

Jaypee University of Information Technology
Waknaghat, Distt. Solan (H.P.)

Learning Resource Center

CLASS NUM:

BOOK NUM.:

ACCESSION NO.: SP09038/SP0913038

This book was issued is overdue due on the date stamped below. If the book is kept over due, a fine will be charged as per the library rules.

Due Date	Due Date	Due Date

Analysis of The Invisible Web

Project Report Submitted in partial fulfillment of the Degree of

Bachelor of Technology

In

Computer Science Engineering

Under the Supervision of

Mr. Suman Saha

By

Shivaca Thakur (091293)

Kamaljit Kaur Virk (091321)

To



Jaypee University of Information and Technology

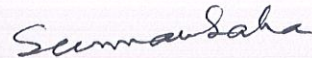
Waknaghat, Solan – 173234, Himachal Pradesh



CERTIFICATE

This is to certify that the work titled **Analysis of Invisible Web** submitted by **Shivaca Thakur (091293) & Kamaljit Kaur Virk (091321)** in partial fulfillment for the award of degree of **Bachelor of Technology** of Jaypee University of Information Technology, Waknaghat has been carried out under my supervision. This work has not been submitted partially or wholly to any other University or Institute for the award of this or any other degree or diploma.

Signature of Guide:



Name of Guide:

Mr. Suman Saha

Designation: **Lecturer**

Date: **May 16, 2012**

ACKNOWLEDGEMENT

This project could not have been at the stage it is right now had it not been for the cooperation of Mr. Suman Saha, our project guide, who was always there to tell us how to go about our project in a systematic manner and who always took out time to help us with our technical and non-technical doubts at various stages of the project.

Equally important was the contribution of Dr. Nitin, our project supervisor, who kept faith in our ability to complete the project well and on time.

Last but not the least; it was our fellow students who came to our rescue whenever we got stuck in any piece of code or otherwise.

Names of Students:

Shivaca Thakur (091293)

Kamaljit Kaur Virk (091321)

Date:

May 16, 2012

Contents

1. Certificate	2
2. Acknowledgement	3
3. Introduction	5
4. Deep web resources	9
5. Accessing the Deep Web	10
6. Crawling the Deep Web	11
7. Classifying Resources	12
8. Search Engine	13
9. How Search Engines Work?	14
• Web crawler	15
• Search Engine Indexers	18
• Query processor	19
10. Issues with Search Engines	20
• Cost of Crawling	20
• “Dump” Crawlers	20
• User Expectations & Skills	21
• Speedy Response vs Thorough results	22
• Bias towards Text	22
11. Why Search engines can't see the Invisible Web?	23
• Case1-7	24
12. Surface Web vs Invisible Web	30
13. Types of Invisibility	31
• Opaque web	32
• Private Web	32
• Proprietary Web	33
• Truly Invisible Web	33
14. Technologies	35
15. Project Design	37
16. Sample code of a Web Crawler	39
17. References	49

Introduction

The surface Web (also known as the visible Web or indexable Web) is that portion of the World Wide Web that is indexable by conventional search engines. The part of the Web that is not reachable this way is called the Deep Web. Search engines construct a database of the Web by using programs called spiders or Web crawlers that begin with a list of known Web pages.

The spider gets a copy of each page and indexes it, storing useful information that will let the page be quickly retrieved again later. Any hyperlinks to new pages are added to the list of pages to be crawled. Eventually all reachable pages are indexed, unless the spider runs out of time or disk space. The collection of reachable pages defines the Surface Web.

For various reasons (e.g., the Robots Exclusion Standard, links generated by JavaScript and Flash, password-protection) some pages cannot be reached by the spider. These 'invisible' pages are referred to as the Deep Web.

A 2005 study queried the Google, MSN, Yahoo!, and Ask Jeeves search engines with search terms from 75 different languages and determined that there were over 11.5 billion web pages in the publicly indexable Web as of January 2005.

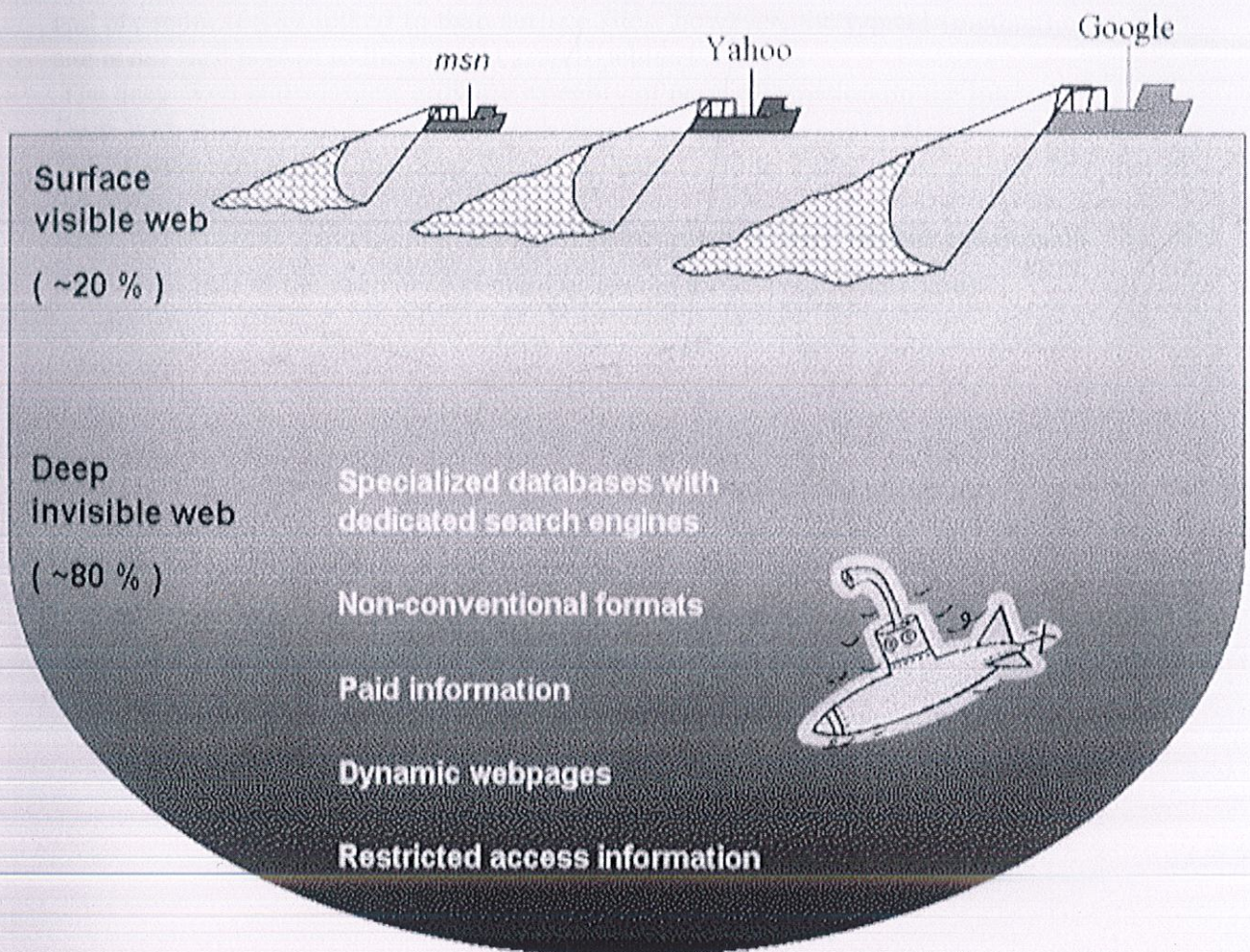
As of June 2008, the indexed web contains at least 6'309 billion pages.

The Deep Web (also called the Deepnet, the Invisible Web, the Undernet or the hidden Web) is World Wide Web content that is not part of the Surface Web, which is indexed by standard search engines.

It should not be confused with the dark Internet, the computers that can no longer be reached via Internet, or with the distributed file sharing network Darknet, which could be classified as a smaller part of the Deep Web.

One of the pages includes "Deep Web Sites" which indicates that the 60 known, largest deep Web sites contain data of about 750 terabytes (HTML included basis), or roughly 40 times the size of the known surface Web. These sites appear in a broad array of domains from science to law to images and commerce. The total number of records or documents within this group is about 85 billion.

Mike Bergman, founder of BrightPlanet, credited with coining the phrase, said that searching on the Internet today can be compared to dragging a net across the surface of the ocean: a great deal may be caught in the net, but there is a wealth of information that is deep and therefore missed. Most of the Web's information is buried far down on dynamically generated sites, and standard search engines do not find it. Traditional search engines cannot "see" or retrieve content in the deep Web—those pages do not exist until they are created dynamically as the result of a specific search. The deep Web is several orders of magnitude larger than the surface Web.

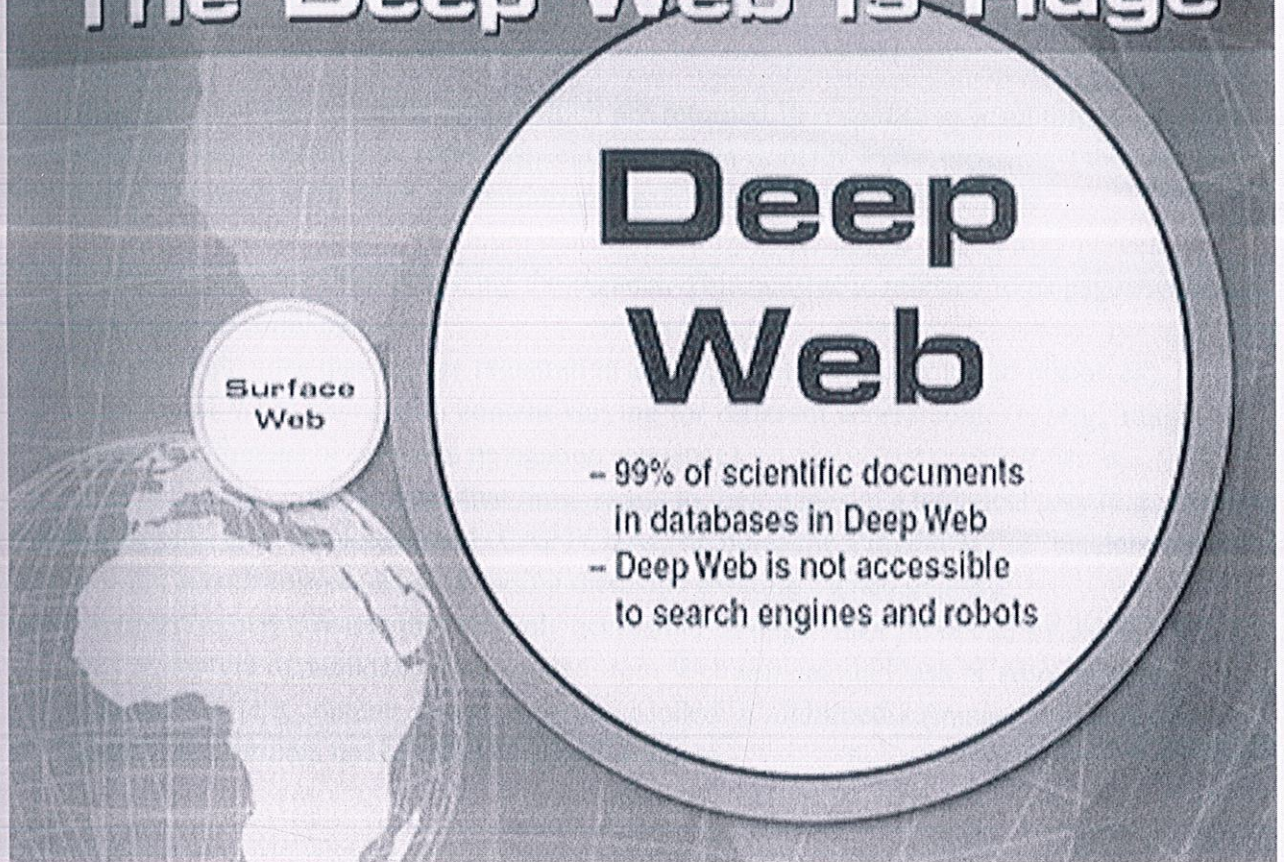


Juanicó – Environmental Consultants Ltd.

Basically, the folks at BrightPlanet found that "Deep Web sources store their content in searchable databases that only produce results dynamically in response to a direct request." Ordinary "spider" indexing of "surface" web sites misses this content, which BrightPlanet says is truly vast:

- Public information on the deep Web is currently 400 to 550 times larger than the commonly defined World Wide Web
- The deep Web contains 7,500 terabytes of information, compared to 19 terabytes of information in the surface Web
- The deep Web contains nearly 550 billion individual documents compared to the 1 billion of the surface Web
- More than an estimated 100,000 deep Web sites presently exist
- 60 of the largest deep Web sites collectively contain about 750 terabytes of information – sufficient by themselves to exceed the size of the surface Web by 40 times
- On average, deep Web sites receive about 50% greater monthly traffic than surface sites and are more highly linked to than surface sites; however, the typical (median) deep Web site is not well known to the Internet search public
- The deep Web is the largest growing category of new information on the Internet
- Deep Web sites tend to be narrower with deeper content than conventional surface sites
- Total quality content of the deep Web is at least 1,000 to 2,000 times greater than that of the surface Web
- Deep Web content is highly relevant to every information need, market and domain
- More than half of the deep Web content resides in topic specific databases

The Deep Web Is Huge



Deep Web Resources

Deep Web resources may be classified into one or more of the following categories:

- Dynamic content: dynamic pages which are returned in response to a submitted query or accessed only through a form, especially if open-domain input elements (such as text fields) are used; such fields are hard to navigate without domain knowledge.
- Unlinked content: pages which are not linked to by other pages, which may prevent Web crawling programs from accessing the content. This content is referred to as pages without backlinks (or inlinks).
- Private Web: sites that require registration and login (password-protected resources).
- Contextual Web: pages with content varying for different access contexts (e.g., ranges of client IP addresses or previous navigation sequence).
- Limited access content: sites that limit access to their pages in a technical way (e.g., using the Robots Exclusion Standard, CAPTCHAs, or no-cache Pragma HTTP headers which prohibit search engines from browsing them and creating cached copies).
- Scripted content: pages that are only accessible through links produced by JavaScript as well as content dynamically downloaded from Web servers via Flash or Ajax solutions.
- Non-HTML/text content: textual content encoded in multimedia (image or video) files or specific file formats not handled by search engines.

Accessing the deep web

To discover content on the Web, search engines use web crawlers that follow hyperlinks through known protocol virtual port numbers. This technique is ideal for discovering resources on the surface Web but is often ineffective at finding deep Web resources. For example, these crawlers do not attempt to find dynamic pages that are the result of database queries due to the infinite number of queries that are possible. It has been noted that this can be (partially) overcome by providing links to query results, but this could unintentionally inflate the popularity for a member of the deep Web.

In 2005, Yahoo! made a small part of the deep Web searchable by releasing Yahoo! Subscriptions. This search engine searches through a few subscription-only Web sites. Some subscription websites display their full content to search engine robots so they will show up in user searches, but then show users a login or subscription page when they click a link from the search engine results page.

DeepPeep, Intute, Deep Web Technologies, and Scirus are a few search engines that have accessed the deep web. Intute ran out of funding and is now a temporary static archive as of July, 2011.

Crawling the deep Web

Researchers have been exploring how the deep Web can be crawled in an automatic fashion. In 2001, Sriram Raghavan and Hector Garcia-Molina presented an architectural model for a hidden-Web crawler that used key terms provided by users or collected from the query interfaces to query a Web form and crawl the deep Web resources.

Alexandros Ntoulas, Petros Zerfos, and Junghoo Cho of UCLA created a hidden-Web crawler that automatically generated meaningful queries to issue against search forms. Several form query languages (e.g., DEQUEL) have been proposed that, besides issuing a query, also allow to extract structured data from result pages. Another effort is DeepPeep, a project of the University of Utah sponsored by the National Science Foundation, which gathered hidden-Web sources (Web forms) in different domains based on novel focused crawler techniques.

Commercial search engines have begun exploring alternative methods to crawl the deep Web. The Sitemap Protocol (first developed by Google) and mod oai are mechanisms that allow search engines and other interested parties to discover deep Web resources on particular Web servers.

Both mechanisms allow Web servers to advertise the URLs that are accessible on them, thereby allowing automatic discovery of resources that are not directly linked to the surface Web. Google's deep Web surfacing system pre-computes submissions for each HTML form and adds the resulting HTML pages into the

Google search engine index. The surfaced results account for a thousand queries per second to deep Web content.

In this system, the pre-computation of submissions is done using three algorithms:

- (1) selecting input values for text search inputs that accept keywords,
- (2) identifying inputs which accept only values of a specific type (e.g., date), and
- (3) selecting a small number of input combinations that generate URLs suitable for inclusion into the Web search index.

Classifying resources

Automatically determining if a Web resource is a member of the surface Web or the deep Web is difficult. If a resource is indexed by a search engine, it is not necessarily a member of the surface Web, because the resource could have been found using another method (e.g., the Sitemap Protocol, mod oai, OAIster) instead of traditional crawling.

If a search engine provides a backlink for a resource, one may assume that the resource is in the surface Web. Unfortunately, search engines do not always provide all backlinks to resources. Even if a backlink does exist, there is no way to determine if the resource providing the link is itself in the surface Web without crawling all of the Web. Furthermore, a resource may reside in the surface Web, but it has not yet been found by a search engine.

Therefore, if we have an arbitrary resource, we cannot know for sure if the resource resides in the surface Web or deep Web without a complete crawl of the Web.

Most of the work of classifying search results has been in categorizing the surface Web by topic. For classification of deep Web resources, Ipeirotis et al. presented an algorithm that classifies a deep Web site into the category that generates the largest number of hits for some carefully selected, topically-focused queries. Deep Web directories under development include OAIster at the University of Michigan, Intute at the University of Manchester, Infomine at the University of California at Riverside, and DirectSearch (by Gary Price). This classification poses a challenge while searching the deep Web whereby two levels of categorization are required.

The first level is to categorize sites into vertical topics (e.g., health, travel, automobiles) and sub-topics according to the nature of the content underlying their databases.

The more difficult challenge is to categorize and map the information extracted from multiple deep Web sources according to end-user needs. Deep Web search reports cannot display URLs like traditional search reports. End users expect their search tools to not only find what they are looking for quickly, but to be intuitive and user-friendly.

In order to be meaningful, the search reports have to offer some depth to the nature of content that underlie the sources or else the end-user will be lost in the sea of URLs that do not indicate what content lies beneath them. The format in which search results are to be presented varies widely by the particular topic of the search and the type of content being exposed.

The challenge is to find and map similar data elements from multiple disparate sources so that search results may be exposed in a unified format on the search report irrespective of their source.

Search Engine

Search engines are databases containing full-text indexes of Web pages. When you use a search engine, you are actually searching this database of retrieved Web pages, not the Web itself. Search engine databases are finely tuned to provide rapid results, which is impossible if the engines were to attempt to search the billions of pages on the Web in real time.

Search engines are similar to telephone white pages, which contain simple listings of names and addresses. Unlike yellow pages, which are organized by category and often include a lot of descriptive information about businesses, white pages provide minimal, bare bones information. However, they're organized in a way that makes it very easy to look up an address simply by using a name like "Smith" or "Veerhoven."

Search engines are compiled by software "robots" that voraciously suck millions of pages into their indices every day. When you search an index, you're trying to coax it to find a good match between the keywords you type in and all of the words contained in the search engine's database. In essence, you're relying on a computer to essentially do simple patternmatching between your search terms and the words in the index. AltaVista, HotBot, and Google are examples of search engines.

How Search Engines Work

Search engines are complex programs. In a nutshell, they consist of several distinct parts:

- The Web crawler (or spider), which finds and fetches Web pages
- The indexer, which as its name implies, indexes every word on every page and stores the resulting index of words in a huge database
- The query processor, which compares your search query to the index and recommends the best possible matching documents

Let's take a closer look at each part.

Web Crawler

Web crawlers are the “scouts” for search engines, with the sole mission of finding and retrieving pages on the Web and handing them off to the search engine’s indexers, which we discuss in the next section. It’s easy to imagine a Web crawler as a little sprite scuttling across the luminous strands of cyberspace, but in reality Web crawlers do not traverse the Web at all.

In fact, crawlers function much like your Web browser, by sending a request to a Web server for a Web page, downloading the entire page, and then handing it off to the search engine’s indexer. Crawlers, of course, request and fetch pages much more quickly than you can with a Web browser. In fact most Web crawlers can request hundreds or even thousands of unique pages simultaneously.

Given this power, most crawlers are programmed to spread out their requests for pages from individual servers over a period of time to avoid overwhelming the server or consuming so much bandwidth that human users are crowded out.

Crawlers find pages in two ways. Most search engines have an “add URL” form, which allows Web authors to notify the search engine of a Web page’s address. In the early days of the Web, this method for alerting a search engine to the existence of a new Web page worked well—the crawler simply took the list of all URLs submitted and retrieved the underlying pages.

Unfortunately, spammers figured out how to create automated bots that bombarded the add URL form with millions of URLs pointing to spam pages. Most search engines now reject almost 95 percent of all URLs submitted through their add URL forms. It’s likely that, over time, most search engines will phase-out their add URL forms in favor of the Second method that crawlers can use to discover pages—one that’s more easy to control.

This second method of Web page discovery takes advantage of the hypertext links embedded in most Web pages. When a crawler fetches a page, it culls all of the links appearing on the page and adds them to a queue for subsequent crawling. As the crawler works its way through the queue, links found on each new page are also added to the queue. Harvesting links from actual Web pages dramatically decreases the amount of spam a crawler encounters, because most Web authors only link to what they believe are high-quality pages.

By harvesting links from every page it encounters, a crawler can quickly build a list of links that can cover broad reaches of the Web. This technique also allows crawlers to probe deep within individual sites, following internal navigation links. In theory, a crawler can discover and index virtually every page on a site starting from a single URL, if the site is well designed with extensive internal navigation links.

Although their function is simple, crawlers must be programmed to handle several challenges. First, since most crawlers send out simultaneous requests for thousands of pages, the queue of “visit soon” URLs must be constantly examined and compared with URLs already existing in the search engine’s index. Duplicates in the queue must be eliminated to prevent the crawler from fetching the same page more than once.

If a Web page has already been crawled and indexed, the crawler must determine if enough time has passed to justify revisiting the page, to assure that the most up-to-date copy is in the index. And because crawling is a resource-intensive operation that costs money, most search engines limit the number of pages that will be crawled and indexed from any one Web site. This is a crucial point—you can’t assume that just because a search engine indexes some pages from a site that it indexes all of the site’s pages.

Because much of the Web is highly connected via hypertext links, crawling can be surprisingly efficient.

The behavior of a Web crawler is the outcome of a combination of policies:

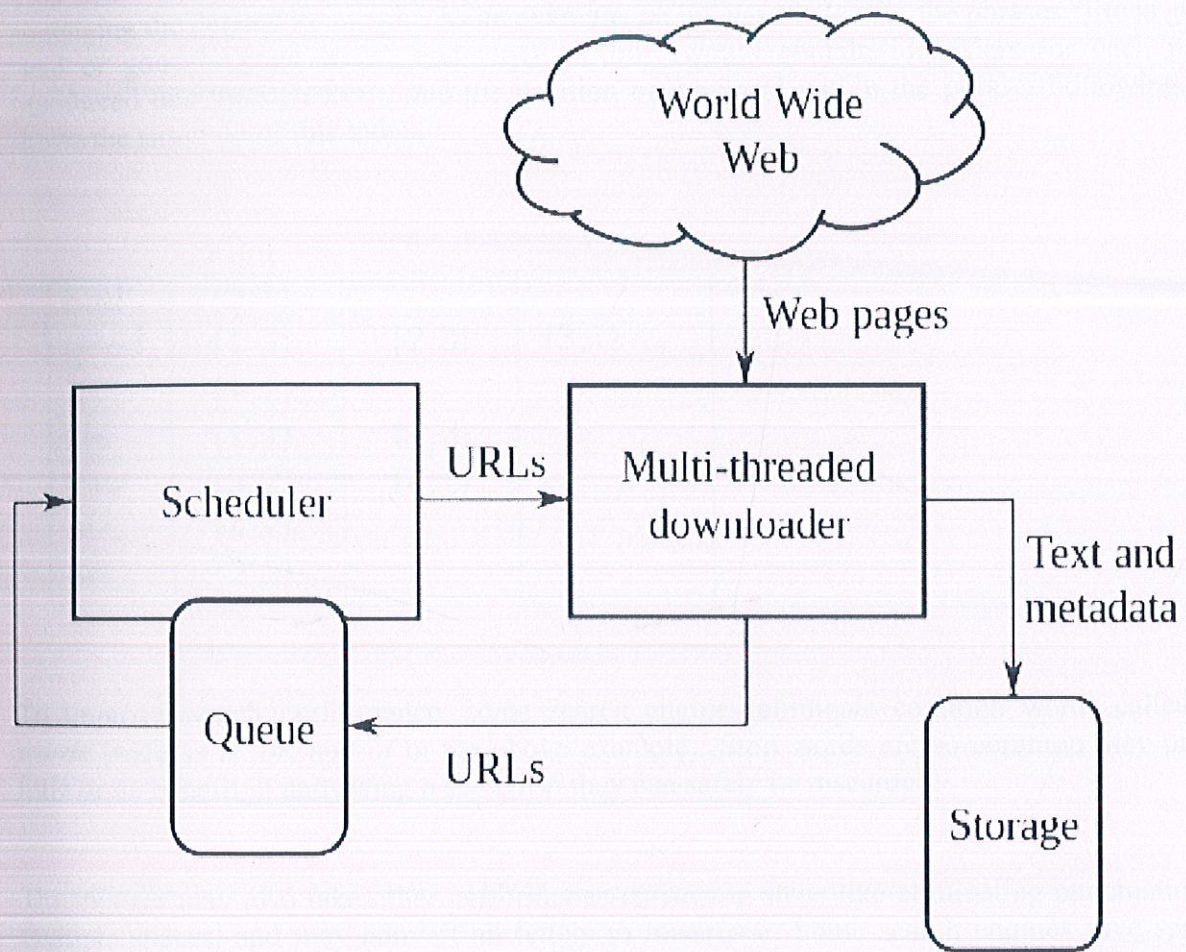
- a selection policy that states which pages to download,
- a re-visit policy that states when to check for changes to the pages,
- a politeness policy that states how to avoid overloading Web sites, and
- a parallelization policy that states how to coordinate distributed Web crawlers.

A May 2000 study published by researchers at AltaVista, Compaq, and IBM drew several interesting conclusions that demonstrate that crawling can, in theory, discover most pages on the visible Web (Broder et al., 2000).

The study found that:

- For any randomly chosen source and destination page, the probability that a direct hyperlink path exists from the source to the destination is only 24 percent.
- If a direct hypertext path does exist between randomly chosen pages, its average length is 16 links. In other words, a Web browser would have to click links on 16 pages to get from random page A to random page B. This finding is less than the 19 degrees of separation postulated in a previous study, but also excludes the 76 percent of pages lacking direct paths.
- If an undirected path exists (meaning that links can be followed forward or backward, a technique available to search engine spiders but not to a person using a Web browser), its average length is about six degrees.

- More than 90 percent of all pages on the Web are reachable from one another by following either forward or backward links. This is good news for search engines attempting to create comprehensive indexes of the Web.



Search Engine Indexers

When a crawler fetches a page, it hands it off to an indexer, which stores the full text of the page in the search engine's database, typically in an inverted index data structure.

An inverted index is sorted alphabetically, with each index entry storing the word, a list of the documents in which the word appears, and in some cases the actual locations within the text where the word occurs.

This structure is ideally suited to keyword-based queries, providing rapid access to documents containing the desired keywords. As an example, an inverted index for the phrases "life is good," "bad or good," "good love," and "love of life" would contain identifiers for each phrase (numbered one through four), and the position of the word within the phrase. Following table shows the structure of this index.

bad	(2,1)		
good	(1,3)	(2,3)	(3,1)
is	(1,2)		
life	(1,1)	(4,3)	
love	(3,2)	(4,1)	
of	(4,2)		
or	(2,2)		

To improve search performance, some search engines eliminate common words called *stop words* (such as *is*, *or*, and *of* in the above example). Stop words are so common they provide little or no benefit in narrowing a search so they can safely be discarded.

The indexer may also take other performance-enhancing steps like eliminating punctuation and multiple spaces, and may convert all letters to lowercase. Some search engines save space in their indexes by truncating words to their root form, relying on the query processor to expand queries by adding suffixes to the root forms of search terms.

Indexing the full text of Web pages allows a search engine to go beyond simply matching single keywords. If the location of each word is recorded, proximity operators such as NEAR can be used to limit searches. The engine can also match multi-word phrases, sentences, or even larger chunks of text. If a search engine indexes HTML code in addition to the text on the page, searches can also be limited to specific fields on a page, such as the title, URL, body, and so on.

The Query Processor

The query processor is arguably the most complex part of a search engine. The query processor has several parts, including the primary user interface (the search form), the actual “engine” that evaluates a query and matches it with the most relevant documents in the search engine database of indexed Web pages, and the results-output formatter.

The search form and the results format vary little from search engine to search engine. All have both basic and advanced search forms, each offering slightly varying limiting, control, and other user-specified functions.

And most result formats are equally similar, typically displaying search results and a few additional extras like suggested related searches, most popular searches, and so on.

The major differentiator of one search engine from another lies in the way relevance is calculated. Each engine is unique, emphasizing certain variables and downplaying others to calculate the relevance of a document as it pertains to a query.

Some engines rely heavily on statistical analysis of text, performing sophisticated pattern-matching comparisons to find the most relevant documents for a query.

Others use link analysis, attempting to capture the collective wisdom of the Web by finding the documents most cited by other Web authors for a particular query. How an engine calculates relevance is ultimately what forms its “personality” and determines its suitability for handling a particular type of query.

Search engine companies closely guard the formulas used to calculate relevance, and change them constantly as algorithms are updated for improved quality or tweaked to outwit the latest technique used by spammers.

Nonetheless, over time, a searcher can generally get to know how well a particular engine will perform for a query, and select an engine appropriately.

Issues with Search Engines

Just as Web directories have a set of issues of concern to a searcher, so do search engines. Some of these issues are technical; others have to do with choices made by the architects and engineers who create and maintain the engines.

Cost of crawling:

Crawling the Web is a resource-intensive operation. The search engine provider must maintain computers with sufficient power and processing capability to keep up with the explosive growth of the Web, as well as a high-speed connection to the Internet backbone. It costs money every time a page is fetched and stored in the search engine's database. There are also costs associated with query processing, but in general crawling is by far the most expensive part of maintaining a search engine.

Since no search engine's resources are unlimited, decisions must be made to keep the cost of crawling within an acceptable budgetary range. Some engines limit the total number of pages in their index, dumping older pages when newer ones are found. Others limit the frequency of recrawl, so pages in the index may be stale or out of date. Still others limit their crawling to certain portions or domains believed to contain reliable, non-duplicated material.

Whenever an engine decides to limit its crawling, it means pages that potentially could be included in its index are not. It's tempting to think that these unretrieved pages are part of the Invisible Web, but they aren't. They are visible and indexable, but the search engines have made a conscious decision not to index them. Competent searchers must take this into account when planning a research strategy. Much has been made of these overlooked pages, and many of the major engines are making serious efforts to include them and make their indexes more comprehensive. Unfortunately, the engines have also discovered through their "deep crawls" that there's a tremendous amount of duplication and spam on the Web. The trade-off between excluding bogus material and assuring that all truly relevant material will be found is a difficult one, and virtually assures that no engine will ever have a totally comprehensive index of the Web.

"Dumb" crawlers:

At their most basic level, crawlers are uncomplicated programs. Crawlers are designed simply to find and fetch Web pages. To discover unindexed pages, crawlers rely on links they've discovered on other pages. If a Web page has no links pointing to it, a search engine spider cannot retrieve it unless it has been submitted to the search engine's "add URL" form. Another problem with search engines compiled by crawlers is simply that it takes a lot of time to crawl the entire Web, even when the crawler is hitting millions of pages a day. Crawler lag time is a two-fold issue:

First, there's typically a gap between when a page is published on the Web and when a crawler discovers it.

Second, there's a time lag between when a crawler first finds a page and when it recrawls the page looking for fresh content.

Both of these time issues can contribute to incomplete or inaccurate search results. Current generation crawlers also have little ability to determine the quality or appropriateness of a Web page, or whether it is a page that changes frequently and should be recrawled on a timely basis.

User expectations and skills:

Users often have unrealistic expectations of what search engines can do and the data that they contain. Trying to determine the best handful of documents from a corpus of millions or billions of pages, using just a few keywords in a query is almost an impossible task.

Yet most searchers do little more than enter simple two- or three-word queries, and rarely take advantage of the advanced limiting and control functions all search engines offer. Search engines go to great lengths to successfully cope with these woefully basic queries.

One way they cope is to create a set of preprogrammed results for popular queries. For example, if a pop star releases a hit song that generates lots of interest, the engine may be preprogrammed to respond with results pointing to biographical information about the star, discographies, and links to other related music resources.

Another method is to adjust results so that the most popular, commonly selected pages rise to the top of a result lists. These are just two techniques search engines apply to help users who can't or won't take advantage of the powerful tools that are available to them.

Taking the time to learn how a search engine works, and taking advantage of the full capabilities it offers can improve search results dramatically.

Speedy Response vs. Thorough Results:

Now that we live on “Internet time,” everyone expects nearly instantaneous results from search engines. To accommodate this demand for speed, search engines rarely do as thorough an analysis as they might if time were not an issue. Shortcuts are taken, total results are truncated, and invariably important documents will be omitted from a result list.

Fortunately, increases in both processing power and bandwidth are providing search engines with the capability to use more computationally intensive techniques without sacrificing the need for speed.

Unfortunately, the relentless growth of the Web works against improvements in computing power and bandwidth simply because as the Web grows the search space required to fully evaluate it also increases.

Bias toward text:

Most current-generation search engines are highly optimized to index *text*. If there is no text on a page—say it’s nothing but a graphic image, or a sound or audio file—there is nothing for the engine to index.

For non-text objects such as images, audio, video, or other streaming media files, a search engine can record, in an Archielike manner, filename and location details but not much more. While researchers are working on techniques for indexing non-text objects, for the time being non-text objects make up a considerable part of the Invisible Web.

Why search engines cannot see the Invisible Web

Text—more specifically hypertext—is the fundamental medium of the Web. The primary function of search engines is to help users locate hypertext documents of interest. Search engines are highly tuned and optimized to deal with text pages, and even more specifically, text pages that have been encoded with the HyperText Markup Language (HTML).

As the Web evolves and additional media become commonplace, search engines will undoubtedly offer new ways of searching for this information. But for now, the core function of most Web search engines is to help users locate text documents.

HTML documents are simple. Each page has two parts: a “head” and a “body,” which are clearly separated in the source code of an HTML page. The head portion contains a title, which is

displayed (logically enough) in the title bar at the very top of a browser's window. The head portion may also contain some additional metadata describing the document, which can be used by a search engine to help classify the document. For the most part, other than the title, the head of a document contains information and data that help the Web browser display the page but is irrelevant to a search engine. The body portion contains the actual document itself. This is the meat that the search engine wants to digest.

The simplicity of this format makes it easy for search engines to retrieve HTML documents, index every word on every page, and store them in huge databases that can be searched on demand. Problems arise when content doesn't conform to this simple Web page model. To understand why, it's helpful to consider the process of crawling and the factors that influence whether a page either can or will be successfully crawled and indexed.

The first determination a crawler attempts to make is whether access to pages on a server it is attempting to crawl is restricted. Webmasters can use three methods to prevent a search engine from indexing a page.

Two methods use blocking techniques specified in the Robots Exclusion Protocol (<http://info.webcrawler.com/mak/projects/robots.html>) that most crawlers voluntarily honor and one creates a technical roadblock that cannot be circumvented.

The Robots Exclusion Protocol is a set of rules that enables a Webmaster to specify which parts of a server are open to search engine crawlers, and which parts are off-limits. The Webmaster simply creates a list of files or directories that should not be crawled or indexed, and saves this list on the server in a file named robots.txt. This optional file, stored by convention at the top level of a Web site, is nothing more than a polite request to the crawler to keep out, but most major search engines respect the protocol and will not index files specified in robots.txt.

The second means of preventing a page from being indexed works in the same way as the robots.txt file, but is page-specific. Webmasters can prevent a page from being crawled by including a "noindex" meta tag instruction in the "head" portion of the document. Either robots.txt or the noindex meta tag can be used to block crawlers. The only difference between the two is that the noindex meta tag is page specific, while the robots.txt file can be used to prevent indexing of individual pages, groups of files, or even entire Web sites.

Password protecting a page is the third means of preventing it from being crawled and indexed by a search engine. This technique is much stronger than the first two because it uses a technical barrier rather than a voluntary standard.

Why would a Webmaster block crawlers from a page using the Robots Exclusion Protocol rather than simply password protecting the pages?

Password-protected pages can be accessed only by the select few users who know the password. Pages excluded from engines using the Robots Exclusion Protocol, on the other hand, can be

accessed by anyone *except* a search engine crawler. The most common reason Webmasters block pages from indexing is that their content changes so frequently that the engines cannot keep up.

Pages using any of the three methods described here are part of the Invisible Web. In many cases, they contain no technical roadblocks that prevent crawlers from spidering and indexing the page. They are part of the Invisible Web because the Webmaster has opted to keep them out of the search engines.

Once a crawler has determined whether it is permitted to access a page, the next step is to attempt to fetch it and hand it off to the search engine's indexer component. This crucial step determines whether a page is visible or invisible. Let's examine some variations that crawlers encounter as they discover pages on the Web, using the same logic they do to determine whether a page is indexable.

Case 1.

The crawler encounters a page that is straightforward HTML text, possibly including basic Web graphics. This is the most common type of Web page. It is visible and can be indexed.

Case 2.

The crawler encounters a page made up of HTML, but it's a form consisting of text fields, check boxes, or other components requiring user input. It might be a sign-in page, requiring a user name and password. It might be a form requiring the selection of one or more options. The form itself, since it's made up of simple HTML, can be fetched and indexed. But the content behind the form (what the user sees after clicking the submit button) may be invisible to a search engine. There are two possibilities here:

- The form is used simply to select user preferences. Other pages on the site consist of straightforward HTML that can be crawled and indexed (presuming there are links from other pages elsewhere on the Web pointing to the pages). In this case, the form and the content behind it are visible and can be included in a search engine index.

A good example is Hoover's Business Profiles (<http://www.hoovers.com>), which provides a form to search for a company, but presents company profiles in straightforward HTML that can be indexed.

- The form is used to collect user-specified information that will generate dynamic pages when the information is submitted. In this case, although the form is visible the content "behind" it is invisible. Since the only way to access the content is by using the form, how can a crawler—which is simply designed to request and fetch pages—possibly know what to enter into the form?

Since forms can literally have infinite variations, if they function to access dynamic content they are essentially roadblocks for crawlers. A good example of this type of Invisible Web site is The World Bank Group's Economics of Tobacco Control Country Data Report Database, which allows you to select any country and choose a wide range of reports for that country (<http://www1.worldbank.org/tobacco/database.asp>). It's interesting to note that this database is just one part of a much larger site, the bulk of which is fully visible.

So even if the search engines do a comprehensive job of indexing the visible part of the site, this valuable information still remains hidden to all but those searchers who visit the site and discover the database on their own.

In the future, forms will pose less of a challenge to search engines. Several projects are underway aimed at creating more intelligent crawlers that can fill out forms and retrieve information. One approach uses preprogrammed "brokers" designed to interact with the forms of specific databases. Other approaches combine brute force with artificial intelligence to "guess" what to enter into forms, allowing the crawler to punch through the form and retrieve information.

However, even if general-purpose search engines do acquire the ability to crawl content in databases, it's likely that the native search tools provided by each database will remain the best way to interact with them.

Case 3.

The crawler encounters a dynamically generated page assembled and displayed on demand. The telltale sign of a dynamically generated page is the "?" symbol appearing in its URL. Technically, these pages are part of the visible Web. Crawlers can fetch any page that can be displayed in a Web browser, regardless of whether it's a static page stored on a server or generated dynamically.

A good example of this type of Invisible Web site is Compaq's experimental SpeechBot search engine, which indexes audio and video content using speech recognition, and converts the streaming media files to viewable text (<http://www.speechbot.com>). Somewhat ironically, one could make a good argument that most search engine result pages are themselves Invisible Web content, since they generate dynamic pages on the fly in response to user search terms.

Dynamically generated pages pose a challenge for crawlers.

Dynamic pages are created by a script, a computer program that selects from various options to assemble a customized page. Until the script is actually run, a crawler has no way of knowing what it will actually do. The script should simply assemble a customized Web page.

Unfortunately, unethical Webmasters have created scripts to generate millions of similar but not quite identical pages in an effort to "spamdex" the search engine with bogus pages. Sloppy

programming can also result in a script that puts a spider into an endless loop, repeatedly retrieving the same page.

These “spider traps” can be a real drag on the engines, so most have simply made the decision not to crawl or index URLs that generate dynamic content. They’re “apartheid” pages on the Web—separate but equal, making up a big portion of the “opaque” Web that potentially can be indexed but is not. Inktomi’s FAQ about its crawler, named “Slurp,” offers this explanation:

“Slurp now has the ability to crawl dynamic links or dynamically generated documents. It will not, however, crawl them by default. There are a number of good reasons for this. A couple of reasons are that dynamically generated documents can make up infinite URL spaces, and that dynamically generated links and documents can be different for every retrieval so there is no use in indexing them” (<http://www.inktomi.com/slurp.html>).

As crawler technology improves, it’s likely that one type of dynamically generated content will increasingly be crawled and indexed. This is content that essentially consists of static pages that are stored in databases for production efficiency reasons. As search engines learn which sites providing dynamically generated content can be trusted not to subject crawlers to spider traps, content from these sites will begin to appear in search engine indices.

For now, most dynamically generated content is squarely in the realm of the Invisible Web.

Case 4.

The crawler encounters an HTML page with nothing to index. There are thousands, if not millions, of pages that have a basic HTML framework, but which contain only Flash, images in the .gif, .jpeg, or other Web graphics format, streaming media, or other non-text content in the body of the page.

These types of pages are truly parts of the Invisible Web because there’s nothing for the search engine to index. Specialized multimedia search engines, such as ditto.com and WebSeek are able to recognize some of these non-text file types and index minimal information about them, such as file name and size, but these are far from keyword searchable solutions.

Case 5.

The crawler encounters a site offering dynamic, real-time data. There are a wide variety of sites providing this kind of information, ranging from real-time stock quotes to airline flight arrival information.

These sites are also part of the Invisible Web, because these data streams are, from a practical standpoint, unindexable. While it's technically possible to index many kinds of real-time data streams, the value would only be for historical purposes, and the enormous amount of data captured would quickly strain a search engine's storage capacity, so it's a futile exercise.

A good example of this type of Invisible Web site is TheTrip.com's Flight tracker, which provides real-time flight arrival information taken directly from the cockpit of in-flight airplanes (<http://www.trip.com/ft/home/0,2096,1-1,00.shtml>).

Case 6.

The crawler encounters a PDF or Postscript file. PDF and Postscript are text formats that preserve the look of a document and display it identically regardless of the type of computer used to view it. Technically, it's a straightforward task to convert a PDF or Postscript file to plain text that can be indexed by a search engine.

However, most search engines have chosen not to go to the time and expense of indexing files of this type.

One reason is that most documents in these formats are technical or academic papers, useful to a small community of scholars but irrelevant to the majority of search engine users, though this is changing as governments increasingly adopt the PDF format for their official documents.

Another reason is the expense of conversion to plain text. Search engine companies must make business decisions on how best to allocate resources, and typically they elect not to work with these formats.

An experimental search engine called Research Index, created by computer scientists at the NEC Research Institute, not only indexes PDF and Postscript files, it also takes advantage of the unique features that commonly appear in documents using the format to improve search results (<http://www.researchindex.com>).

For example, academic papers typically cite other documents, and include lists of references to related material. In addition to indexing the full text of documents, Research Index also creates a citation index that makes it easy to locate related documents. It also appears that citation searching has little overlap with keyword searching, so combining the two can greatly enhance the relevance of results.

We hope that the major search engines will follow Google's example and gradually adopt the pioneering work being done by the developers of Research Index. Until then, files in PDF or Postscript format remain firmly in the realm of the Invisible Web.

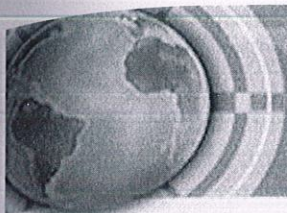
Case 7.

The crawler encounters a database offering a Web interface. There are tens of thousands of databases containing extremely valuable information available via the Web. But search engines cannot index the material in them. Although we present this as a unique case, Web-accessible databases are essentially a combination of Cases 2 and 3.

Databases generate Web pages dynamically, responding to commands issued through an HTML form. Though the interface to the database is an HTML form, the database itself may have been created before the development of HTML, and its legacy system is incompatible with protocols used by the engines, or they may require registration to access the data. Finally, they may be proprietary, accessible only to select users, or users who have paid a fee for access.

Ironically, the original HTTP specification developed by Tim Berners-Lee included a feature called format negotiation that allowed a client to say what kinds of data it could handle and allow a server to return data in any acceptable format. Berners-Lee's vision encompassed the information in the Invisible Web, but this vision—at least from a search engine standpoint—has largely been unrealized.

Type of Invisible Web Content	Why It's Invisible
Disconnected page	No links for crawlers to find the page
Page consisting primarily of images, audio, or video	Insufficient text for the search engine to "understand" what the page is about
Pages consisting primarily of PDF or Postscript, Flash, Shockwave, Executables (programs) or Compressed files (.zip, .tar, etc.)	Technically indexable, but usually ignored, primarily for business or policy reasons
Content in relational databases	Crawlers can't fill out required fields in interactive forms
Real-time content	Ephemeral data; huge quantities; rapidly changing information
Dynamically generated content	Customized content is irrelevant for most searchers; fear of "spider traps"



Surface Web vs Deep Web

2006

- Size: Estimated to be 8+ billion (Google) to 45 billion (About.com) web pages

- Static, crawlable web pages

- Large amounts of unfiltered information

- Limited to what is easily found by search engines

- Size: Estimated to be 5 to 500 times larger (BrightPlanet)

- Dynamically generated content that lives inside databases

- High-quality, managed, subject-specific content

- Growing faster than surface web (BrightPlanet)

Four Types of Invisibility

Technical reasons aside, there are other reasons that some kinds of material that can be accessed either on or via the Internet are not included in search engines. There are really four “types” of Invisible Web content. We make these distinctions not so much to make hard and fast distinctions between the types, but rather to help illustrate the Amorphous boundary of the Invisible Web that makes defining it in concrete terms so difficult.

The four types of invisibility are:

- The Opaque Web
- The Private Web
- The Proprietary Web
- The Truly Invisible Web

The Opaque Web

The Opaque Web consists of files that can be, but are not, included in search engine indices. The Opaque Web is quite large, and presents a unique challenge to a searcher. Whereas the deep content in many truly Invisible Web sites is accessible if you know how to find it, material on the Opaque Web is often much harder to find.

The biggest part of the Opaque Web consists of files that the search engines can crawl and index, but simply do not. There are a variety of reasons for this; let's look at them.

- Depth of Crawl
- Frequency of Crawl
- Maximum Number of Viewable Results
- Disconnected URL'S

The Private Web

The Private Web consists of technically indexable Web pages that have deliberately been excluded from search engines. There are three ways that Webmasters can exclude a page from a search engine:

- Password protect the page. A search engine spider cannot go past the form that requires a username and password.
- Use the robots.txt file to disallow a search spider from accessing the page.
- Use the "noindex" meta tag to prevent the spider from reading past the head portion of the page and indexing the body.

For the most part, the Private Web is of little concern to most searchers. Private Web pages simply use the public Web as an efficient delivery and access medium, but in general are not intended for use beyond the people who have permission to access the pages.

There are other types of pages that have restricted access that may be of interest to searchers, yet they typically aren't included in search engine indices. These pages are part of the Proprietary Web, which we describe next.

The Proprietary Web

Search engines cannot for the most part access pages on the Proprietary Web, because they are only accessible to people who have agreed to special terms in exchange for viewing the content.

Proprietary pages may simply be content that's only accessible to users willing to register to view them. Registration in many cases is free, but a search crawler clearly cannot satisfy the requirements of even the simplest registration process.

Examples of free proprietary Web sites include *The New York Times*, Salon's "The Well" community, Infonautics' "Company Sleuth" site, and countless others.

Other types of proprietary content are available only for a fee, whether on a per-page basis or via some sort of subscription mechanism.

Examples of proprietary fee-based Web sites include the Electric Library, Northern Light's Special Collection Documents, and *The Wall Street Journal* Interactive Edition.

Proprietary Web services are not the same as traditional online information providers, such as Dialog, LexisNexis, and Dow Jones. These services offer Web access to proprietary information, but use legacy database systems that existed long before the Web came into being.

While the content offered by these services is exceptional, they are not considered to be Web or Internet providers.

The Truly Invisible Web

Some Web sites or pages are truly invisible, meaning that there are technical reasons that search engines can't spider or index the material they have to offer. A definition of what constitutes a truly invisible resource must necessarily be somewhat fluid, since the engines are constantly improving and adapting their methods to embrace new types of content. But at the time of writing this book, truly invisible content consisted of several types of resources.

The simplest, and least likely to remain invisible over time, are Web pages that use file formats that current generation Web crawlers aren't programmed to handle. These file formats include PDF, Postscript, Flash, Shockwave, executables (programs), and compressed files. There are two reasons search engines do not currently index these types of files.

First, the files have little or no textual context, so it's difficult to categorize them, or compare them for relevance to other text documents. The addition of metadata to the HTML container

carrying the file could solve this problem, but it would nonetheless be the metadata description that got indexed rather than the contents of the file itself.

The second reason certain types of files don't appear in search indices is simply because the search engines have chosen to omit them. They can be indexed, but aren't. You can see a great example of this in action with the Research Index engine, which retrieves and indices PDF, postscript, and even compressed files in real time, creating a searchable database that's specific to your query.

AltaVista's Search Engine product for creating local site search services is capable of indexing more than 250 file formats, but the flagship public search engine includes only a few of these formats. It's typically lack of willingness, not an ability issue with file formats.

More problematic are dynamically generated Web pages. Again, in some cases, it's not a technical problem but rather unwillingness on the part of the engines to index this type of content. This occurs specifically when a non-interactive script is used to generate a page. These are static pages, and generate static HTML that the engine could spider.

The problem is that unscrupulous use of scripts can also lead crawlers into "spider traps" where the spider is literally trapped within a huge site of thousands, if not millions, of pages designed solely to spam the search engine. This is a major problem for the engines, so they've simply opted not to index URLs that contain script commands.

Finally, information stored in relational databases, which cannot be extracted without a specific query to the database, is truly invisible. Crawlers aren't programmed to understand either the database structure or the command language used to extract information.

How technologies these days are changing a part of invisible web to the visible web?

Amazon

Amazon.com is hoping to leverage human intelligence with a new service called Amazon Mechanical Turk. The name comes from Wolfgang von Kempelen's Turk, which was supposedly a chess-playing automaton (in the late 1700s) but actually had a human inside it. Amazon's director of Web service software Peter Cohen pointed to the company's A9 search service and its yellow pages feature. That service offers users photographs of, for example, pizza restaurants near specific addresses. But he said that asking a computer to choose the best one from a number of possible images isn't practical. A person, on the other hand, could make such a decision in seconds.

Also known as HIT's-Human intelligence tasks!

Google

But now search engines like Google have come up with a way to search images

Google analyses the image, creating a mathematical model based on shapes, lines, proportions, colors and other elements, it then matches the model against images already in the Google's index. Google then does page analysis to take a text based guess at what the image is, which is part the process of identifying the image and returning similar results. Search by Image looks for similar content on the web, so unique or never-before-seen images won't work well.

Shazam

There is a cool service called Shazam, which take a short sample of music, and identifies the song. There are couple ways to use it, but one of the more convenient is to install their free app onto an iPhone. Just hit the "tag now" button, hold the phone's mic up to a speaker, and it will usually identify the song and provide artist information, as well as a link to purchase the album.

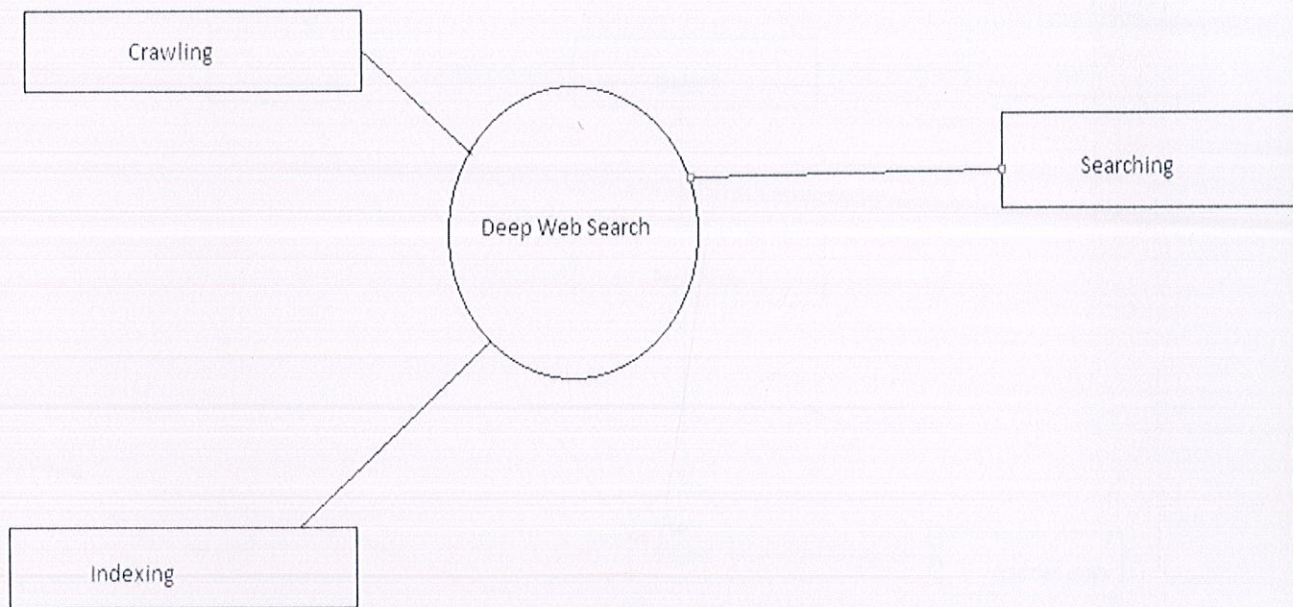
What is so remarkable about the service, is that it works on very obscure songs and will do so even with extraneous background noise.

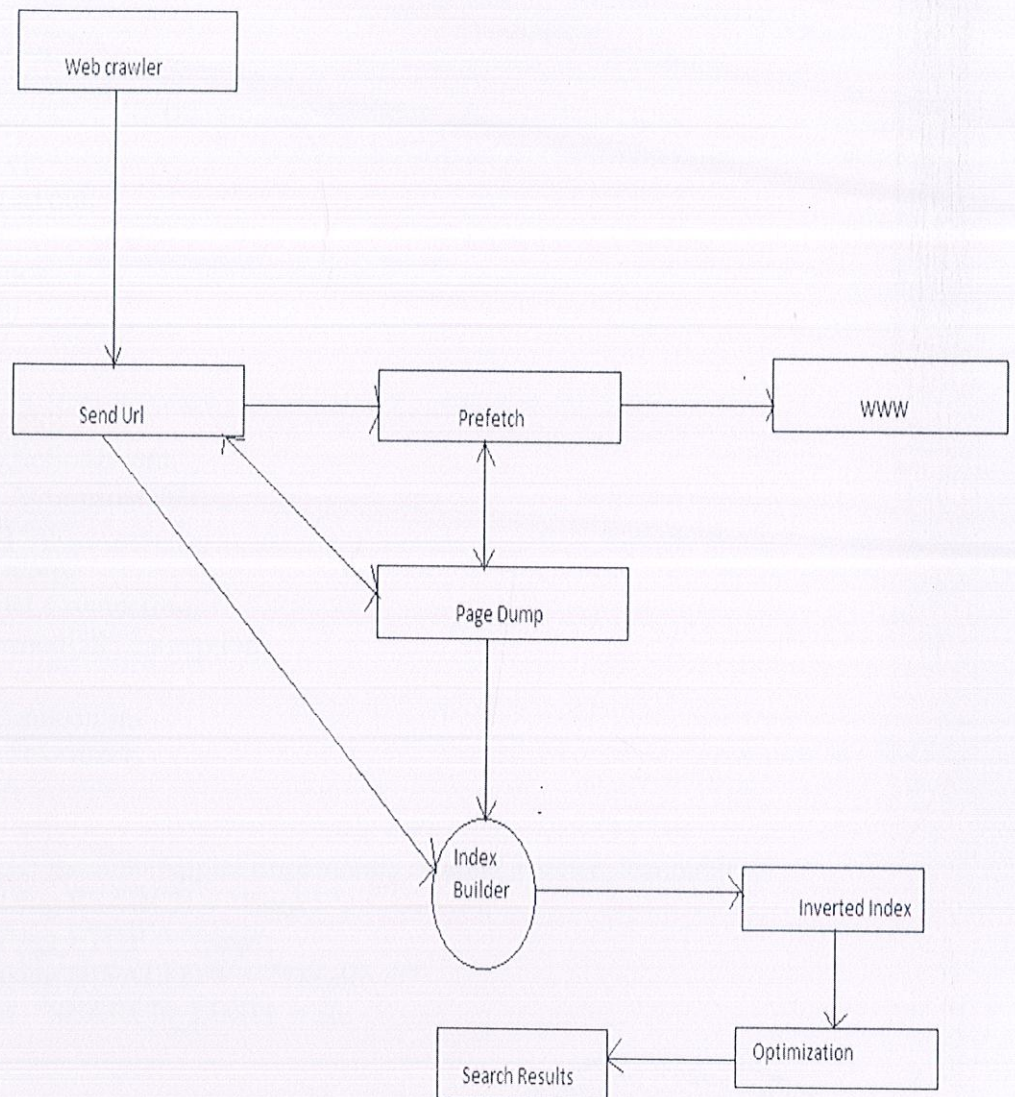
It relies on fingerprinting music based on the spectrogram.

Here are the basic steps:

1. Beforehand, Shazam fingerprints a comprehensive catalog of music, and stores the fingerprints in a database.
2. A user "tags" a song they hear, which fingerprints a 10 second sample of audio.
3. The Shazam app uploads the fingerprint to Shazam's service, which runs a search for a matching fingerprint in their database.
4. If a match is found, the song info is returned to the user, otherwise an error is returned.

Project Design





Sample Code (Basic Crawler) :

```
package webcrawler;

import java.applet.Applet;
import java.awt.BorderLayout;
import java.awt.Button;
import java.awt.Choice;
import java.awt.FlowLayout;
import java.awt.Frame;
import java.awt.Graphics;
import java.awt.Label;
import java.awt.List;
import java.awt.Panel;
import java.awt.TextField;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.IOException;
import java.io.InputStream;
import java.net.HttpURLConnection;
import java.net.MalformedURLException;
import java.net.URL;
import java.net.URLConnection;
import java.util.StringTokenizer;
import java.util.Vector;

public class WebCrawler extends Applet implements ActionListener, Runnable {
    public static final String SEARCH = "Search";
    public static final String STOP = "Stop";
    public static final String DISALLOW = "Disallow:";
    public static final int SEARCH_LIMIT = 50;

    Panel panelMain;
    List listMatches;
    Label labelStatus;

    // URLs to be searched
    Vector vectorToSearch;
    // URLs already searched
    Vector vectorSearched;
    // URLs which match
    Vector vectorMatches;

    Thread searchThread;

    TextField textURL;
```




```

Choice choiceType;

public void init() {

    // set up the main UI panel
    panelMain = new Panel();
    panelMain.setLayout(new BorderLayout(5, 5));

    // text entry components
    Panel panelEntry = new Panel();
    panelEntry.setLayout(new BorderLayout(5, 5));

    Panel panelURL = new Panel();
    panelURL.setLayout(new FlowLayout(FlowLayout.LEFT, 5, 5));
    Label labelURL = new Label("Starting URL: ", Label.RIGHT);
    panelURL.add(labelURL);
    textURL = new TextField("", 40);
    panelURL.add(textURL);
    panelEntry.add("North", panelURL);

    Panel panelType = new Panel();
    panelType.setLayout(new FlowLayout(FlowLayout.LEFT, 5, 5));
    Label labelType = new Label("Content type: ", Label.RIGHT);
    panelType.add(labelType);
    choiceType = new Choice();
    choiceType.addItem("text/html");
    choiceType.addItem("audio/basic");
    choiceType.addItem("audio/au");
    choiceType.addItem("audio/aiff");
    choiceType.addItem("audio/wav");
    choiceType.addItem("video/mpeg");
    choiceType.addItem("video/x-avi");
    panelType.add(choiceType);
    panelEntry.add("South", panelType);

    panelMain.add("North", panelEntry);

    // list of result URLs
    Panel panelListButtons = new Panel();
    panelListButtons.setLayout(new BorderLayout(5, 5));

    Panel panelList = new Panel();
    panelList.setLayout(new BorderLayout(5, 5));
    Label labelResults = new Label("Search results");
    panelList.add("North", labelResults);
    Panel panelListCurrent = new Panel();

```



```

panelListCurrent.setLayout(new BorderLayout(5, 5));
listMatches = new List(10);
panelListCurrent.add("North", listMatches);
labelStatus = new Label("");
panelListCurrent.add("South", labelStatus);
panelList.add("South", panelListCurrent);

panelListButtons.add("North", panelList);

// control buttons
Panel panelButtons = new Panel();
Button buttonSearch = new Button(SEARCH);
buttonSearch.addActionListener(this);
panelButtons.add(buttonSearch);
Button buttonStop = new Button(STOP);
buttonStop.addActionListener(this);
panelButtons.add(buttonStop);

panelListButtons.add("South", panelButtons);

panelMain.add("South", panelListButtons);

add(panelMain);
setVisible(true);

repaint();

// initialize search data structures
vectorToSearch = new Vector();
vectorSearched = new Vector();
vectorMatches = new Vector();

// set default for URL access
URLConnection.setDefaultAllowUserInteraction(false);
}

public void start() {

}

public void stop() {
    if (searchThread != null) {
        setStatus("stopping...");
        searchThread = null;
    }
}
}

```



```

public void destroy() {
}

boolean robotSafe(URL url) {
    String strHost = url.getHost();

    // form URL of the robots.txt file
    String strRobot = "http://" + strHost + "/robots.txt";
    URL urlRobot;
    try {
        urlRobot = new URL(strRobot);
    } catch (MalformedURLException e) {
        // something weird is happening, so don't trust it
        return false;
    }

    String strCommands;
    try {
        InputStream urlRobotStream = urlRobot.openStream();

        // read in entire file
        byte b[] = new byte[1000];
        int numRead = urlRobotStream.read(b);
        strCommands = new String(b, 0, numRead);
        while (numRead != -1) {
            if (Thread.currentThread() != searchThread)
                break;
            numRead = urlRobotStream.read(b);
            if (numRead != -1) {
                String newCommands = new String(b, 0, numRead);
                strCommands += newCommands;
            }
        }
        urlRobotStream.close();
    } catch (IOException e) {
        // if there is no robots.txt file, it is OK to search
        return true;
    }

    // assume that this robots.txt refers to us and
    // search for "Disallow:" commands.
    String strURL = url.getFile();
    int index = 0;
    while ((index = strCommands.indexOf(DISALLOW, index)) != -1) {

```



```

        index += DISALLOW.length();
        String strPath = strCommands.substring(index);
        StringTokenizer st = new StringTokenizer(strPath);

        if (!st.hasMoreTokens())
            break;

        String strBadPath = st.nextToken();

        // if the URL starts with a disallowed path, it is not safe
        if (strURL.indexOf(strBadPath) == 0)
            return false;
    }

    return true;
}

public void paint(Graphics g) {
    //Draw a Rectangle around the applet's display area.
    g.drawRect(0, 0, getSize().width - 1, getSize().height - 1);

    panelMain.paint(g);
    panelMain.paintComponents(g);
    // update(g);
    // panelMain.update(g);
}

public void run() {
    String strURL = textURL.getText();
    String strTargetType = choiceType.getSelectedItem();
    int numberSearched = 0;
    int numberFound = 0;

    if (strURL.length() == 0) {
        setStatus("ERROR: must enter a starting URL");
        return;
    }

    // initialize search data structures
    vectorToSearch.removeAllElements();
    vectorSearched.removeAllElements();
    vectorMatches.removeAllElements();
    listMatches.removeAll();

    vectorToSearch.addElement(strURL);

```



```

while ((vectorToSearch.size() > 0)
    && (Thread.currentThread() == searchThread)) {
    // get the first element from the to be searched list
    strURL = (String) vectorToSearch.elementAt(0);

    setStatus("searching " + strURL);

    URL url;
    try {
        url = new URL(strURL);
    } catch (MalformedURLException e) {
        setStatus("ERROR: invalid URL " + strURL);
        break;
    }

    // mark the URL as searched (we want this one way or the other)
    vectorToSearch.removeElementAt(0);
    vectorSearched.addElement(strURL);

    // can only search http: protocol URLs
    if (url.getProtocol().compareTo("http") != 0)
        break;

    // test to make sure it is before searching
    if (!robotSafe(url))
        break;

    try {
        // try opening the URL
        URLConnection urlConnection = url.openConnection();

        urlConnection.setAllowUserInteraction(false);

        InputStream urlStream = url.openStream();
        String type
            = urlConnection.getContentType();

        if (type == null)
            break;
        if (type.compareTo("text/html") != 0)
            break;

        // search the input stream for links
        // first, read in the entire URL
        byte b[] = new byte[1000];
        int numRead = urlStream.read(b);

```



```

String content = new String(b, 0, numRead);
while (numRead != -1) {
    if (Thread.currentThread() != searchThread)
        break;
    numRead = urlStream.read(b);
    if (numRead != -1) {
        String newContent = new String(b, 0, numRead);
        content += newContent;
    }
}
urlStream.close();

//textURL.setText("1. " + content);
if (Thread.currentThread() != searchThread)
    break;

String lowerCaseContent = content.toLowerCase();

int index = 0;
boolean hasForm = false;
while ((index = lowerCaseContent.indexOf("<a", index)) != -1)
{
    if ((index = lowerCaseContent.indexOf("href", index)) == -1)
        break;
    if ((index = lowerCaseContent.indexOf("=", index)) == -1)
        break;

    if (lowerCaseContent.indexOf("<form") >= 0){
        hasForm = true;
    }
    if (Thread.currentThread() != searchThread)
        break;

    index++;
    String remaining = content.substring(index);

    StringTokenizer st
        = new StringTokenizer(remaining, "\t\n\r">#");
    String strLink = st.nextToken();

    URL urlLink;
    try {
        urlLink = new URL(url, strLink);
        strLink = urlLink.toString();
    } catch (MalformedURLException e) {

```



```

        setStatus("ERROR: bad URL " + strLink);
        continue;
    }

    // only look at http links
    if (urlLink.getProtocol().compareTo("http") != 0)
        break;

    if (Thread.currentThread() != searchThread)
        break;

    try {
        // try opening the URL
        URLConnection urlLinkConnection
            = urlLink.openConnection();
        urlLinkConnection.setAllowUserInteraction(false);
        InputStream linkStream = urlLink.openStream();
        String strType
            = urlLinkConnection.getContentType();
        int responseCode = 200;
        HttpURLConnection httpConnection;
        if (urlLinkConnection instanceof HttpURLConnection)
        {
            httpConnection = (HttpURLConnection) urlLinkConnection;
            responseCode = httpConnection.getResponseCode();
            //if(strType.indexOf("302 Authentication Failed") >= 0)
            //    responseCode = 302;
        }
        linkStream.close();

        // if another page, add to the end of search list
        if (strType == null || responseCode != 200)
            break;
        if (strType.compareTo("text/html") == 0) {
            // check to see if this URL has already been
            // searched or is going to be searched
            if ((!vectorSearched.contains(strLink))
                && (!vectorToSearch.contains(strLink))) {

                // test to make sure it is robot-safe!
                if (robotSafe(urlLink))
                    vectorToSearch.addElement(strLink);
            }
        }

        // if the proper type, add it to the results list
    }

```



```

        // unless we have already seen it
        if (strType.compareTo(strTargetType) == 0) {
            if (vectorMatches.contains(strLink) == false) {
                listMatches.add(strLink);
                vectorMatches.addElement(strLink);
                numberFound++;
                if (numberFound >= SEARCH_LIMIT)
                    break;
            }
        }
    }
    catch (IOException e) {
        setStatus("ERROR: couldn't open URL " + strLink);
        continue;
    }
}
catch (IOException e) {
    setStatus("ERROR: couldn't open URL " + strURL);
    break;
}

numberSearched++;
if (numberSearched >= SEARCH_LIMIT)
    break;
}

if (numberSearched >= SEARCH_LIMIT || numberFound >= SEARCH_LIMIT)
    setStatus("reached search limit of " + SEARCH_LIMIT);
else
    setStatus("done");
searchThread = null;
// searchThread.stop();
}

void setStatus(String status) {
    labelStatus.setText(status);
}

public void actionPerformed(ActionEvent event) {
    String command = event.getActionCommand();

    if (command.compareTo(SEARCH) == 0) {
        setStatus("searching...");

        // launch a thread to do the search
    }
}

```



```

        if (searchThread == null) {
            searchThread = new Thread(this);
        }
        searchThread.start();
    }
    else if (command.compareTo(STOP) == 0) {
        stop();
    }
}
}
public static void main (String argv[])
{
    Frame f = new Frame("Web Crawler");
    WebCrawler applet = new WebCrawler();
    f.add("Center", applet);

    /* Behind a firewall set your proxy and port here! */

    Properties props= new Properties(System.getProperties());
    props.put("http.proxySet", "true");
    props.put("http.proxyHost", "webcache-cup");
    props.put("http.proxyPort", "8080");

    Properties newprops = new Properties(props);
    System.setProperties(newprops);

    applet.init();
    applet.start();
    f.pack();
    f.show();
}
}

```


References:

- http://en.wikipedia.org/wiki/Surface_Web
- <http://worldwidescience.org/speeches/Oct2008/alternate.html>
- <http://cathryno.global2.vic.edu.au/2010/05/08/deep-web-vs-surface-web/>
- <http://windowssecrets.com/langalist-plus/surface-vs-deep-web%C2%A0/>
- http://en.wikipedia.org/wiki/Deep_Web
- “The Invisible Web”, Uncovering Information Search Engines can’t find, by ‘Chris Sherman’ & ‘Gary Price’.