CHAPTER – 2
LITERATURE SURVEY

<table>
<thead>
<tr>
<th>S. No</th>
<th>Contents</th>
<th>Page No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Web Search Engine</td>
<td>33</td>
</tr>
<tr>
<td>2.2</td>
<td>Hidden Web Search Engine</td>
<td>41</td>
</tr>
<tr>
<td>2.3</td>
<td>Hidden Web Crawler</td>
<td>45</td>
</tr>
<tr>
<td></td>
<td>Bibliography</td>
<td>53</td>
</tr>
</tbody>
</table>
CHAPTER - 2

LITERATURE SURVEY

An ever-increasing amount of information on the Web today cannot be reached by following links as in [17, 18]. In particular, a large part of the Web is “hidden” behind search forms, and is reachable only when users type in a set of keywords, or queries, to the forms. These pages are often referred to as the Hidden Web as in [19] or the Deep Web as in [17], because search engines typically cannot index the pages and do not return them in their results (thus, the pages are essentially “hidden” from a typical Web user).

According to many studies, the size of the Hidden Web increases rapidly as more organizations make their valuable content accessible online through an easy-to-use Web interface as in [17]. In as in [18], Chang et al. estimate that well over 100,000 Hidden-Web sites currently exist on the Web. Moreover, the content provided by many Hidden-Web sites is often of very high quality and can be extremely valuable to many users as in [17]. For example, Pub Med as in [20] hosts many high-quality papers on medical research that were selected from careful peer-review processes, while the site of the US Patent and Trademarks Office as in [21] makes existing patent documents available, helping potential inventors examine “prior art.” In this dissertation, the study how to build a search engine for the Hidden Web i.e., how effectively collect the data from the Hidden Web and enable the users to search for information within the collected data.

Information retrieval (IR) deals with the representation, storage, organization of, and access to information items. The overall goal of an IR system can be stated as
to provide items that are relevant to a user’s information need. In the context of text retrieval, which is the focus of this thesis, information items typically correspond to unstructured or semi-structured documents, while information needs are represented as natural language queries. The key challenge faced by an IR system is to determine the relevance of a document given a user’s query. Since relevance is a prerogative of the user, the IR system can at best estimate it. This task is further aggravated by the fact that both queries and documents are semantically ambiguous expressions of information in natural language. Such an inherent ambiguity precludes a precise match between information needs and items, as would be the case in a data retrieval system, such as a relational database. In order to be able to effectively answer a user’s query, an IR system must be able to first understand the information need underlying this query. In turn, this information need may convey distinct user intents, from a general search for information about a topic, to a search for a particular website.

2.1 Web Search Engine

Web search engines are arguably the most popular instantiation of an IR system. A recent report revealed that at least 100 billion searches are conducted on the leading commercial web search engine each month, amounting to over 3.3 billion searches each day. Besides understanding the information needs of such a mass of users with varying interests and backgrounds, web search engines must also strive to understand the information available on the Web. In particular, the decentralized nature of content publishing on the Web has led to the formation of an unprecedentedly large repository of information, comprising over 30 trillion uniquely addressable documents. While the lack of a central control is key for the democratization of the Web, it also results in a substantial heterogeneity of the
produced content, from its language and writing style, to its authoritativeness and trustworthiness. Another distinctive characteristic of the Web compared to traditional information repositories is its interconnected nature. Indeed, not only do web authors publish massive amounts of information, but they also create links (also known as hyperlinks) between the published information. As a result, the Web can be viewed as a directed graph, with documents represented as nodes, and hyperlinks between documents represented as directed edges. Understanding the web graph is crucial for understanding the structure and dynamics of the Web itself, but it also plays a fundamental role in designing effective and efficient web search engines. The massive-scale, heterogeneous, and interconnected nature of the Web makes it a particularly challenging environment for search. To cope with this challenge, web search engines are typically designed with three core components: crawler, indexer, and query processor.

2.1.1 Crawling

Crawling is the process by which search engines collect documents from the Web into a local corpus. Such a corpus can be then processed by the search engine in order to allow users to efficiently locate information. The overall goal of crawling is to build a corpus as comprehensive as possible, in as little time as possible. To this end, a web crawler must maximize its crawling rate, while making efficient use of its own resources, as well as the resources of the servers that host the desired documents. Crawling the Web can be seen as a graph traversal problem. As shown in Figure below, at all times, the crawler maintains a list of URLs to be visited, the so-called crawling frontier, which is initially filled with a few seed URLs. While the frontier is not empty, the next URL to be visited is removed from it and downloaded by a fetcher
module, after a DNS resolver translates the URL domain into an IP address. The fetched document is processed by the crawl controller and the extracted contents are stored locally for indexing. The URLs extracted from this document and the document’s own URL, for continuous crawls are inserted back into the frontier, so that they can be visited by the crawler at a later time.

Figure 2.1: Schematic view of a crawler

While new documents are created and existing ones are modified at a massive scale, the resources available for crawling—notably, storage and bandwidth—are limited. To make crawling scalable, web crawlers must consider carefully which URLs to visit, and how often to revisit each URL. The decision of which URLs to visit depends on the predicted usefulness of each URL regardless of any particular query. Such a decision could be based on the global importance of the document referred to by the URL or its perceived quality. However, in practice, it has been shown that a simple breadth first search is an effective traversal strategy, as it identifies important pages early in the crawling process. The decision of how often to revisit a particular URL can be even more involved. With the dynamic nature of the Web, by the time a web crawler has finished crawling its frontier, many events could
have happened. These events can include the creation, update, or deletion of documents. Moreover, different documents evolve at different rates. For instance, documents related to news, sports, and personal pages tend to change more frequently than those hosted in educational or governmental domains. At the extreme, recent years have witnessed the emergence of social media, which encourage real-time publishing on collaborative projects, blogs, micro blogs, social networking sites, and virtual game worlds. To provide access to the wealth of information on the Web, a crawler must be able to adapt itself to the publishing patterns of such heterogeneous outlets, e.g., by crawling more often those pages that change more often.

2.1.2 Indexing

The overall goal of indexing is to create a representation of the documents in the local corpus suitable for automatic processing by a search engine. The devised document representations are then stored in appropriate data structures for efficient access by the query processor. Given a corpus of documents (e.g., crawled from the Web), each document is indexed following the general process illustrated in Figure below. Initially, a parser extracts the textual content from each document. The extracted content is then processed by a tokenizer, which splits the raw text into individual tokens. An analyzer performs multiple text operations on individual tokens and records their occurrences in each document. In this process, two main data structures are created, which are at the core of modern indexing architectures. The first of these is a lexicon, which stores information for all unique terms in the corpus, such as their total number of occurrences and the number of documents where they occur. The second structure is an inverted file, which stores, for each term in the lexicon, a posting list, comprising information on the occurrence of the term in
different documents, such as the frequency of the term in each document. To enable efficient storage and retrieval, both structures are typically compressed. Indexing may be performed in a single batch, in which case the whole corpus must be re-indexed when there is an update, or incrementally, through small atomic operations. Parsing web documents can be a complex task. With the global and democratic nature of the Web, web documents can have a variety of content types and character encodings, which may not be immediately identifiable from the document itself (in an HTML header) or from its provider (in an HTTP response header). Even pure textual content may contain noise. Indeed, web documents typically comprise irrelevant content besides their core topic, such as advertisements, client-side scripting code, and frequently a whole HTML template structure. Such a noisy content can hurt not only the effectiveness of a search engine, but also its efficiency, since more content needs to be stored and processed. In order to remove noise and extract cleaner content for indexing, “boilerplate removal” algorithms can be applied. Tokenization is a relatively trivial task for most western languages, in which tokens can be separated by a whitespace or a punctuation character. On the other hand, languages such as German do not separate compound words. In the extreme, East Asian languages such as Chinese, Japanese, and Korean have no word boundaries at all. A similar problem, common to all languages, is the segmentation.
Figure 2.2: Schematic view of query Indexer

Not all identified tokens are directly useful for search. For this reason, each token can be analyzed and transformed through a series of text operations before being indexed. For instance, a search engine can choose not to index too common terms. Such terms, known as stop words, possess little discriminative power for deciding which documents should be retrieved in response to a query. In addition, their presence can also impact efficiency, since their posting lists can be almost as long as the number of documents in the corpus. Besides gap removal, another common text operation is stemming, a process that reduces multiple words to their common grammatical root, so as to increase the chance of retrieving documents that contain a different variant of the query terms. For instance, after stemming, the terms “retrieval”, “retriever”, and “retrieving” can all be reduced to their common root, “retrieve”. Alternatively, the search engine may choose to index all the identified tokens in their original form, in which case text operations are delayed until the query processing stage. This choice is more flexible, as it allows for text operations to be deployed only when they are predicted to be helpful. Different information about terms, documents, and the occurrence of terms in documents can be indexed. The most basic information, which is one of the pillars for query-dependent ranking, is the frequency of a term in a document. Recording the position where each term occurs in each document can also help improve the effectiveness of a search engine. For instance, the terms “information” and “retrieval” appearing next to each other can be a strong indicator of the relevance of a document for the query “information retrieval”. In addition, term frequency and positional information can be recorded for different fields of a document, such as its title, URL, or body. Another valuable source of
evidence, which conveys how a document is described by the rest of the Web, is the anchor text of the incoming hyperlinks to this document. Finally, several other features that can help infer the prior relevance of a document regardless of any query can be computed and stored at indexing time.

2.1.3 Query Processing

Query processing is the component responsible for answering users’ queries. When a user poses a query, the search engine examines its index structures to locate the most relevant documents for this query. Given the size of the Web and the short length of typical web search queries, there may be billions of matching documents for a single query.

![Figure 2.3: Schematic view of query processor](image)

In order to be effective, a search engine must be able to rank the returned documents, so that the most relevant documents are presented ahead of less relevant ones. Query processing consists of three basic operations. Initially, the search engine receives a query, as a typically short and often underspecified representation of the user’s information need. This query may go through a series of query understanding operations, aimed to overcome the gap between the user’s information need and the ill-defined representation of this need in the form of a query. This stage is important, since misinterpreting the user’s information need implies that relevant documents may
never be returned, regardless of how sophisticated the subsequent retrieval is. Once a suitable representation of the user’s query has been created, a matching process retrieves the indexed documents that contain the query terms. Lastly, to ensure that the user is presented with the most likely relevant documents for the query, the retrieved documents are scored and sorted by a ranking process. Query understanding aims at deriving a representation of the user’s query that is better suited for a search engine. Typical query understanding operations include refinements of the original query, such as spelling correction, acronym expansion, stemming, term deletion, query segmentation, and named entity recognition. Other common query understanding operations are query topic classification, aimed at restricting the scope of the retrieved documents, and query expansion, to enhance the query representation with useful terms from the local corpus or from external resources, such as a query log or a knowledge base such as Wikipedia. Users typically expect instant responses from a web search engine. This makes it inefficient to fully score all documents matching the query terms. Hence, scoring is typically performed as a multi-layer process. In the first layer, matching documents from the entire corpus are returned as an disarrayed set using a standard Boolean retrieval approach. The second layer deploys an unsupervised query-dependent ranking approach, in order to provide an overall ordering of the initially matched documents at a low cost. This cost can be made even lower by deploying efficient matching techniques, so as to short circuit the examination of the posting lists of documents that will not make the final ranked list. Finally, in the third layer, machine-learned ranking can be deployed to integrate ranking evidence from multiple features.
2.2 Hidden Web Search Engine

2.2.1 Benefits

A Hidden-Web search engine can bring tremendous benefits to the users including the following:

i. **Taking advantage of unexplored informative content:** Our Hidden-web crawler will permit a normal Web client to effectively investigate the incomprehensible measure of informative data that is chiefly "concealed" at present. Since a larger part of Web clients depend on internet searchers to run across pages, when pages are not ordered via internet searchers, they are unrealistic to be seen by numerous Web clients. Unless clients head off straight to Hidden-Websites and issue inquiries there, they can't access the pages at the destinations.

ii. **Enhancing client experience:** Even if a client is conscious of various Hidden-Web locales, the client as of now needs to waste a huge measure of time and enterprise, going by the sum of the conceivably important locales, questioning each of them and investigating the effect, by making the Hidden-Web pages searchable at a midway area. Also additionally to essentially lessen the client's squandered time and venture in looking the Hidden Web.

iii. **Diminishing potential inclination:** Due to the substantial dependence of numerous Web clients on web indexes for finding informative data, web crawlers impact how the clients recognize the Web as in [22]. Clients don't fundamentally recognize what really exists on the Web, yet what is ordered
via web crawlers as in [22]. Consistent with a later article as in [23], a few conglomerations have distinguished the vitality of carrying qualified data of their Hidden-Web destinations onto the surface, and dedicated respectable assets towards this venture. By downloading the Hidden-Web informative content and making it approachable to evacuate some of this inclination.

The Deep Web comprises information that exists on the Web however are unavailable by content web search tools through universal creeping and indexing as in [24]. The most prevalent path to enter this information is to physically load up Html structures on inquiry interfaces. This methodology is not versatile given the overpowering size of Deep Web as in [25]. Programmed access to this information has assembled much consideration recently. This requires programmed comprehension of inquiry interfaces. The Deep Web is described by its developing scale, dominion differing qualities, and various organized databases as in [26]. It is developing at this quick pace, to the point that successfully evaluating its size is a troublesome issue as in [26, 27, 28, 29]. In the past, scientists have proposed numerous results for make the Deep Web information more convenient to clients. Ru and Horowitz as in [30] order these results into 2 classes: dynamic substance archive, and ongoing inquiry provisions.

There is an amplification to this arrangement plan by determining 3 objective based classes: (i) results for expansion the substance per cleavability on content internet searchers, for example collecting/indexing dynamic pages as in [31, 32] and making store of dynamic page substance as in [33,34, 35]; (ii) results for expansion the intra-area inquiry capability, for example meta-web indexes as in [36, 37, 38, 39, 40, 41]; (iii) results for perform learning conglomereration, for example philosophy
Programmed comprehension of Deep Web seek interfaces is the essential to achieve any of the previously stated results. Profound Web holds no less than 10 million high caliber interfaces as in [43] and programmed recovery of such interfaces has likewise accepted extraordinary consideration as in [44]. Seek interface comprehension is the methodology of concentrating semantic qualified information from an interface. A pursuit interface holds a grouping of interface segments, i.e., content marks and structure components (textbox, choice record, and so on.). An interface is fundamentally configured for human comprehension and questioning. It doesn't have a standard layout of parts and there is unbounded number of conceivable layout designs as in [45]. In this way, while human clients effortlessly observe an interface dependent upon past encounters and surface signals, machine preparing of an interface is testing.

2.2.2 Challenges

Given the enormous size of the Hidden Web and the fact that the information is only accessible through a search form, building a search engine for the Hidden Web imposes many important challenges, including the following:

2.2.2.1. Crawling the Deep Web

Robots are programs that cross the Web robotically and download pages for explore engines. Traditional robots today lay mainly on top of the hyper-links on top of the Web to find out and download documents. Due to the lack of links pointing to the Deep Web documents, the current look for engines might not directory the Hidden-Web documents.
2.2.2.2. Updating the Hidden-Web pages

The data on the Web nowadays is highly volatile. New and improved content is constantly added to serve the users’ needs in a better way. At the same time, the potentially useful content is removed from the Web at an administrator’s whim. Once our crawler has downloaded the information from the Hidden-Web, it needs to periodically refresh its local copy in order to enable users to search for up-to-date information. However, because pages change independently and at different rates, it is very significant for the agents to select cautiously which documents to refresh: if the crawler downloads pages that have not changed, then it is simply wasting time and bandwidth instead of updating our local copy. Therefore, one challenge that the crawler has to face is being able to identify which pages on the Hidden-Web have changed and download them in a timely manner.

2.2.2.3. Indexing and searching the Hidden Web

Once there are downloaded the Hidden-Web pages, it enables the users to search for useful information. Search engines typically do this by maintaining important overturned indexes which are replicated dozens of era for scalability. Given the enormous size of available information on the Hidden Web, this task can become very costly. One way to reduce the cost is to replace some of the full inverted indexes with smaller, pruned inverted indexes as in [46, 47], where the most frequently accessed portions of the index are stored. While this approach can provide major development in presentation, it leads to conspicuous deprivation in the excellence of
the find outcome, when the peak answers are computed only from the trimmed directory as in [46, 47].

Since our goal is to provide the users with Hidden-Web results of the highest quality, this degradation is clearly undesirable. Therefore, the challenge to address is how to keep away from a few deprivation of outcome excellence owing to the pruning-based presentation optimization, while still understanding its benefit.

Of late some locales on the Web watch a regularly expanding divide of their movement originating from web index referrals. For numerous business Web locales, an expansion in web search tool referrals interprets to an expansion in bargains, income, and benefits.

Some Web webpage administrators attempt to impact the positioning of their pages inside list items by crafting “bogus” or spam Web pages which aim solely at manipulating search engines and are useless to any human user. In the case of the Hidden-Web, these malicious Web site operators may try to pollute our index by injecting spam content in their Hidden-Web databases so that our crawler can download it. Therefore, one challenge that to address is to create an effective Web spam detection method, in order to ensure the quality of the data that returns to users.

2.3. Hidden Webcrawler

Crawlers are systems that consequently cross the Web diagram, recovering pages and raising a nearby storehouse of the share of the Web that they visit. Hinging on the provision close by, the pages in the vault are either used to raise seek records, or are subjected to different types of investigation (e.g., content mining). Generally,
crawlers have just focused on a part of the Web called the freely file capable Web (Piw) as in [48]. This implies the set of pages reachable perfectly by accompanying hypertext connections, overlooking pursuit structures and pages that require commission or former enlistment.

In any case, various later studies as in [49, 48, 50] have watched that a noteworthy portion of Web substance indeed untruths outside the Piw. Particularly, extensive partitions of the Web are "stowed away" behind pursuit shapes, in searchable organized and unstructured databases (called the concealed Web as in [51] or profound Web as in [49]). Pages in the concealed Web are powerfully created according to questions submitted through the inquiry shapes. The concealed Web presses on to develop, as conglomerations with expansive measures of astounding informative data (e.g., the Census Bureau, Patents and Trademarks Office, news media associations) are putting their substance web, furnishing Web-receptive pursuit offices over existing databases. For example, the site Invisibleweb.com records over 10000 such databases extending from chronicles of work postings to catalogs, news documents, and electronic inventories. Later gauges as in [49] put the measure of the shrouded Web as far as produced Html pages) at around 500 times the extent of the Piw.

Note that creeping dynamic pages from a database comes to be fundamentally simpler if the site hosting the database is agreeable. For example, a crawler may be utilized by a conglomeration to assemble and file pages and databases on its neighborhood intranet. Thus, the web servers running on the inside system might be arranged to distinguish asks for from the crawler and accordingly, fare the whole
database in some predefined configuration. This methodology is as of recently utilized by some e-business destinations, which distinguish asks for from the crawlers of major web search tool associations and accordingly, fare their whole catalog/database for indexing.

Later studies indicate that a huge portion of Web substance can't be arrived at by taking after connections as in [17, 18]. Specifically, an imposing part of the Web is "covered up" behind pursuit structures and is reachable just when clients sort in a set of essential words, or inquiries, to the shapes. These pages are frequently implied as the Hidden Web as in [52] or the Deep Web as in [17], in light of the fact that web search tools regularly can't list the pages and don't return them in their outcomes (consequently, the pages are basically "stowed away" from an ordinary Web client). Consistent with numerous examines, the measure of the Hidden Web builds quickly as additional conglomerations put their profitable substance online through a simple to utilize Web interface as within [17]. As in [18], Chang et al. appraise that well over 100,000 Hidden-Web locales presently exist on the Web. In addition, the substance furnished by numerous Hidden-Web locales is regularly of exceptionally amazing and could be amazingly profitable to numerous clients as in [17]. Case in point, Pub Med hosts numerous high caliber papers on therapeutic research that were chosen from cautious associate audit techniques, while the webpage of the Us Patent and Trademarks Office 1 makes existing patent reports accessible, assisting potential creators analyze "former art" an adequate Hidden-Web crawler can have a gigantic effect on how clients seek qualified information on the Web:
The Hidden-Web crawler endeavors to computerize this procedure for Hidden Web destinations with printed substance, accordingly minimizing the partnered expenses and exertion needed. There are two center tests to achieving a successful Hidden-Web crawler: (a) The crawler must have the ability to grasp and model a question interface, and (b) The crawler needs to think of serious questions to issue to the question interface as in [53], where a technique for studying inquiry interfaces was displayed.

Here, there is a result for the second test, i.e. how a crawler can consequently produce inquiries with the intention that it can find and download the Hidden-Web pages. Decidedly, when the inquiry shapes record all conceivable qualities for a question, the result is straightforward. Exhaustively issue all conceivable inquiries, one question at once. The point when the question shapes have a "free content" data, then again, an interminable number of questions are conceivable, so it can't exhaustively issue all conceivable inquiries. Hence, what questions would it be advisable for us to pick? In what manner can the crawler consequently concoct dynamite questions without comprehension the semantics of the inquiry shape.

2.3.1. A general Deep Web crawling algorithm:

Given that the main "section" to the pages in a Hidden-Web webpage is its inquiry from, a Hidden-Web crawler might as well accompany the three steps portrayed in the past area. That is, the crawler needs to produce an inquiry, issue it to the Web website, download the outcome record page, and accompany the connections to download the real pages. As a rule, a crawler has restricted time and system assets, so the crawler rehashes these steps until it employments.
In above figure 2.4 the non specific calculation for a Hidden-Web crawler. For straightforwardness, accept that the Hidden-Web crawler issues single-term questions just. The crawler first chooses which inquiry term it is set to utilize (Step (2)), issues the question, and recovers the outcome record page (Step (3)). At long last, taking into account the connections discovered on the effect file page, it downloads the Hidden-Web pages from the webpage (Step (4)). This same methodology is rehashed until all the ready assets are utilized up (Step (1)).

Given this calculation that is the most basic choice that a crawler needs to make is the thing that inquiry to issue afterward. Provided that the crawler can issue efficacious inquiries that will give back numerous matching pages, the crawler can

**Algorithm:**

(1) while ( there are available resources ) do
   // select a term to send to the site
   (2) q= Select Term()
   // send query and acquire result index page
   (3) R(qi) = QueryWebsite( qi )
   // download the pages of interest
   (4) Download( R(qi) )
(5) done

Figure 2.4: Crawling hidden web procedure
fulfill its slithering at an opportune time utilizing least assets. Interestingly, if the crawler issues totally incidental questions that don't give back any matching pages, it might squander every last bit of its assets essentially issuing inquiries while never recovering real pages. Accordingly, how the crawler chooses the following question can incredibly influence its viability.

2.3.2. Trouble formalization:

Figure 2.5: A set-formalization of the optimal query selection problem.

Hypothetically, the issue of inquiry determination could be formalized as accompanies: expect that the crawler downloads pages from a Web webpage that has a set of pages $S$ (the rectangle in Figure 2.5). stand for every Web page in $S$ as a focus (dabs in Figure 2.5). every potential inquiry $q_i$ that issue could be seen as a subset of $S$, holding all the focuses (pages) that are returned when the issue $q_i$ to the webpage.

Every subset is connected with a weight that speaks for the expense of issuing the inquiry. Under this formalization, the objective is to find which subsets (inquiries) spread the most extreme number of focuses (Web pages) with the base aggregate weight (expense).
This issue is comparable to the situated blanket issue in diagram speculation as in [18]. There are two primary challenges that to address in this formalization. First and foremost, in a down-to-earth scenario, the crawler does not know which Web pages will be returned by which inquiries, so the subsets of S are not known ahead of time. Without knowing these subsets the crawler can't choose which inquiries to pick to boost the scope. Second, the set-blanket issue is known to be NP-Hard as in [18], so a proficient calculation to take care of this issue optimally in polynomial time has yet to be discovered.

Figure 2.6: Flow of control in a general WebCrawler

2.3.3. Features of a WebCrawler

All Web Crawlers should satisfy some predefined features. The main features of a WebCrawler are as in [54]:

i. **Robustness**: It may happen that a Web server creates spider traps and generates web pages that may mislead crawler to get stuck while fetching
unlimited figure of documents in a particular area. So crawlers designed should be flexible enough to handle such traps.

ii. **Politeness:** Web servers have their own limitations in terms of hidden and open policies for regulating the crawler visit rate. So WebCrawler should have high regard for these politeness policies.

iii. **Distributed:** It is the requirement of time that a WebCrawler should have the capability to work in a distributed environment crossways many devices.

iv. **Scalable:** The crawler architecture should provide support to expand the crawl rate by adding extra machines and bandwidth in the existing architecture.

v. **Quality:** As it is not possible to index the whole WWW pages and as also a large fraction of web pages are useless for end users, so the crawler should be biased enough to crawl relevant pages first.

vi. **Performance and efficiency:** The crawler should utilize the available resources such as processor, storage and network bandwidth in the best manner possible.

vii. **Freshness:** As large fraction of web pages are dynamic in nature, so the crawler should adopt some policy to revisit the crawled pages repeatedly in incremental or periodic fashion to keep the repository up-to-date.

viii. **Extensible:** Crawler should be designed in such a way so that it may come up with changes like new data format, new fetch protocol and so on. This may be satisfied if crawler architecture is designed in a modular approach. Though optimum crawlers are those which support all the above mentioned features but few features are a must such as robustness, politeness etc and a few are desirable like extensible etc.