When the form is submitted, the query is split in two other queries **812** and **814**. Query **814** goes to the Informix database **816** which has tracked the student's navigation in the course. The Informix database **816** generates a dynamic set **822** of URLs (S2) which represents the pages he visited last week. Dynamic set **822** of URLs S2 is then passed to dynamic filter **824**, which sets up an associative array of flags as explained above, with reference to FIG. **7**.

Query **812** goes to the Harvest search engine **816** which is configured to index all the pages of the course. Upon receiving query **812**, the Harvest search engine **816** starts to return pages from its index. The pages returned are all of the pages for the course no matter when the student visited the course pages, since student access information is not stored in the search index. The Harvest search engine **816** also returns pages that the student has not yet visited. The output set **818** from Harvest S1, is processed against the Flag array by dynamic filter **824** and the intersection is returned. The algorithm also filters out duplicates.

Dynamic filtering according to the present invention, can be implemented anywhere one desires to restrict/filter one set of URLs by another set of URLs. This may be desirable, for example, for security reasons, when a company wants to restrict access to its resources based on the employee's identity. Different employees may have associated with different lists of URLs that they have permission to read. When a search is performed, the company can insure that only those URLs that are not restricted for the particular user are presented. This addresses a common problem that search engines have: returning summaries of pages to which a user does not have access. Usually, search engines return summaries of protected pages, even if the user is restricted access to those pages via web server password protection or other mechanisms. Adult related material can also be filtered the same way. Internet service providers could use dynamic filtering to prevent children from searching adult web sites and newsgroups. Personalizing search engines in an educational environment can be taken to higher levels. A student can associate various topics with the visited pages and then, search the database based on URL's that are associated with a particular topic. A teacher may also mark certain pages as being essential for a final exam, or as potential topics for independent study. Then searching could be restricted to this set of pages only.

For each component of the user's query, there are a few natural ways in which the query may be either restricted further or relaxed further to potentially produce either fewer results or a greater number of results. This concept is best illustrated by an example. Suppose the user's query specifies that the phrase "cryptographic protocols" appear as a heading in the document. A more restrictive, or stricter, query would require that "cryptographic protocols" be part of the title, which would typically yield fewer results in comparison to the user's query. A less restrictive, broader query would specify that "cryptographic protocols" appear anywhere in the text, which would typically yield a greater number of results in comparison to the user's query. Another way to relax the user's query is to require that the words "cryptographic" and "protocols" appear near each other in a heading, rather than as a phrase. Relaxation and restriction of the query is precisely the process that a user currently performs, with no help from the search engine, in order to refine the user's search. A query tuner according to the present invention offers the user a helpful guide in the user's quest by taking the user's query and generating a small number of additional queries according to a query hierarchy. The user's query and the additional queries are forwarded to the search engine. The query tuner, then, evaluates the results of all the queries and suggests possible query reformulations to the user, together with the expected number of matching documents each reformulation would yield.

Since each search engine has its own query language, the query tuner, as shown in FIG. **2A**, is defined for an abstract query language that can easily be mapped to any particular engine's language. In most search engines, there are two types of data that the user can input: content that is to be matched to the text of documents, and structure, or meta-data, related to each document that represents conditions to be satisfied in order for a document to be considered a match. For example, in the AltaVista query shown in Table 2.

Table 2

title: "cryptographic protocols"
English language
dated after Jan. 1, 1997

For this example, content is "cryptographic protocols" and meta-data is all the other information, consisting of the requirements that the content appear in the title of the document, that the document's language be English, and that the document be dated after Jan. 1, 1997. The query requests all pages that have the keyword "cryptographic protocols" in the title, that are written in the English, and that are dated after Jan. 1, 1997.

In addition, to the types of data, there is an implicit or explicit Boolean operation to be performed on the different parts of the query. In the above query, there is an implicit "AND" operator among all of the query parts. In other words, implicitly the query specifies that the phrase "cryptographic protocols" appear in the content of the document AND that this match be in the title AND that the document's language be English AND that the document be dated after Jan. 1, 1997. A query may be formally defined to be a Boolean expression Q=(q_i) op (q_j), where op is a Boolean operation from the set {AND, OR AND NOT, BEFORE$_w$, NEAR$_w$}, and each of q_i, q_j, is either a Boolean expression, or of the form (m:k), where m is a (possibly empty) meta-data quantifier and k is a (possibly empty) keyword. The Boolean operators AND, OR and NOT have the standard meanings; x NEAR$_w$ y is TRUE if an only if x appears within w words of y; x BEFORE$_w$, y is TRUE if x appears at most w words before y.

A document satisfies or matches a query if it satisfies the Boolean expression Q, where satisfying (m:k) means that the keyword k satisfies (or appears in) meta data m. A keyword k can be a single word, a phrase, or a word with some wildcard characters. Meta-data can be any structural information such as title, heading, URL, domain, filename, file extension, date; it can also be a specification to the quality of the match required. For example, an approximate match meta-data operator (approx$_d$ y) evaluates to TRUE for a phrase x if and only if x can be transformed into y (or vice versa) with at most d single-letter deletes, insertions or substitutions; a synonyms operator (syn y) evaluates to TRUE for a phrase x if and only if x is a synonym of y.

A hierarchy forest may be defined on all the meta-data of a query, where each tree contains meta-data that has certain

relationship to other meta data. As shown in FIG. **9**(*a*), for example, the structural meta-data of an HTML document forms a natural hierarchical list from the tags that specify the most prominent information in the document, i.e. title, to tags that specify the least prominent information in the document, i.e. text. In FIG. **9**(*a*), requiring the keywords to appear in the title **900** is more restrictive than requiring the keywords to appear in the title or level **1** heading **902**. Cell **902**, in turn is more restrictive than requiring the keywords to appear in the title or level **1** or level **2** heading **904**. An even broader search query can be done with respect to cell **904**, which adds the level **2** headings to the query in cell **902**. The broadest most general query can be done with respect to cell **906**, which allows any text to be searched.

In general, each hierarchy tree is ordered top to bottom from meta-data values that most restrict the query to values that least restrict the query. For example, numerous hierarchies are appropriate for the date field, depending on the desired granularity. In the context of recent news stories, a daily granularity is appropriate as shown in FIG. **9**(*b*). Cell **910** restricts the search to documents dated after Jan. 2, 1997 while cell **912** restricts the search to documents dated after Jan. 1, 1997. A search with a cell **912** restriction is broader and encompasses the search with a cell **910** restriction. An even broader search can be done with a cell **914** restriction which includes documents dated after Dec. 31, 1996. On the other hand, in the context of general web pages, a yearly or biannual granularity is more relevant as shown in FIG. **9**(*c*). Cell **920** restricts the search to documents dated after Jul. 1, 1997 while cell **927** restricts the search to documents dated after Jan. 1, 1997. A broader search can be done with respect to documents dated after Jul. 1, 1996 in cell **924**.

As with the meta-data, there is a hierarchy for the keywords. For example, as shown in FIG. **10**(*a*), the top of the hierarchy is represented by cell **1010** and "keyword" corresponds to the most restrictive search query. Second on the hierarchy is cell **1012** corresponding to a broader search that can be done with the "all the English stemmings of keyword". Cell **1014** is at the bottom of the hierarchy and corresponds to the broadest search query related to "keyword or any of its synonyms".

Finally, a hierarchy on the Boolean operators that form the query Q is defined as follows. For a single-word keyword with or without wildcards, the hierarchy is shown in FIG. **10**(*a*). When the keyword is a phrase, it is converted into a Boolean expression to which the Boolean hierarchy applies. More specifically, if $k=w_1, w_2 \ldots w_t$, and w_i is the i-th word in the keyword phrase, m:k becomes $(m:w_1)$ $BEFORE_1$ $(m:w_2)$ $BEFORE_1$ . . . $BEFORE_1(m:w_t)$. Although not shown, the bottom-most node in each hierarchy is the NULL expression.

These query hierarchies may be used to help the user refine a given query more effectively. In the Internet's current state, the slowest operation for a user performing a search is the network delays in communicating with the search engine. In a typical search session, the user formulates a query, sends it to the search engine, waits some time, receives an answer, then reformulates the query and repeats the process. Some of the user's frustration comes from having to pay for the network delay during each query reformulation. The present invention cuts down the number of reformulation iterations used to find the relevant information. When the user poses a query, the browser generates a number of related queries and sends all the queries to the search engine in parallel. The time to receive the complete results for the users query and just the number of matches for each of the related queries is asymptotically the same as the time to receive the results of just the user's query. Referring to FIG. **2**A step **55**, the next step performed by the query tuner is to formulate the related queries. This process as well as how the results of the query aid the user is described below.

The formulation of related queries according the query hierarchies is illustrated based on a sample query $Q=((title;cryptographic)\ BEFORE_1(title;protocols))$ AND ((English language) AND (dated after Jan. 1, 1997)). The term item is used to refer to any atomic part of the query: a meta-datum, a keyword or a Boolean operator. For example, Q contains the following set of items {title, cryptographic, $BEFORE_1$, title, protocols, AND, English language, AND, dated after Jan. 1, 1997}. For each query item t, define h(t) to be the node in the hierarchy forest corresponding to the item t. Related queries consists of a set of queries, each of which takes the original user query and modifies some items in it by either restricting or broadening them according to the hierarchy forest. The act of broadening (restricting) a query item t corresponds to using a descendant (an ancestor) of h(t) in place of t within Q.

For example, one set of related queries for our sample query Q is shown in Table 3

Table 3

((title:cryptographic) $BEFORE_1$ (title: protocols))

((<h1> or title: cryptographic) $BEFORE_1$ (<h1> or title: protocols)) AND((English language) AND (dated after Jan. 1, 1997))

((title:cryptographic) $BEFORE_2$ (title: protocols)) AND ((English language) AND (dated after Jan. 1, 1997))

((title:cryptographic) $NEAR_1$ (title: protocols) AND ((English language) AND (dated after Jan. 1, 1997))

((title:cryptographic) $BEFORE_1$ (title: protocols)) AND (dated after Jan. 1, 1997))

((title:cryptographic) $BEFORE_1$ (title: protocols)) AND (English language))

((title:cryptographic) $BEFORE_1$ (title: protocols)) AND (English language) AND (dated after Jan. 1, 1997)

Where <h1> represents the main index level or the highest level heading in the HTML of the page.

The exemplary tree shown in FIG. **10**(*b*) indicates that the search can be contracted or restricted by moving up the tree. In addition, it indicates that the search can be expanded by moving down the tree. For example, a search limited to x AND y according to cell **1040** can be restricted by moving up the tree and searching according to cell **1050** where the search is restricted to x NEARn y. In contrast, the search can be expanded by moving down the tree and searching for only x according to cell **1030** or searching for only y according to cell **1035**. The search can be further expanded by searching for x OR y according to cell **1020**.

The generation of a set of related queries may be accomplished, for example, by holding all but one items of Q constant, while broadening or restricting the chosen item t of Q. The broadening (restricting) may be accomplished by traversing any number of edges up (down) the hierarchy tree from h(t). Since different edges in the hierarchy forest have different restrictive/broadening effects on the query, it is more efficient to traverse different number of edges in the tree for different items in the query. In the exemplary embodiment of the invention, the number of edges traversed is a fraction of the height of the hierarchy tree. Formally, an f-family is defined as a set of queries, each of which takes the original user query and modifies a set $\{t_1\ t_2 \ldots t_s\}$ of items in the query, where each modified item t_j is replaced by a node that is exactly $\min\{1, f*H\}$ edges away from $h(t_j)$