N-Sums: A Framework for Web-based Search Engines

Apisitt Rattana and Andrew Davison
Dept. of Computer Engineering
Prince of Songkla University
Hat Yai, Songkhla 90112, Thailand
E-mail: dandrew@ratree.psu.ac.th

Abstract

The N-Sums framework is aimed at making the design, implementation, and maintenance of specialized domain search engines easier, with the resulting applications able to produce more useful matches and less bad hits. The heart of N-Sums is the realization that effective search requires a combination of search strategies.

ComicSearch is a specialized domain search engine for finding comic covers and other information, built using the N-Sums framework. It has outperformed the popular Copernic general purpose search engine in all of its tests.

1. Introduction

N-Sums (*Niche-Search Using Multiple Strategies*) is a framework for creating *effective* Web search engines for specialized domains (e.g. finding comic books, finding the latest soccer scores), By effective, we mean that the engine will return matches containing a high percentage of URLs pointing to the requested information and a low percentage of links to poorly related or unrelated data,

At the heart of N-Sums is the recognition that a variety of different search mechanisms must be combined in order to obtain good search results. This differs from the present trend in search engine design which hopes that 'more of the same' will lead to improved answers.

In the following sections, we describe our framework, and a particular example of its use, the ComicSearch search engine. We compare ComicSearch with one of the leading 'super-search' engines, Copernic [3], and see that ComicSearch performs significantly better within its chosen domain.

2. Four Search Strategies

N-Sums identifies four categories of Web search, each with their own advantages and disadvantages:

- general purpose search engines;
- specialized domain search engines;
- direct search of static/dynamic Web pages;
- local database(s) of search information.

N-Sums combines these to emphasize their positive features while reducing their negative ones.

2.1. General Purpose Search Engines

General purpose search engines, such as Google and AltaVista, possess many advantages: they have simple query interfaces (usually phrase-based, with boolean operators), results are presented in a structured format, typically including the match's title, a text summary, and a score, and their internal indexes are relatively current.

Also grouped into this category are 'super-search' engines, such as Copernic. Copernic passes a user's query onto 15 other search engines (e.g. Google, Hotbot), and collects their results. The main advantage of Copernic-like applications are their increased coverage of the Web by utilizing multiple engines.

Despite their utility, general purpose search engines have some problems. A common one is the weakness of their query languages (e.g. they almost never allow regular expressions). A consequence of poor query expressiveness is the increased chance of obtaining bad hits (URLs pointing to unrelated information). This is especially true if the queries contain common words (e.g. mad, action, battle) or words with different meanings in different contexts (e.g. wolverine, superman). Applications like Copernic make this problem worse by requiring users to formulate queries in a lowest common denominator notation that is understandable to all of their component engines. 'Super-search' engines also increase the possibility of duplicate hits even though they make some attempt to filter them out.

Another problem with general purpose search engines is the slow update frequency of their internal indexes, caused by the need to re-examine large parts of the Web. This update rate can be inadequate for some kinds of search, such as sports scores or cinema movie listings, which are highly transient.

General purpose search engines make a strong selling point of their Web coverage – Google currently advertises an index of over 1.3 billion pages (early March 2001). However, current estimates place the number of pages at between 3 and 4 billion, and growing rapidly [2]. General purpose search engines will never be able to index every page, and this will only become more apparent as the Web continues to grow.

The extensive coverage and weak query power of these engines means that numerous matches are returned (often in the 1000's). The engines attempt to score the matches, and supply summaries of the pages, to alleviate the user's feeling of being swamped by data. Even so, the user usually has to perform a manual, second search by scrolling through the returned matches, trying to select links which are really useful. At best this will require a reading of the summaries, at worst it involves downloading the full page for a result to examine it in detail; this is time-consuming and wasteful of resources.

2.2. Specialized Domain Search Engines

This kind of search engine focuses on a particular domain, a popular example being AuctionHawk (`http://www.auctionhawk.com`), a 'super-search' engine which limits its searches to online auctions. It passes its queries to the search engines attached to Ebay, Amazon, Yahoo, and over 100 other sites.

Advantages of specialized domain search engines include the reduction of bad hits since topics outside the domain are not examined, a potential for increased speed since the domain is smaller, and the likelihood that the domain's index is more current. The reduction in bad hits often means that a user will not have to carry out a second search of the results. A specialized domain offers the opportunity to refine the query language. For example, AuctionHawk allows searches to be restricted to auctions which include images.

An obvious problem with specialized domain search engines is that they may exclude certain kinds of data of interest to the user.

2.3. Searching Web Pages Directly

Directly searching Web pages is often overlooked since most pages do not hold collections of data. Examples which can be usefully searched include pages of recent sports scores, pages holding catalogs of images, and links pages.

Web pages of this kind can be divided into two groups: *static* and *dynamic*. Static Web pages are those where the information does not change, or changes very rarely. Web pages holding comic cover images are usually static, since covers do not change once published. Dynamic Web pages change frequently, as with pages holding soccer scores.

In fact, all Web pages change over time: perhaps they are reformatted or moved to another location. The static label is something of a fiction, but is still useful as it suggests that the repeated search of static Web pages can be replaced by the search of locally-held data extracted from those pages. This approach is examined in the next subsection.

An advantage of searching Web pages directly is the opportunity to tailor the search to the page format and content, and to filter the information more accurately. This requires tools which are not found in most programming languages, such as regular expressions for pattern matching on unstructured text, and the ability to parse HTML for searching over the structured parts of a page. Some research Web languages contain these capabilities [e.g. 6], and there are several Java classes that offer this functionality [1, 5].

A major drawback is the difficulty of writing search code that can handle unpredictable changes in the pages. Robust code should detect change, perhaps by comparing page metrics gathered during the current search (e.g. the page modification time, page size) against earlier values. Another drawback is the need to develop search code for each Web page under consideration.

2.4. Local Storage of Search Information

It is useful if the data in a static Web page is (manually) extracted and its contents and format converted to a local form. The advantages are that the engine designer can decide on the local data format and its search capabilities. The outcome is that the engine can quickly examine the local database rather than carry out network communication with many Web pages, each with their own format and search requirements.

A significant drawback is how to devise a data format that can encompass disparate Web pages. It must be succinct, but also easy to maintain. This latter point is required to handle the inevitable changes to the data, the addition of new sources, and the disappearance of others.

A local database may also be useful for simplifying the search of *dynamic* Web pages. For example, the database may contain the URLs of the pages together with the regular expressions for searching them. In effect, the database stores meta-level information about the pages, which is likely to change more slowly than the data itself.

3. Development Strategies

In this section, we describe in overview how N-Sums is applied when designing, implementing, and maintaining a search engine.

3.1. Design Issues

A N-Sums search engine will usually employ one general purpose search engine to carry out a through global search as a backup to the specialized domain searches done by its other components. Crucial design elements for the general purpose and specialized domain searches are the development of suitable queries, and a way of judging (or scoring) the results they give. The primary aim is to develop queries and filters which reduce or eliminate the need for a second, manual search of the engine's results, and to remove the need to examine the results pages.

The process of query design is akin to the refinement of test cases in program validation. Queries must be formulated which adequately cover the search domain and test the various capabilities of the search engine. Disappointing results for the test queries can be utilized in two ways: the queries can be modified (e.g. by adding extra keywords), or the choice of search engine can be altered. This latter approach means that the designer should try out a range of engines. It may not be possible to modify the query in a satisfactory manner, due to the inherent limitations of the engine's query interface. In that situation, the results must be filtered after being returned.

For a general purpose search engine, a query must be complex enough to remove bad hits. When choosing specialized domain search engines, the goal is coverage. A single specialized domain engine will probably not be able to handle all the likely queries.

Designing the search of Web pages falls into two distinct categories, page retrieval and filtering. Filtering can be based on regular expression matching for unstructured text, parsing and regular expressions for more structured formats such as results tables.

3.2. Implementation Issues

A N-Sums application will typically employ one thread (or process) for each search component, and a further thread for the GUI interface. It will be necessary to coordinate the search threads when they are passing their results to the GUI and when the user presses the start or stop buttons for the searches. These requirements are easily addressed by programming in Java.

A real-world issue is network communication through a firewall; Java has support for proxies and user authentication. Most search engines accept information encoded using the HTTP GET protocol, but some forms-based engines require the POST protocol [4]. Both of these are supported by Java's networking classes.

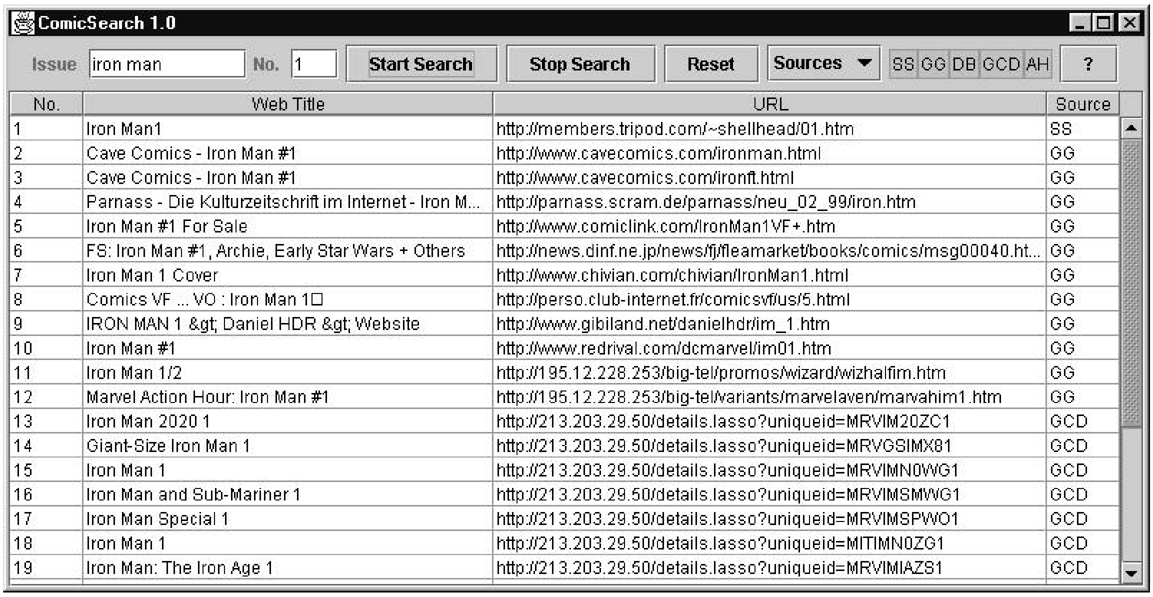
3.3. Maintenance Issues

Maintenance is a pressing problem for search engines because of the dynamic nature of the Web. The application should include a utility for testing the liveness of its components. At the simplest level, the continued existence of the component search engines and Web pages must be monitored. A more complex task is to detect when they change. For search engines, the query and/or the results formats may change.

A related point is the discovery of new sources of information, which requires repeated surveys of the Web.

4. ComicSearch

ComicSearch finds information about comics – the user enters a comic title (or a phrase from the title) and issue number of interest, and the search engine returns a table of URLs. The user can click on a URL to open it in the system's default browser. The results of a query for "iron man" issue 1 are shown in Figure 1. "iron man" matches against all the comics which contain that phrase (e.g. "Giant-Size Iron Man"), but only details on the comics with the given issue number are returned.



Issue: iron man

No.: 1

No.	Web Title	URL	Source
1	Iron Man1	http://members.tripod.com/~shellhead/01.htm	SS
2	Cave Comics - Iron Man #1	http://www.cavecomics.com/ironman.html	GG
3	Cave Comics - Iron Man #1	http://www.cavecomics.com/ironft.html	GG
4	Parnass - Die Kulturzeitschrift im Internet - Iron M...	http://parnass.scram.de/parnass/neu_02_99/iron.htm	GG
5	Iron Man #1 For Sale	http://www.comiclink.com/IronMan1VF+.htm	GG
6	FS: Iron Man #1, Archie, Early Star Wars + Others	http://news.dinf.ne.jp/news/fj/fleamarket/books/comics/msg00040.ht...	GG
7	Iron Man 1 Cover	http://www.chivian.com/chivian/IronMan1.html	GG
8	Comics VF ... VO : Iron Man 1□	http://perso.club-internet.fr/comicsvf/us/5.html	GG
9	IRON MAN 1 &gt; Daniel HDR &gt; Website	http://www.gibiland.net/danielhdr/im_1.htm	GG
10	Iron Man #1	http://www.redrival.com/dcmarvel/im01.htm	GG
11	Iron Man 1/2	http://195.12.228.253/big-tel/promos/wizard/wizhalfim.htm	GG
12	Marvel Action Hour: Iron Man #1	http://195.12.228.253/big-tel/variants/marvelaven/marvahim1.htm	GG
13	Iron Man 2020 1	http://213.203.29.50/details.lasso?uniqueid=MRVIM20ZC1	GCD
14	Giant-Size Iron Man 1	http://213.203.29.50/details.lasso?uniqueid=MRVGSIMX81	GCD
15	Iron Man 1	http://213.203.29.50/details.lasso?uniqueid=MRVIMN0WG1	GCD
16	Iron Man and Sub-Mariner 1	http://213.203.29.50/details.lasso?uniqueid=MRVIMSMWG1	GCD
17	Iron Man Special 1	http://213.203.29.50/details.lasso?uniqueid=MRVIMSPWO1	GCD
18	Iron Man 1	http://213.203.29.50/details.lasso?uniqueid=MITIMN0ZG1	GCD
19	Iron Man: The Iron Age 1	http://213.203.29.50/details.lasso?uniqueid=MRVIMIAZS1	GCD

Figure 1. ComicSearch Results for "iron man" 1.

ComicSearch is primarily aimed at US comics produced during the so-called Golden, Silver, and Bronze ages (roughly 1939 – 1970). These comics are the main concern of collectors. ComicSearch is based on a similar search engine, MCCSE, which concentrated on the Marvel Comic publishing company [7].

Google is used as the general purpose search engine, and acts as a backup to three specialized domain search engines, AuctionHawk, the Grand Comic Database (GCD, `http://213.203.29.50/`), and the search engine at Nick Simon's Marvel Silver Age site (`http://www.geocities.com/Area51/Zone/4414/index.html`). GCD stores author and artist information on over 64,000 comics, but has many gaps in its data, some errors, and a poor selection of cover images. It is also somewhat daunting for non-technical users due to its complex query interface. Nick Simon's site focuses on a single US publisher, Marvel, and a particular era (about 1956 to 1970), but is very comprehensive within those boundaries.

ComicSearch comes with a small database of useful comic sites (currently about 40). For each site, a list is given of the comic titles which can be found there, together with issue numbers and URLs. However, storing a single URL for each issue would quickly lead to a very large database. Instead, a format was designed around storing issue *ranges* and URL *patterns* using place-holders. For example:

```
Title: The Mighty Banana
Issues: 1 5 7 9-205 1004
imageURL: http://foo.com/mb**.html
```

These three lines represent 201 URLs for comics. At run time, ComicSearch substitutes the issue number for the place-holding *'s in the image URL, left-padding it with 0's if necessary. A search for "The Mighty Banana" issue 7 will return the URL `http://foo.com/mb07.html`. Issues with more than two digits are substituted with no padding (e.g. `http://foo.com/mb1004.html`).

The database format was designed to be understandable by its users, the intention being that they could extend it. The online documentation for ComicSearch encourages users to submit their information by e-mail for inclusion in future releases of the application.

ComicSearch utilizes Java threads to query its component search engines and local database. An important aspect of the threads is the filtering of the results returned by the engines. For example, the Google thread uses a regular expression based on the comic title followed by spaces or letters or a '#' and then the issue number. This relatively simple pattern is applied to the title lines of the Google results, and filters out 50-70% of the bad hits on average, compared with the unfiltered results. ComicSearch employs the SteveSoft regular expression class [2] for this, although much can be achieved with Java's String class alone. The number of good hits filtered out is typically quite small, about 5%. A similar technique is used in the AuctionHawk thread, but was unnecessary in the GCD and Nick Simon threads due to their accuracy.

A specially coded networking class lets ComicSearch operate through proxies/firewalls, and deals with user authentication.

There are no secondary searches of the results by visiting their Web pages. The success of the filters makes this extra step unnecessary. Also there is no direct searching of Web pages. Since the comic sites are quite static, their details were converted into database entries.

The specialized domain search engines do not return duplicate URLs since they search in different places on the Web, but Google does occasionally return a few hits found by the others. The removal of duplicate URLs from the results table would be straightforward, but is not currently carried out. However, the *contents* of the results pages do often overlap, but this is not seen as a bad thing – it is quite useful to have repeated information, images, etc. from different sources, to allow comparisons between them. For example in figure 1, the results rows 1, 2, 3, 5, 7, 8, 9, 10, 15, and 18 all refer to the same comic, but the details on the pages vary, such as where to buy the comic, it's cost, the publication history, and the cover image size and quality.

ComicSearch was made freely available over the Internet in March 2001. It can be obtained from `http://fivedots.coe.psu.ac.th/~ad/ComicSearch/readme.html`.

5. Comparisons with Copernic

We have run extensive tests of ComicSearch and compared them with Copernic, the popular 'super-search' engine (`http://www.copernic.com`).

Partial Title and issue no.	*No. of exact matches*	*Rows where the exact matches occur*	*No. of related matches*	*No. of unrelated matches*	*Total no. of matches*
Wolverine 1	1	32	27	40	68
Superman 100	2	2, 7	10	35	47
Spider-man 122	4	1, 22, 28, 40	17	45	66
Mad 99	0		0	58	58
Green Lantern 59	1	3	26	23	50
Flash 13	0		1	70	71

Table 1. Copernic results for six queries.

Table 1 shows six typical comic queries, the number of exact matches, related matches, unrelated matches, and the total number of matches returned by Copernic. An exact match is a URL to a page which describes a comic containing the partial title string with the given issue number, related matches are URLs to pages about the general comic or its characters. Unrelated matches have nothing to do with the comic. The 'rows' column gives the row positions of the exact matches in Copernic's output after it had been sorted into decreasing order by score. The query input was a (partial) comic title and issue number, separated by a space.

Partial Title and issue no.	*No. of exact matches*	*Sources which supplied the exact matches*	*No. of related matches*	*No. of unrelated matches*	*Total no. of matches*
Wolverine 1	37	GG/18, AH/13, GCD/6	4	3	43
Superman 100	10	AH/4, DB/3, GG/2, GCD/1	0	0	10
Spider-man 122	9	GCD/3, GG/3, DB/2, AH/1	0	0	9
Mad 99	3	GG/2, DB/1	0	2	5
Green Lantern 59	8	DB/3, GCD/3, AH/2	0	0	8
Flash 13	11	DB/4, GCD/3, GG/3, AH/1	0	19	30

Table 2. ComicSearch results for six queries.

Table 2 shows the same queries processed by ComicSearch. A 'rows' column is not included, partly because the output from ComicSearch is ordered nondeterministically due to its threaded behaviour. The other reason is that the exact matches almost always appear before the related or unrelated ones, a consequence of the slow response rate of Google. Instead, a 'sources' column is given, which details the number of exact matches contributed by each search thread. AH is AuctionHawk, DB is the local database, GCD is the Grand Comic Database, GG is Google, and NS is Nick Simon's site.

ComicSearch produces fewer results than Copernic, but the quality is higher; quality can be measured as the percentage of exact matches in the total number of matches. Another indicator is the coverage of the exact matches. Queries such as "flash 13" can match on many different comics which use the word "flash" in their titles, and different volumes within the same comic title. ComicSearch frequently returns at least one link to each of these possibilities.

The presence of so many unrelated matches in Copernic's output can make using it quite tiresome: finding the best URLs frequently involves a lengthy study of the 50 or more results. This indicates that usability is as much affected by the number of bad hits as the number of good ones.

The number of unrelated matches generated by Copernic points to a difficulty when searching for comics using general purpose search engines – comic titles regularly use common words such as 'flash' and 'mad'. Even a word like 'wolverine' has a number of meanings; in this instance, a US football team and the Canadian animal.

ComicSearch is helped *and* hindered by the presence of Google, which accounts for all the unrelated matches, but also occasionally turns up exact matches missed by the other searches. Fortunately, the number of unrelated matches from Google is significantly reduced by having ComicSearch filter its results.

The local database contributes exact matches in almost all the example queries, but its success depends on the choice of queries. AuctionHawk returns exact matches quite often, but this depends on the auctions currently in progress; when an auction finishes, the information will disappear soon after. Nick Simon's search engine does not return anything for the test queries, but this is because the titles and issues are outside its range of interest. GCD information can sometimes be rather brief.

6. Conclusions

The N-Sums framework rests on the idea that effective search engines for specific domains (e.g. comics, soccer scores) are best designed using a combination of search strategies. These utilize:

- general purpose search engines;
- specialized domain search engines;
- direct search of static/dynamic Web pages;
- local database(s) of search information.

We described how these different approaches have consequences for the design, implementation, and maintenance of the resulting search engine.

We used the N-Sums framework to build ComicSearch, a search engine focussing on US comics from the 1940's to the 1970's. It performs significantly better than general purpose search engines due to its multi-faceted approach to finding results. ComicSearch was made freely available on the Web in March 2001.

Our future work will utilize N-Sums to build three more search engines: one for finding celebrity mailing addresses, one for returning the latest soccer scores for any team, and a tool for finding online resources for computing textbooks (e.g. slides, software, exercises). We expect that these efforts will indicate further avenues for the refinement of N-Sums.

References

[1] BRANDT, S.R. "Regular Expressions in Java", Package `com.stevesoft.pat` version 1.4. Available at `http://javaregex.com/`, March 2001.

[2] CLIENT HELP DESK. "Web Statistics: Size, the Average Page", Available at `http://www.clienthelpdesk.com/statistics_research/web_statistics.html`, March 2001

[3] COPERNIC TECHNOLOGIES. *Copernic 2001 Basic*, version 5. Available at `http://www.copernic.com`, March 2001.

[4] FIELDING, R., GETTYS, J., MOGUL, J., FRYSTYK, H., MASINTER, L., LEACH, P., BERNERS-LEE, T. "Hypertext Transfer Protocol – HTTP/1.1", RFC 2616, Available at `ftp://ftp.isi.edu/in-notes/rfc2616.txt`, 1999.

[5] HOTHOUSE OBJECTS. "Tags: HTML Toolkit for Java", version 1.0.5. Available at `http://www.hothouseobjects.com/`, March 2001.

[6] KISTLER, T. AND MARAIS, H. "WebL – A Programming Language for the Web", SRC Research Report, Digital Systems Research Center, Palo Alto, California, USA., 1998.

[7] RATTANA, A. "Marvel Comics Cover Search Engine (MCCSE)", Senior Project, Dept. of Comp. Eng., PSU, Hat Yai, Thailand, February 2001.