**CS380 Information Retrieval**

**HW 4 / Group Project 2**

**Dates:**
**Topic selection: March 20**
**Report due April 8**
**Presentations to be made: April 6 and April 8 (?)**

**Groups:**
Given our virtual status, keeping the groups from project 1 is OK. However rearrangement is also fine.

**General Instructions:**
Unlike the first group project where the number of groups working on particular topics was limited, groups are free to choose topics without restriction. That said, unique topics and presentations might be expected to, on average, receive a slightly higher grade simply because they are not in direct competition with others.

**Topics:**
1. Web Crawling. Write a web crawler that crawls the Bryn Mawr and Haverford domains. The crawler should not crawl pages outside of these domains. The crawler should retrieve essentially 100% of the documents within these domains. The crawler should retrieve any page at most twice during crawling. (The crawler should also respect any robot restrictions.) One result of the crawl must be a list of every page (URL) found during the crawl along with its size and the date that the page last changed (as given in the page headers). This list if to be submitted electronically, NOT on paper. From all groups writing crawlers I will assemble a list of all the pages found. Part of the crawler grade will be based on the percentage of pages a specific crawler found in the merged list. Within the web crawling space you should choose any of the following specialities, or define your own specialty:
    1. Speed. Do everything you can to optimize the rate a which pages are crawled. Multithreading will be necessary. You will probably want to run on more than one computer. The presentation and paper should focus on how the crawler needed to be adapted make it faster.

    2. Topic direction. Order the crawl based on some desired topic. The crawl should still be complete, but the order of crawl should be different. The presentation and paper should focus on what adjustment the crawler needed to be topic directed and then on how the topic direction changed the ordering of the crawl.  You will likely need to do at least 2 complete crawls, with 2 different topics. You will need to track the exactly order in which pages were crawled so you can compare ordering between the crawls.
    3. Page-rank direction. Order the crawl based on the page rank estimate for the page.  This estimate should evolve while the crawl is in progress.  That is, do not do a complete crawl, then calculate the page rank for each page, then recrawl with a page ranks calculated from the first, blind crawl.  The presentation and paper should focus on how the page calculation can be done in parallel with a crawl and how that affects the crawl.
2. A client / server game.  Pick a simple 2 player game, for instance "connect 4".  Implement that game so that the principle play logic is on the server side.  Logic may be written in any language so long at it can be called from the command line. (You can execute any command line program from PHP. If you take this direction, see me.) The client will be within a standard web browser and may use javascript (Javascript is not required.)   However, the javascript and intelligence on the client side should be minimal. Essentially the only intelligence on the client side should be with respect to formatting the display. The server-side game play may be arbitrarily stupid.  The only requirement is that the server make legal moves. (Once things are working you may want to make the game play smarter  This project is about the client-server interactions rather than the quality of game play.
   Notes: This is possible to do remotely, but will be a challenge.  It is also possible to set up Macintosh computers to run both the client and server (PHP) locally, and only at the end install on CS department machines. Doing so is entirely up to you.
3. Snippet finder. One of the things that people use to evaluate a search result is a snippet of the text of the document that suggests why the document is relevant to the query.  Write a system to find snippets. For instance, a binary-search style approach would be to cut the document in half, evaluate both halves (with respect to the query) and pick the half with the higher score. Repeat until you have an appropriately sized snippet.  (This binary-search approach is flawed in

several ways but can be made to work.) Other approaches might work even better. (See for instance, section 8.7 of the text.)  While speed is certainly important and is a focus of the text, I am more interested in finding high quality snippets.