# AI Search Engine Implementation

Mike Murphy

November 7, 2004

## The Problem

A lack of a unified method for accessing technical support information for the Linux operating system has made it difficult for Linux users to find assistance on the Clemson campus. The available support resources for technological applications as a whole are scattered among various sites and servers, resulting in added difficulties for those who would prefer not to subsidize Billy of Redmond.

While the Clemson campus website is indexed by the Google search engine, it is difficult to find specific help with a standard search algorithm. The user often either is left out in the cold without any results whatsoever, or Google happily reports hundreds or even thousands of potential results, many of which are advertising gimmicks or are otherwise irrelevant. Achieving a searching goal in this kind of system requires considerable skill at query revision and manual filtering of results. Narrowing on a specific topic is challenging because the underlying search structure is apparently a flat collection of results indexed by keyword, with no hierarchical structure.

## General Solution Design

The first key to solving the above problems was to enforce some type of structure on otherwise un-structured data. For this application, a tree data structure was selected to provide this enforcement. Conceptually, the tree links together related *resources* at its leaf nodes, using interior nodes called *concepts*. In this case, "concept" is a literal term: these concept nodes capture the essence of the common thread that unites the resources under each concept.

With the data structure in place, searching the tree is a breadth-first process that matches keywords in the query to keywords at each resource node. If exactly one resource matches the query, then that resource is presented to the user. However, if the user's query does not produce a unique result, then the user is presented with one or two menu selections that help to narrow the query. Should no match occur, or should the user be unhappy with the available resources, a default item is available, which is contact information for a person who may be able to give additional assistance the user.

To help keep the query tree balanced, an advisory clustering algorithm was provided for the search tree administrator. This algorithm first calculates the average number of resources per concept, then it provides balancing advice on each concept that exceeds that average by a user-selectable amount. Shannon's information theory formula is used to determine the best means by which to split the universe of keywords within the resources inside each concept. Because of the inherent challenges of producing suitable divisions and associated user prompting via Artificial Intelligence algorithms, the balancing feature was limited to providing advice to the human administrator, who must make any actual changes manually.

The solution implementation includes basic Web-based administrative functions to add and edit concepts and resources. These administrative tools were implemented as extensions to the search tools themselves. Basic htaccess security, provided by the Apache HTTP server, limits the availability of the administrative components.

## Technologies Used

Implementing this solution required the use of several technologies, including Perl, Python, Bash shell scripting, and the HyperText Markup Language (HTML). The back-end engine that performs the searching was written in Python, while Bash scripts were used for data access and modification. Perl provided the Common Gateway Interface HTTP front-end. Though these scripting languages are not ideal in terms of performance, they were chosen because of the risk posed by bugs in a C-based CGI application. Issues that potentially could bring down the College of Engineering and Science server were not considered acceptable in this design.

## Algorithms

Since the tree has a fixed depth of two, with the keywords for search comparison stored at the leaf nodes, an uninformed breadth-first search provides the searching functionality. A user-selectable "any" or "all" keyword match is available on the query form. In the case of an "any" match, a resource node matches the query if any keyword in the query matches any keyword in the resource node. On the other hand, an "all" match requires that every keyword in the query match a keyword in the resource.

For resource discrimination within concepts (and concept discrimination, if necessary), a simple selection system is employed. The query string is actually still used here: any resources (or concepts) which do not match the query according to the matching algorithm, are excluded from the menu presented to the user. Each menu presentation further has a “default” choice that the user may select if he or she does not find an acceptable solution among the presented choices.

On the administrative side, several other algorithms are employed in order to facilitate the job of the database maintainer. Apart from scripts to edit and create new resources and concepts, the two major features of this tool suite are a balancing assistant and a visualizer. The balancing assistant is a tool that provides a suggestion to the administrator as to how the tree should be re-balanced when there are too many resources assigned to too few concepts. For this tool, the underlying algorithm is a clustering algorithm that uses Shannon’s information theory formula to group the resources of an offending node according to keyword discrimination. In this implementation, Shannon’s formula identifies (or at least attempts to identify) the keyword whose presence or absence from any given resource, splits the concept’s local universe of resources in the most optimal manner. Due to the limitations of the “most optimal,” the implementation only makes a recommendation: it is up to the administrator to make a reasonable adjustment to the search tree.

The other major administrative algorithm is the visualization algorithm, which produces an HTML unordered, hierarchical list of concepts and resources. Each concept or resource is output as a form, the submission of which provides access to each concept or resource for viewing or editing. A preorder traversal of the tree is used to drive the output.

## Language Challenges

Implementing this search engine system was not easy in large part due to the languages available for server-side Web applications. As configured by both the department and the college, the Apache server offers CGI support only for true CGI programs (i.e., written and compiled in a language like C) and Perl scripts. Because of stability issues mentioned earlier, it was decided that the Web front-end for this system would be written in Perl, with back-end scripts written in the simpler Python and Bash languages.

Achieving inter-script communication between scripts written in three different languages was not trivial. Because Perl has built-in security support, some scripts that work properly in shell mode do not run at all when run from the server. Numerous debugging steps and adjustments were needed in order to make Perl function at all. Then, there were still issues with getting Python and Bash scripts to output valid HTML (actually, XHTML) code, which had to be passed through the Perl scripts and out to the user. Not least among these challenges was the use of numerous escape sequences for such things as the double-quotes required around XHTML element field values.

## Server Issues

Producing a working prototype application was made ever more difficult by various subtle issues with Solaris, Apache, and security policies in use by the department. One rather silly issue that appeared in the code was the use of the "echo" command inside Bash scripts. On Linux and Mac OS X systems, echo can handle escape sequences for special characters, provided that the appropriate switch has been used. On Solaris, however, there are at least three different versions of echo available: the shell built-in, /bin/echo, and /usr/bin/echo. The former two versions do not support escape sequences even with the most magic of incantations, while the latter properly handles escape sequences even without the proper switch.

As a "feature" of the server, Apache runs all its child processes, including scripts, with the same user and group settings as Apache itself, despite the concept of service of multiple users in different groups. Thus, in order for any of the administration scripts to make any changes to files on disk, the database directory had to be set world-writable. There was no work-around to this problem, because (according to Jay Harris) Perl must be run setuid root for scripts to be able to drop permissions properly. As a result, a significant security hole was exposed in the process of implementing the demonstration prototype.

## Needed Changes

As a result of the server issues, it was determined that several changes would need to be made before this implementation can be deployed on the Clemson Linux Initiative website. First, the security hole created by the world-writable database directory must be fixed. Unless the College of Engineering and Science system administrator (Corey Ferrier) approves an alternate work-around, this will necessitate removal of the web-based administration suite completely. In this case, a console-based administration package would need to be created, so that the database could be administered via regular secure shell access to the server.

A second major change that may be needed prior to operational deployment is that some optimization may be necessary for large databases. At present, each concept and resource is stored as a collection of files, which are linked together in the tree structure at run-time. These multiple files cause multiple disk accesses, which may result in poor performance once the database reaches a significant size. Depending on the expected size of the resource collection, a user-space open-source relational database package could even be required.

Other needed changes are generally minor. An “exclusive-match-all” search filter, which could rank the suitability of a match-all match by the number of keywords actually matched, may be of use. In addition, some minor adjustments to the resource/concept discrimination steps may be needed to increase user accessibility. Apart from these minor adjustments, the only other significant task will be to make cosmetic adjustments to the search tools, so as to fit the CLI website.