Crawlie – A general purpose web crawler

Features

- Custom priority/heuristic functionality to guide the crawler through the web
- Never go the same place twice!
- Massive parallelization, decide for yourself how far you want to go!
- Tree traversal back to seed from any discovered page
- Automatically store your results to a neat database (SQLite as of now)
- Download discovered files based on suffix, URL or regex matching
- Simple graphical user interface
- Automatically save your current session and all working data and resume later!
- Loosely coupled and easily maintainable architecture

How to use

Jar file

Download the jar, configure the properties file and run.

Source

Download the source, the required libraries and run the Crawlie.java class.

Required libraries

- guava-16.0.1
- jsoup-1.7.3
- sqlite-jdbc-3.7.2

How it works

The crawler starts from a seed, then checks all the links on that page, and continues to the url that yielded the highest priority given the configuration. This can be searching for pdf-files, url that include "hello" etc. It will periodically store all the analyzed pages to a database. Due to the database becoming insanely heavy I've chosen not to include the source files. The crawler can also mark files/urls for download. For instance, one scenario could be: Go to hello.com and download all the images on that domain.

Potential weaknesses

- Requires massive amounts of computer power if you intend to use it extensively
- High, monotonically increasing, memory usage due to having to cache discovered URLs in order to not enter a crawler loop

Future improvements

The improvements and potential is highly dependent on the line of work the crawler is used for. However, some specifics might be:

- Implement a GUI for the configuration
- Use big data analytics to make sense of data collected
- Implement a relevance filter to prioritize pages based on content rather than URL

How to expand

Expansion is easy. It's already heavily parallelized with SWING (the GUI) being the biggest bottleneck. If instead of writing to a local database, decentralized instances could write to one common one. Depending on the implementation this could be very suitable for clusters and other multi- and/or super-computers.