Efficient Social Website Crawling Using Cluster Graph

Yifei Jiang
yifei.jiang@colorado.edu

Qin Lv
qin.lv@colorado.edu

Department of Computer Science
University of Colorado at Boulder

Technical Report CU-CS-1056-09

November 2009

Efficient Social Website Crawling Using Cluster Graph

Yifei Jiang
Department of Computer Science
Univ. of Colorado at Boulder
yifei.jiang@colorado.edu

Qin Lv
Department of Computer Science
Univ. of Colorado at Boulder
qin.lv@colorado.edu

ABSTRACT

Online social communities have gained significant popularity in recent years and have become an area of active research. Compared with general websites or well-structured Web forums, user-centered social websites pose several unique challenges for crawling, a fundamental task for data collection and data mining of large-scale online social communities: (1) Social websites have more complex link structures and much higher indegree and outdegree, resulting in a large number of duplicate links; (2) Social websites contain large amounts of duplicate content usually listed under different URLs; (3) Social websites are interactive in nature, containing a large number of action or uninformative webpages such as login, tell-a-friend, or commenting; and (4) Social webpages differ dramatically in URL format, link structure, and page layout, due to their diverse semantics, functionalities, and user customization. Previous crawler designs targeting the general Web or well-structured Web forums are inadequate for social websites, wasting network bandwidth, storage space, and causing extra overload in social network analysis and data mining tasks.

This work tackles the problem of efficient social website crawling by proposing two key techniques: (1) URL-based webpage clustering that identifies frequent itemsets in URLs and groups webpages into semantic clusters; and (2) cluster graph pruning that removes edges and nodes representing duplicate links, duplicate or uninformative content. The offline trained webpage cluster graph is then used at runtime to direct the crawling process. By using only URLs and page link structures, our cluster-graph-based approach can successfully address the challenges in crawling social websites. Extensive evaluations on three different social websites demonstrate that our approach can effectively and efficiently crawl large amounts of informative social content while dramatically reducing the number of duplicate links as well as the amount of duplicate or uninformative content.

1. INTRODUCTION

Recently, online social communities have gained significant popularity and are now among the most popular sites in the Web. Over two thirds of the global online population visit a social network or blogging site, and the sector now accounts for almost 10% of all Internet time [32]. Facebook, one of the world's most popular social networks, is visited monthly by three in every 10 people online. Another popular social network, MySpace, has over 253 million users and a traffic rank of 9 [31]. The popularity of these sites provides great opportunities for the research community, spanning from structural analysis of social networks [30], to social network-based information dissemination [3], and to the study of social behavior [36].

Research on online social communities requires a fundamental step – obtaining data from social websites. While a large number of social websites exist, few of them provide readily-available data sets. One possible approach is to use the APIs provided by individual websites (e.g., the Flickr API [19]). Although the APIs provide a convenient way for obtaining semantically-structured data, their functionalities are usually limited and do not provide all the information needed for effective social community analysis. In addition, the fact that only a small number of social websites provide APIs and that APIs vary from site to site has made it difficult to apply this approach to a large number of diverse social websites. Instead, web crawlers that rely on HTML screen scraping are needed to obtain large amounts of data from diverse social websites.

Web crawling has been an area of active research for over a decade, and many crawlers have been developed [8, 12, 14, 25, 40, 27]. Compared with general websites and well-structured Web forums, social websites pose several unique challenges for Web crawling:

Duplicate Links: Online social communities support many types of social interactions among a huge number of users. As a result, a social website has complex link structures and its webpages are tightly intertwined with each other. For example, a user's profile page may link to all his/her friends, communities, interest tags, blog entries, and vice versa. Such high indegree and outdegree of social webpages would generate a huge number of duplicate links in the crawling process.

Duplicate Content: Online social websites usually support powerful and convenient user interfaces for browsing and social interaction. The same content may be displayed in multiple ways (e.g., user posts by name, by date, or by topic), and the same content may be listed under different URLs and webpages. This problem is related to the near-duplicate detection problem in Web crawling [6, 16], but it focuses more on the *semantic* meaning of a cluster of webpages. For example, a webpage of user posts by date and a webpage of user posts by topic may be syntactically different, while the two clusters of webpages (by date and by topic) would have the same content.

Uninformative Pages: Online social communities are user oriented and interactive in nature. To facilitate easy interactions among users and update of individual or community information, most social websites provide extensive "action" webpages such as login, updating a profile, adding a friend, tell-a-friend, or commenting. Such "action" webpages provide no extra information for the crawling process yet may waste a lot of network bandwidth and computation

.

resources if not dealt with carefully.

Diverse and Complex URL/Link/Page Structures: Due to the diversity of people's interests and social interactions, social websites usually exhibit diverse and complex structures in terms of URL formats, hyperlinks, and page layout structures. For instance, users' profile pages are highly customizable, and these profile pages may link to other user, post, community pages with diverse content structures and URL characteristics. This differs in particular from well-structured Web forums, which tend to have regular URL formats as well as easy-to-follow link and page structures such as list of topics and page flipping links.

In addition, social websites are massive in size and highly dynamic, as they are visited by millions of people on daily basis. High efficiency and high scalability are thus essential for a social website crawler. However, due to the challenges listed above, previous Web crawlers designed for general websites or Web forums do not perform well on social websites. Not only do they waste computation resources, network bandwidth and storage space, the redundant or uninformative data gathered also cause extra overhead on future social network analysis or data mining tasks.

In this work, we first conduct a detailed analysis of social websites and identify their key characteristics. We have then designed and implemented an efficient crawler for social websites, which consists of four main components: (1) an offline random webpage sampling method to quickly generate a small yet representative sample set of webpages; (2) an offline URL-based clustering algorithm to identify semantically similar webpages; (3) an offline cluster graph pruning algorithm to remove redundant edges and unimportant clusters in the cluster graph; and (4) an online crawling process using cluster-graph-based filtering. Evaluation results on three different social websites demonstrate that our crawler can efficiently crawl large amounts of semantically important social content while dramatically reducing duplicate links, duplicate content, and uninformative webpages.

The rest of this paper is organized as follows. Section 2 presents the analysis and key characteristics of social websites. Section 3 gives an overview of our crawler design, and the details of the main components are presented in Section 4. Evaluation results are described in Section 5. Section 6 surveys the related work and Section 7 concludes.

2. ANALYSIS OF SOCIAL WEBSITES

In this section, we compare social websites with general websites and Web forums in order to identify the key characteristics of social websites for efficient crawler design.

After a careful examination of various types of online social communities, we pick three social websites of different sizes and different user communities. *LiveJournal* is a popular blogging website with over 17 million users, 19 million journals, and ∼200,000 new journals published daily. *Shelfari* is a popular social network for people who love books. And *Jaiku* is a social website that supports the sharing of short messages (activity streams) among friends. The number of daily visits of the three social websites are around 6 million for LiveJournal, 30,000 for Shelfari, and 10,000 for Jaiku [37]. From these three social websites, we collected a combined social websites data set of 2.3 million webpages. For comparison purposes, we also obtained a general websites data set of 2.3 million webpages, and a Web forums data set of 2.3 million webpages.

Table 1: Indegree of Different Websites

Websites	Sample size	Min	Max	Avg	Stdev
social	2.3M	0	875,646	60	3,872
general	2.3M	0	67,033	36	445
forum	2.3M	0	320,430	26	1,384

Table 2: Outdegree of Different Websites

Websites	Sample size	Min	Max	Avg	Stdev
social	2.3M	0	20,123	92	124
general	2.3M	0	10,272	35	68
forum	2.3M	0	8,288	46	61

Web graph structure. We first compare the graph structures of social websites, general websites, and Web forums, focusing on their indegree and outdegree, which are the number of hyperlinks pointing to or from a webpage. Table 1 and Table 2 show the min, max, average, and standard deviation of indegree and outdegree for the social websites, general websites, and Web forums, and Figure 1 compares their indegree and outdegree distributions. As shown in the figure and the tables, social websites have much higher indegree and outdegree than general websites and Web forums. E.g., the fraction of webpages with >= 10 indegree is 0.47 for social websites, 0.22 for general websites, and 0.10 for Web forums; and the fraction of webpages with >= 100 outdegree is 0.25 for social websites, and only 0.07 for general websites and Web forums. Such high indegree and outdegree of social websites would generate a huge number of duplicate links, resulting in long duplication check time when generic web crawlers are used.

Duplicate and uninformative content. Compared with general websites, the majority of the content in social websites are generated by individual users. Most social websites provide rich functionalities (web interfaces) for users to

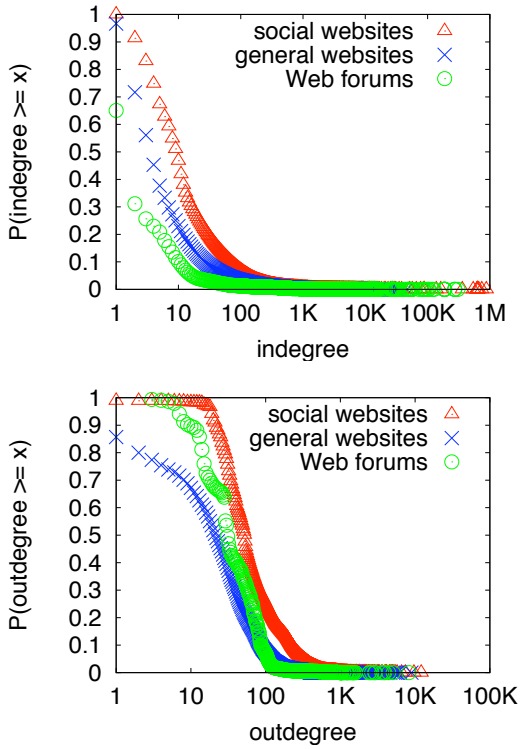

Figure 1: Comparison of indegree and outdegree distributions of social websites, general websites, and Web forums.

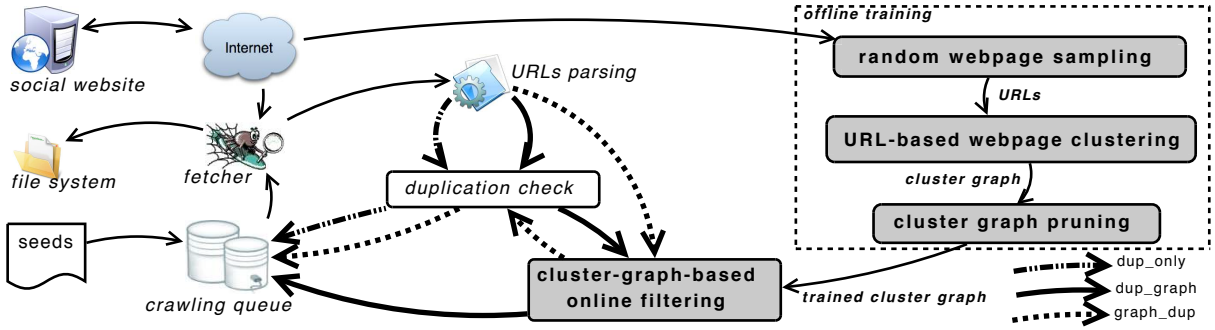

Figure 2: Overview: Efficient crawling of social websites (new components shown on the right).

Table 3: Duplicate or Uninformative Content in Different Websites

Websites	Sample size	Duplicate content pages (%)	Uninformative pages (%)
social	2.3M	21.49	12.57
general	2.3M	0.08	0.31
forum	2.3M	0.83	3.07

browse and create new content, as well as interacting with each other. For example, users can be listed by both friend list and community members, and user posts can be listed by name, by date, or by topic. There are also many "action" webpages for users to write a comment, add a new friend, or send a short message to a friend or a group. Although very useful and well-appreciated by social website users, these browsing and "action" webpages pose a major challenge for Web crawlers, as they typically contain duplicate or uninformative content. To confirm this observation, we analyze the content of social websites, general websites, and Web forums. For each website, we first cluster the webpages by their URL patterns (see details in Section 4.2), then manually examine the individual clusters to determine if a cluster represents duplicate or uninformative content. As shown in Table 3, the percentages of duplicate and uninformative pages in social websites are orders of magnitude higher than that of general websites and Web forums.

Diverse and complex URL/link/page structures. Unlike topic-centered general websites or Web forums, which are tailored for information viewing and have a limited or regular format for user-posted information, user-centered social websites provide much more functionalities and flexibility for users to create/customize their own webpages and link to a wide variety of webpages of other users and communities. For example, LiveJournal provides 788 templates for different journal styles, and each can be further customized by individual users. Since different users may have dramatically different interests and social interests, correspondingly, the social websites and online social communities they participate in tend to have very different URL patterns, hyperlink structures, and page layouts.

We further compare social websites with Web forums to explain their differences and why previous crawler designs for Web forums [12, 40] are not applicable to social websites.

Page layout. In a Web forum, the whole website has a uniform style and webpages in one cluster (e.g., list of threads) usually have the same layout. However, in a social website, various templates exist (e.g., 788 journal styles at LiveJournal) and can be further customized by users. This results in a large number of content layout types, even for webpages with similar semantics (e.g., user profile pages). Therefore, using DOM trees to identify semantic webpage clusters may not work for social websites.

URL format. In a Web forum, URLs are regularly formatted, and only a few variables (e.g., board ID, post ID) exist at specific positions in the path or query component of URL strings. In a social website, many variables exist (e.g., user, tag, community) and can appear at various positions (e.g., authority, path, query) in URL strings. Such complex and diverse URL formats make URL-based webpage clustering a much more challenging problem in social websites than in Web forums.

Link structure. A Web forum allows users to browse and participate in threaded discussions. Links between Web forum pages are usually well structured and exhibit specific characteristics, which can be used for identifying valuable links in a Web forum [40]. Social websites support a wide variety of functionalities, and allow users to create more diverse link structures to support their social interaction and information sharing needs. Such mixed and noisy link structures make it difficult to identify specific types of valuable links at the page level. Instead, our crawler design focuses on pruning unimportant links at the cluster graph level.

Based on the analysis and comparison above, in order for a Web crawler to efficiently gather large amounts of semantically-important social data from social websites, it is essential that the crawler can quickly yet accurately identify and prune out duplicate links, duplicate content, and uninformative webpages, tackling the diverse and complex URL formats, link structures, and page layouts in social websites.

3. OVERVIEW OF CRAWLER DESIGN

In this section, we give an overview of the proposed efficient Web crawler targeting social websites. We start with a brief description of generic Web crawler design, then highlight the main design contributions of the new crawling system. The detailed design of the crawling system is presented in Section 4.

Figure 2 illustrates the generic Web crawler design (left) and the new design components (right). As shown on the left, a generic Web crawler first gets a URL from the crawling queue and fetches the corresponding webpage. It then parses the URLs (hyperlinks) contained in that page. The duplication check step filters out URLs that are already crawled or added to the crawling queue, and the remaining URLs are inserted into the crawling queue. However, when crawling social websites, a generic Web crawler would spend a lot of time checking and pruning duplicate links, and it generally

fails to detect duplicate or uninformative content and ends up downloading all these content, which is wasteful and affects the overall crawling efficiency and effectiveness.

The right diagram in Figure 2 highlights the new components in our crawler design, which utilizes an offline training process to generate a cluster graph, and uses the cluster graph at runtime to guide the crawling processing. Compared with generic Web crawlers, our crawler consists of four new components, and the first three components are used in offline training:

Random Webpage Sampling: First, an improved random webpage sampling method is developed to quickly crawl a small yet representative set of sample webpages from a given social website.

URL-Based Webpage Clustering: The sampled webpages are then clustered into various semantic groups using FP-growth based frequent itemset mining and a hierarchical clustering algorithm based on their URL patterns.

Cluster Graph Pruning: The URL clusters are connected into a weighted directed cluster graph by examining the link structures among the URLs, and our cluster graph pruning algorithm then effectively removes certain nodes and edges in the cluster graph representing duplicate links, duplicate content, or uninformative pages.

Cluster-Graph-Based Online Filtering: The offline-trained cluster graph is used at runtime to guide the crawling process, filtering out newly-parsed URLs that do not belong to any cluster in the cluster graph, or if the hyperlinks they represent do not have a corresponding edge in the cluster graph.

Note that the cluster-graph-based filtering step can be applied either before (dotted line in Figure 2) or after (solid line in Figure 2) the duplication check step. As shown in the evaluation results (Section 5), it is more efficient to perform cluster-graph-based filtering before duplication check.

4. SOCIAL WEBSITE CRAWLER DESIGN

In this section, we describe in detail the four main components of our social website crawler: (1) random webpage sampling; (2) URL-based webpage clustering; (3) cluster graph pruning; and (4) cluster-graph-based filtering. The first three components are used for offline training of the cluster graph, which is then used in the fourth component for online crawling of social websites.

4.1 Random Webpage Sampling

Given a specific social website to crawl, our first task is to quickly generate a small yet representative set of sample webpages of the social website. The sampled set should represent webpages of various URL patterns and link structures with roughly the same proportions as that of the complete social website. Traditional webpage sampling or crawling techniques that use breadth-first, depth-first, or highest-degree-node-first exploration are not effective as they tend to generate localized, biased, or even skewed samples. A simple random sampling approach that selects a URL uniformly random from the current crawling queue cannot achieve overall uniform random sampling since the URLs discovered earlier have much higher chances of being selected than URLs that are inserted to the crawling queue at a later time.

We propose an improved algorithm for random webpage sampling. The main steps of the algorithm are shown in Algorithm 1. When a newly discovered URL u_i is inserted to the crawling queue, its initial probability of already sampled s_i, probability of not yet sampled r_i, and remaining sampling probability p_i are set as follows:

$$s_i = 0; \qquad r_i = 1; \qquad p_i = \xi - s_i = \xi;$$

where ξ is a parameter specifying the desired sampling probability of each URL. To select the next URL to crawl, we examine all the URLs in the current crawling queue and compute the normalized sampling probability q_i for each URL u_i: $q_i = p_i / \sum_{u_j \in U} p_j$. We then randomly select a URL with the corresponding normalized sampling probability. Let u_k be the URL selected, it is moved from the crawling queue U to the crawled set V. The probabilities of each of the remaining URLs u_i in the crawling queue are adjusted accordingly to reflect the new s_i, r_i, and p_i values.

$$s_i = s_i + r_i \times q_i; \qquad r_i = r_i \times (1 - q_i); \qquad p_i = \xi - s_i;$$

If u_i's remaining sampling probability p_i is 0 or less, it is removed from the crawling queue. Next, we crawl u_k, identify each of the newly discovered URLs u_j, and add it to the crawling queue with the initial values of s_j, r_j, and p_j.

Our random webpage sampling method uses a single parameter ξ, the desired sampling probability of each individual URL. This parameter controls how quickly we move away from the initial seed URLs. The larger ξ is, the slower we move in the web graph, sampling more in a local region. The smaller ξ is, the faster we move in the web graph but may generate skewed samples as in depth-first sampling. To choose an appropriate value for ξ, let us consider the average outdegree of a social website, $\overline{K}$. Let K be the number of newly discovered URLs after crawling each URL (i.e., fan-out). Initially, the expected value of K equals $\overline{K}$ since most of outgoing links are new. As more and more URLs are

Algorithm 1 Random Webpage Sampling

```
U: URLs in the crawling queue
V: URLs crawled
W: URLs withdrawn
ξ: desired sampling probability for each URL
s_i: probability of URL u_i already sampled
r_i: probability of URL u_i not yet sampled
p_i: remaining sampling probability of URL u_i
q_i: normalized sampling probability of URL u_i

U = {seed URLs}; V = ∅; W = ∅
for each u_i ∈ U do
   s_i = 0; r_i = 1; p_i = ξ;
end for

while crawl next URL do
   for each u_i ∈ U do
      q_i = p_i / Σ_{u_j ∈ U} p_j
   end for
   randomly select a URL u_k with probability q_k
   U = U − {u_k}; V = V + {u_k}
   X = {u_j : u_k links to u_j, u_j ∉ U, u_j ∉ V, u_j ∉ W}
   for each u_i ∈ U do
      s_i = s_i + r_i × q_i; r_i = r_i × (1 − q_i); p_i = ξ − s_i
      if p_i ≤ 0 then
         U = U − {u_i}; W = W + {u_i}
      end if
   end for
   for each u_j ∈ X do
      s_j = 0; r_j = 1; p_j = ξ; U = U + {u_j}
   end for
end while
```

Table 4: Semantic Meanings of URL Clusters

Example cluster	Semantic meaning
http://*.livejournal.com/	user homepage
http://*.livejournal.com/profile	user profile page
http://*.livejournal.com/$.html	user post
http://*.livejournal.com/tag	user's list of tags
http://www.livejournal.com/$.html	website news

crawled, many of the outgoing links may have already been crawled or exist in the crawling queue, so K would decrease over time. When sampling a small subset of webpages, K is expected to be only slightly smaller than the average outdegree $\overline{K}$. To ensure proper execution of the sampling process, the value of ξ should be within the range $[1/K, 1]$. Based on this analysis, to generate a small sample set that is diverse yet representative, we choose the following ξ value: $\xi = \frac{1}{\overline{K}} + \frac{1-\frac{1}{\overline{K}}}{\alpha}$, where constant α is a positive integer to ensure that ξ is slightly larger than $1/\overline{K}$.

4.2 URL-Based Webpage Clustering

The goal of webpage clustering is to identify semantically coherent clusters, such as user profiles or group posts. Table 4 lists some example clusters and their semantic meanings. One key challenge is to distinguish *variable keywords* in URL strings, such as user ID, post ID, tag ID, and date, from *functional keywords*, such as profile, news, etc. Existing clustering methods that use webpage content or structural layout do not work well for social website pages, since semantically similar webpages (e.g., users' profile pages) may differ dramatically in terms of content, link structure, and (customized) page layout.

There are also several challenges when using URLs for webpage clustering. First, URLs with similar syntax may belong to different semantic clusters, e.g., URL 1 and URL 2 in Figure 3. Note that the pattern/cluster of URL 2 is actually a superset of the pattern/cluster of URL 1. Second, the semantic words contained in URLs (e.g., profile, post, group) are easily understood by human but difficult for computers. What is more, URLs of social webpages tend to be much noisier than general websites and Web forums, since social website users have more flexibility in creating and naming new content (e.g., URLs 3, 4, 5, 6 in Figure 3). Traditional distance-based clustering method is inadequate. For example, in Figure 3, the distances of URL 3&4 and URL 3&5 are difficult to determine. Since each website has its own URL formats, we have no prior knowledge about the positions of those variables. Third, *functional keywords* in URL strings may vary with websites. For example, the keyword "profile" might correspond to"main", "info", or even digits in other websites. A predefined set of *functional keywords* in URLs may not work for social websites.

To address these issues, we propose a novel URL-based webpage clustering algorithm based on FP-growth frequent itemsets mining [21] and hierarchical clustering [9]. This is based on the observation that functional keywords appear frequently in URL strings and usually at specific positions. Such URL patterns are generally stable and correspond to the semantics of different webpages.

As shown in Algorithm 2, our clustering algorithm works in four stages: 1) conversion from URLs to item vectors; 2) FP-growth frequent itemsets mining; 3) hierarchical clustering tree construction; and 4) URL pattern generation.

At the first stage, each sampled URL is converted into a vector. A URL is a relatively semi-structured string, which can be treated as a sequence of substrings separated by several special characters. According to [38], a URL can have five parts: scheme, authority, path, query, and fragment. For example, the following URL

`http://news.example.com/over/there?name=dog\#nose`

can be separated as: "http://" (scheme); "news.example.com" (authority); "/over/there" (path); "?name=dog" (query); and "#nose" (fragment).

After getting the five parts for each URL, we use authority, path, and query for clustering since these three parts contain the most important information of a URL. Each part is further split into items, using separation characters '.' for authority, '/' and '.' for path, '&' and '=' for query. The final item vector for the example URL above is

$$[\{\textbf{news}, \textbf{example}, \textbf{com}\}, \{\textbf{over}, \textbf{there}\}, \{\textbf{name}, \textbf{dog}\}].$$

At the second stage, all the URL vectors are divided into M equal-sized bins $(b_1, \ldots, b_M)$. For each of the three URL parts (authority, path, query) and each bin, we use the FP-growth algorithm to identify all frequent itemsets with a *minimum support value* of S (i.e., minimum number of occurrences of an itemset in a bin). Let $F(b_i)$ $(i = 1, \ldots, M)$ be

Algorithm 2 URL-Based Webpage Clustering

```
U = {all sampled URLs}, N = |U|

for each u ∈ U do
    split u into a vector of three parts: authority, path, query
    each part contains one or multiple items
    v = [{A_1, A_2, ···}, {P_1, P_2, ···}, {Q_1, Q_2, ···}]
end for
split all URL vectors into M partitions of equal size
for each of three parts (Authority, Path, Query) do
    for i = 1 to M do
        find frequent itemsets S_i for the ith bin
        using the FP-growth algorithm
    end for
    keep the frequent itemsets that appeared in all M bins
    sort frequent itemsets in ascending order of itemset size
end for

initiate pattern tree with a single root node T = {R}
for each u ∈ U do
    A = largest matched Authority frequent itemset of u
    if A is not empty then
        create node A as child of R if A not exist in T
        P = largest matched Path frequent itemset of u
        if P is not empty then
            create node P as child of A if P not exist in T
            Q = largest matched Query frequent itemset of u
            if Q is not empty then
                create node Q as child of P if Q not exist in T
                add u to the set of URLs in Q
            end if
            add u to the set of URLs in P
        end if
        add u to the set of URLs in A
    end if
    add u to the set of URLs in R
end for
start from the leaf nodes in T and go upwards
for each non-empty node X in T do
    generate patterns for the set of URLs in X
end for
sort all patterns in increasing size (# matched URLs)
remove the smallest patterns s.t.
the remaining patterns cover (1 − δ) × N URLs
```

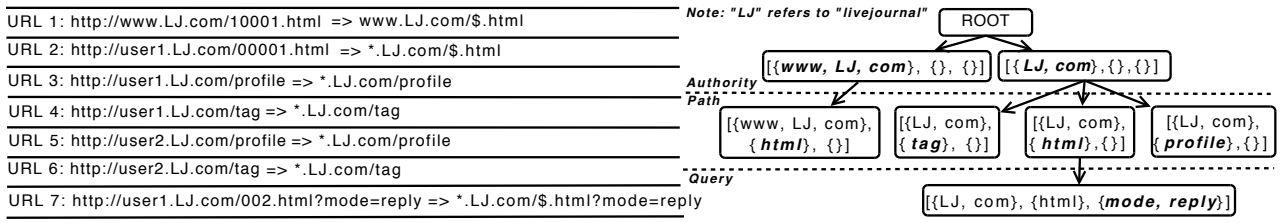

Figure 3: Example URLs 1-7, and their clustering subtree with three layers (authority, path, and query).

the frequent itemsets in the i-th bin, then the final frequent itemsets $I = F(b_1) \cap F(b_2) \cap \ldots \cap F(b_M)$. By partitioning the URLs into multiple bins and only considering itemsets that are frequent in all bins, we can reduce the false positive rate and effectively identify itemsets corresponding to valuable semantic clusters. FP-growth adopts an efficient divide-and-conquer strategy. It first compresses the database by representing frequent items into an FP-tree, which retains the itemset association information. It then divides the compressed database into a set of conditional databases, each associated with one frequent item, and mines each conditional database separately. The number of bins M and the minimum support value S in each bin should be set such that stable frequent itemsets can be detected. Based on our experiments, $M = 6$ and $S = 5$ generally work well.

At the third stage, we generate a three-layer clustering tree, where authority, path, and query correspond to the top, middle, and bottom layer, respectively. We start with a tree T with a single root node R. For each of the sampled URLs, we find its largest matched frequent itemset for each of the three parts: authority, path, and query. We then add a branch to T with the corresponding nodes and frequent itemsets. Each URL is added to the bottom-level node of matched frequent itemset (see Algorithm 2). For example, if we have two frequent itemsets for the authority part:

$\{\textbf{news}, \textbf{example}, \textbf{com}\}$ and $\{\textbf{example}, \textbf{com}\}$,

URL `http://user1.example.com` will be converted to vector **[{user1, example, com},{}, {}]**, and its largest matched authority frequent itemset is **{example, com}**. In this case, a new node A will be created as a child of R, and the URL will be added to the set of URLs in A.

After all the sampled URLs have been assigned to a node in the hierarchical tree, at the last stage, we go through each node in the tree, starting from the leaf nodes and going upwards. For each non-empty node, we examine the URLs assigned to this node and generate patterns by replacing the unmatched items with a '$' if it is a string of digits, or a '*' for anything else. The reason we distingush digits strings from general strings is that for some cases the format of string is the only difference of two clusters. For example, user1.example.com/*.html stands for user1's posts with tag *, and user1.example.com/$.html stands for user1's single post $. We have observed such difference in a few social websites.

After all the patterns are generated, we remove the smallest patterns (measured by the number of URLs matching each pattern) such that the fraction of URLs covered by the remaining patterns is at least $1 - \delta$. In our experiments, we set δ to 0.03 to ensure a 97% coverage of all the sampled URLs. By removing the small patterns, we can effectively remove the noisy or unimportant patterns and dramatically reduce the number of patterns that we have to consider in the cluster graph pruning component.

Figure 3 shows the clustering subtree corresponding to example URLs 1-7. For example, URL 3's vector is [{user1, LJ, com}, {profile}, {}], and its best match is the rightmost node in the path layer, and the corresponding URL pattern for that node is `*.LJ.com/profile`.

4.3 Cluster Graph Pruning

The patterns (clusters) generated from the previous step identify semantically coherent webpages such as user profile pages or group post pages. However, webpages in these clusters still contain duplicate content, uninformative pages, and duplicate links, which we aim to remove through the cluster graph pruning step.

Given the clusters of webpages, we create a corresponding cluster graph as follows:

- A cluster graph $G = (V, E)$ is a weighted directed graph.
- G contains $|V|$ nodes, where each node v_i represents a cluster C_i, and its size is the number of URLs contained in that cluster.
- For each pair of clusters C_i and C_j, if there exists URLs $u_i \in C_i, u_j \in C_j$, and u_i links to u_j, then there is a directed edge e_{ij} points from C_i to C_j.
- Each edge e_{ij} is associated with two weights: one is the number of URLs in C_i pointing to a URL in C_j, and the other is the number of URLs in C_j that are being pointed to by a URL in C_i.

Our cluster graph pruning algorithm includes both edge deletion for the removal of duplicate links in a social website, and node deletion for the removal of duplicate or uninformative content in a social website.

First, for edge deletion, we examine the incoming edges of each cluster C_i in the cluster graph. The intuition here is that if the URLs in cluster C_i can be reached via multiple incoming edges, we can safely remove some of the incoming edges (thus reducing duplicate webpage links) while still maintaining a good coverage (reachability from other clusters) of the URLs in cluster C_i. Specifically, we first sort all the incoming edges in descending order by their coverages of C_i. Starting from single edges, we use dynamic programming to find the coverage of every m-edge combinations. If the highest coverage is greater than the required coverage, we have found the best solution. Otherwise, we continue to consider all $m + 1$ edge combinations. Once we have identified the best edge combination, we can delete the remaining incoming edges of cluster C_i. Based on our experiments, a required coverage of 0.6 is sufficient (Section 5).

After deleting the redundant edges in the cluster graph, we also examine individual nodes in the graph and delete the ones that correspond to duplicate content or uninformative webpages, based on the following heuristics:

Header and footer pages: Pages such as "term of use" and "contact us" are linked by almost all webpages on a website, resulting in a huge number of duplicate links. Since the content on these pages are generally not needed for social network analysis, we can safely remove them. If a cluster is linked to by more than half of the other clusters, and the total number of incoming URL links is more than half of the total number of sampled URLs, we consider this cluster a header/footer cluster and the corresponding node is deleted from the cluster graph.

Duplicate content: Consider the following scenario: a user-post cluster C_k was originally pointed to from both a post-by-date cluster C_i and a post-by-topic cluster C_j. In this case, C_i and C_j contain duplicate content, and either e_{ik} or e_{jk} would be deleted during the edge deletion phase. After edge deletion, if both C_i and C_k have no outgoing edges (i.e., leaf nodes), we can then delete node C_i, thus removing the duplicate content in C_i.

Uninformative pages: After edge deletion, if a cluster node has no outgoing edges, and the cluster itself contains a small number of URLs (e.g., less than the average cluster size - 2 $\times$ the standard deviation of cluster size), this cluster is considered uninformative and deleted.

4.4 Cluster-Graph-Based Filtering

Once the cluster graph has been trained offline (using random webpage sampling, URL-based webpage clustering, and cluster graph pruning), we can start the online crawling process. As shown in Figure 2, every time a new URL u_i is selected, the crawler retrieves the corresponding webpage, parses the HTML file, and generates a list of URLs (hyperlinks from u_i). We check each URL u_j in the list and remove it from the list if

1. u_j does not map to any cluster in the cluster graph, or
2. u_j maps to cluster C_j and u_i maps to cluster C_i, but there is no edge from C_i to C_j in the cluster graph.

The remaining list of URLs are then inserted into the crawling queue for future crawling.

Note that the cluster-graph-based filtering of URLs can be performed either before or after the duplication check step. As shown in the evaluation results, it is more effective to perform the cluster-graph-based filtering before duplication check (Section 5).

5. EVALUATIONS

In this section, we evaluate the performance of our social website crawler. Specifically, we are interested in answering the following questions:

1. How efficient is the cluster-graph-based filtering process? How does it compare to the duplication check process?
2. What is the quality of the crawled webpages? How does it compare to that of the generic crawler?
3. How effective is the cluster-graph-based filtering process, in terms of removing duplicate links, duplicate content, and uninformative webpages?
4. How big a sample set is needed for the offline training process?

5.1 Experimental Setup

Our evaluations are conducted on three popular social websites: LiveJournal, Shelfari, and Jaiku. As described in Section 2, these three websites have different sizes and different user communities. In our evaluations, we consider the following three crawlers:

- **Dup-only (d)**: A generic breadth-first crawler using only duplication check.
- **Dup-graph (dg)**: Proposed crawler using first duplication check, then cluster-graph-based filtering.
- **Graph-dup (gd)**: Proposed crawler using first cluster-graph-based filtering, then duplication check.

All three crawlers are implemented in Java and the offline cluster graph training algorithms are implemented in Python. To support efficient duplication check, we adopt the DRUM (Disk Repository with Update Management) technique proposed by Lee et al. [27]. All experiments are conducted on workstations with the same configuration, with 4GB memory and 2.66GHz Quad-core CPUs.

5.2 Efficiency of the Crawler

To evaluate the efficiency of crawling, we focus on the duplication check step and the cluster-graph-based filtering step, as the other steps (parsing, fetching, storing, etc.) have the same time complexity for all the three different crawlers.

Figure 4 compares the time spent per URL for duplication check and cluster-graph-based filtering of the three crawlers and the three social websites. As shown in the figure, for each URL, duplication check takes about 0.05ms and cluster-graph-based filtering takes about 0.01ms, and both latencies remain stable as the number of URLs increases. This shows that the overhead of cluster-graph-based filtering is small compared to that of duplication check.

Figure 5 compares the total time spent on duplication check and cluster-graph-based filtering by the dup-graph and graph-dup crawlers with that of the dup-only crawler. As shown in the figure, compared with the duplication check time by the dup-only crawler, dup-graph takes 94–99% time for duplication check and 6–8% time for graph filtering; and graph-dup is much more efficient, taking only 19–22% time for duplication and 15–17% time for graph filtering. In other words, the total time spent by graph-dup for duplication check and graph filtering is only 35–39% of the time spent by dup-only for duplication check.

For the dup-graph crawler and the graph-dup crawler, duplicate links can be detected and filtered out by either the duplication check step or the cluster-graph-based filtering step. In the generic crawler, duplicate links can only be filtered by the duplication check step. As shown in Figure 6, the dup-only crawler removes 70 – 90% duplicate links through duplication check. For the dup-graph crawler, the duplication filtered by duplication check is reduced to 50% on average. The reason is that the cluster-graph-based filtering step helps to select pages with less duplicate links. The graph-dup crawler has the lowest duplication check rate. For Shelfari and Jaiku, the duplication rate is only 2% because the cluster-graph-based filtering step effectively removes most of the duplicate links. As we have seen in Figure 4, cluster-graph-based filtering is 5$\times$ faster than duplication check and scales well with the number of URLs. Thus, the more duplicate links handled by the graph filtering step, the more efficient the crawling process.

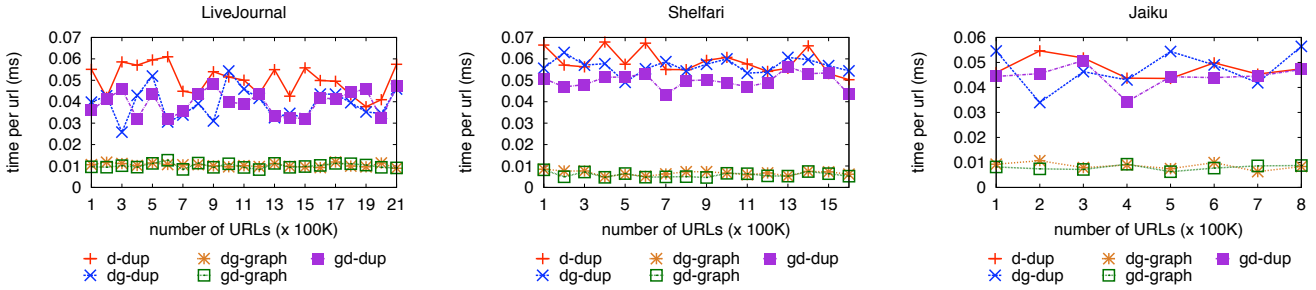

Figure 4: Speed comparison of duplication check and cluster-graph-based filtering.

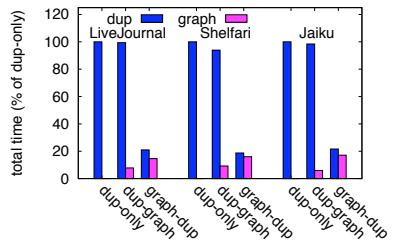

Figure 5: Comparison of total time spent on duplication check and cluster-graph-based filtering.

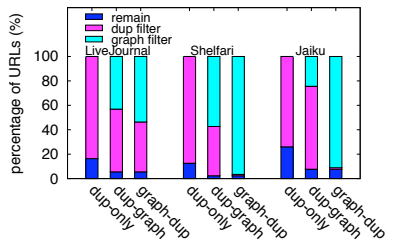

Figure 6: Percentage of URLs filtered out by duplication check or cluster-graph-based filtering.

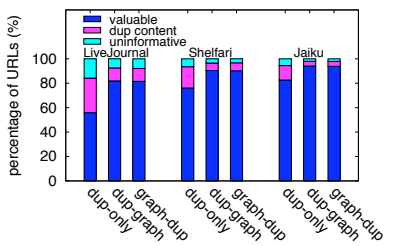

Figure 7: Quality comparison of the webpages crawled using the three different crawlers.

5.3 Quality of the Crawled Webpages

Next, we evaluate the quality of the crawled data, compared with the generic crawler. For each cluster in the trained cluster graph, we manually categorize it into three groups: valuable/informative clusters, duplicate content clusters, and uninformative clusters. Valuable/informative clusters are those clusters that contain pages with useful information such as user profiles. Duplicate content clusters contain pages whose content also exist in other clusters. As an example, consider post list pages sorted by date and post list pages sorted by topic. If both types of pages are crawled, then one type is a duplicate content cluster and the other type is a valuable/informative cluster. Uninformative clusters contain pages with no or little useful information, such as web interface pages that let users leave comments or add new friends. While some of these uninformative pages can be filtered by checking the robots.txt file provided by a website, others are not and need to be detected by a crawler.

As shown in Figure 7, both the dup-graph crawler and the graph-dup crawler achieve 11% – 26% better data quality than the generic crawler. The percentage of valuable pages downloaded by the dup-only crawler is 56% for LiveJournal, 76% for Shelfari, and 83% for Jaiku, while the percentage of valuable pages downloaded by the dup-graph crawler and the graph-dup crawler is 82% for LiveJournal, 90% for Shelfari, and 94% for Jaiku. Table 5 shows four clusters representing the three different content categories, and compare the number of URLs downloaded by each of the three crawlers. The first is a valuable user profile cluster, the next two are duplicate content clusters, and the last one is an uninformative commenting cluster. As shown in the table, both the dup-graph crawler and the graph-dup crawler downloaded many URLs for the valuable cluster and one of the two duplicate content clusters, and zero URL from the other two clusters. On the other hand, the generic crawler downloaded fewer URLs for the first two clusters, but many URLs for the last two clusters.

We have seen from the previous results that cluster-graph-based filtering can efficiently remove unwanted data while preserving the valuable data. Here, we further analyze the false positive rate (FP) and the false negative rate (FN) of this method.

Table 6 shows the false positive rate and the false negative rate for each of the two crawlers (dup-graph and graph-dup) and each of the three social websites. Here, useless pages include both duplicate content pages and uninformative pages. As shown in the table, our cluster-graph-based filtering method has low false negative rates (0.7% – 2.1%) but higher false positive rates (5.9% – 18.5%). The results depend on how we choose the thresholds during the offline training process. Generally, it is more important to maintain a low false negative rate since we want to ensure that the crawler downloads all the valuable information, while a slightly higher false positive rate is acceptable as we can easily remove the false positives after they are downloaded.

5.4 Sample Size for Offline Training

Compared to the generic crawler, proposed crawling system requires an extra offline training process in order to generate the cluster graph used for online filtering. Here, we study how many sampled webpages are needed to achieve good training results. Although it is possible to sample as many webpages as possible, a larger sample size means long time for sampling and the later training steps. Larger sample size may not always improve the quality of training as it may introduce more noisy to the training process.

For each of the three social websites, we experimented with increasingly-larger sample sizes and stop when the training results stabilize. We consider the following measures of

Table 6: False Positive and False Negative of Cluster-Graph-Based Filtering

Website	dg FP(%)	dg FN(%)	gd FP(%)	gd FN(%)
LiveJournal	18.1	2.0	18.5	2.1
Shelfari	9.6	0.9	9.8	1.1
Jaiku	5.9	0.7	6.1	0.7

Table 5: Comparison of the Number of URLs Crawled for Each Content Category

Content category	Cluster pattern	Dup-only	Dup-graph	Graph-dup
valuable: User profile	http://*.livejournal.com/profile	74,827	208,289	208,289
content dup: User posts ordered by time	http://*.livejournal.com	32,452	76,293	75,345
content dup: User posts ordered by tag	http://*.livejournal.com/tag/*	30,595	0	0
uninformative: comments	http://*.livejournal.com/$.html?mode=reply	4,903	0	0

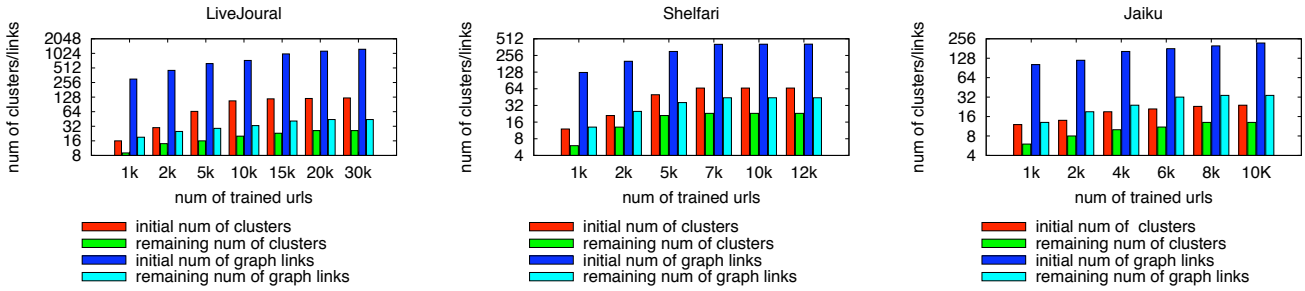

Figure 8: Number of sampled URLs needed for offline cluster graph training.

the training results: (1) the total number of clusters the training process generates; (2) the number of links among all clusters; (3) the number of remaining clusters after cluster graph pruning; (4) the number of links remaining after cluster graph pruning. As shown in Figure 8, for LiveJournal, when the number of sampled URLs is greater than 15K, the four measures stabilize. We can then infer that 15K sampled URLs are sufficient for offline training for LiveJournal. Similarly, based on the results in the figure, we can infer that 7K and 6K sampled URLs are sufficient for the offline training of Shelfari and Jaiku, respectively.

6. RELATED WORK

Our work draws upon research in several areas concerning Web crawling: webpage and graph sampling, Web crawler design, near-duplicate webpage detection, as well as graph node and edge deletion. In this section, we survey research work most related to ours.

Analysis and understanding of Web graph characteristics are important for Web crawler design. Various Web properties have been studied [26, 11, 1, 18]. Several techniques have been proposed for graph sampling [20, 39, 22] and in particular for Web graph sampling [5, 24, 34]. Previous studies show that no sampling method works well in the presence of duplicate links, duplicate content, or invalid nodes [7]. Through improved random webpage sampling, URL-based webpage clustering, cluster graph pruning, and online filtering, our crawler can effectively identify and prune duplicate links, duplicate content, and uninformative webpages when crawling social websites.

Extensive research has been conducted in the area of Web crawler design. One related subarea is focused crawling [13], which crawls only a subset of the Web pertaining to a specific topic. One general approach is to prioritize the pages to be crawled using link-based or content similarity-based page importance measures [2, 10, 15, 29, 42]. Further improvements include reinforcement crawling [33] and context graph-based crawling [17, 25]. Our work differs from focused crawling in that we need to crawl all useful information in social websites instead of specific topics. Moreover, current focused crawlers do not address the problems of duplicate content, links and invalid pages. Other related Web crawler designs include random crawler [8], salable crawler design [27], and the recently developed iRobot crawler for Web forums [12, 40]. Due to the user-centered nature of social websites and the diversity in user interests and social interactions, previous crawler designs targeting general websites or Web forums do not work well when crawling social websites. Schonfeld et al. proposed using sitemaps in Web crawling [35]. However, sitemaps can not solve the challenges of social websites crawling. Also, social websites generally do not provide a sitemap file because of its user-centered nature, each user has her own "sitemap" instead of a global sitemap.

Content-based near-duplicate webpage detection has been studied previously [23, 28, 41]. These techniques detect and remove near-duplicate webpages after they have been downloaded from the Web. Recent work [6, 16] aims to learn URL rewrite or transformation rules such that webpages with duplicate content can be detected by the URL. These approaches focus on pair-wise syntactic similarity between webpages. However, on social websites, a group of webpages may contain duplicate content as another group of webpages (e.g., user posts by date or by topic), but the pairwise similarity between two webpages may be low. However clustering webpages into semantic groups and examine their link structures, our crawler can effectively detect and prune this type of duplicate content. Node and edge deletion problems have been subjected to several theoretical studies and most of them have been shown to be NP-complete [4, 43]. Our heuristics for cluster graph pruning can efficiently and effectively remove redundant edges or unimportant nodes.

7. CONCLUSIONS AND FUTURE WORK

Crawling large-scale social data from social websites is an essential building block for the increasingly-active online social network-based research including social network analysis, data mining, information dissemination and recommendation, social behavior study, and research on online security and privacy issues. In this work, we have analyzed and identified several key challenges of crawling social websites: duplicate links, duplicate content, uninformative webpages, diverse and complex URL/link/page structures. We have then developed an efficient Web crawler for social websites. Our crawler utilizes an offline training process of random webpage sampling, URL-based webpage clustering, and

cluster graph pruning for identifying semantically-important URL clusters and most effective cluster link structures. The trained cluster graph is then used during online crawling to quickly and accurately filter out unneeded webpages. Experimental results on three social websites of different sizes and different user communities show that our crawler can efficiently crawl large amounts of socially-meaningful webpages while dramatically reducing duplicate links, duplicate content, and uninformative webpages.

This work is our first step towards efficient crawling of large-scale online social communities. We are currently collaborating with sociologists to identify important social behavioral and structural features to crawl. We will continue to evaluate the performance of our crawler on other social websites. Webpage content and structure may be integrated in our crawler design. We also plan to investigate revisit strategies for crawling dynamic social websites, which is an important aspect in understanding the evolution of social behavior and information dissemination.

8. REFERENCES

[1] A. Broder et al. Graph structure in the web. In *Proc. WWW'00*, pages 309–320.

[2] S. Abiteboul, M. Preda, and G. Cobena. Adaptive on-line page importance computation. In *Proc. WWW'03*, pages 280–290.

[3] L. A. Adamic, J. Zhang, E. Bakshy, and M. S. Ackerman. Knowledge sharing and yahoo answers: everyone knows something. In *Proc. WWW'08*, pages 665–674.

[4] T. Asano and T. Hirata. Edge-deletion and edge-contraction problems. In *Proc. STOC'82*, pages 245–254.

[5] Z. Bar-Yossef, A. Berg, S. Chien, J. Fakcharoenphol, and D. Weitz. Approximating aggregate queries about web pages via random walks. In *Proc. VLDB'00*, pages 535–544.

[6] Z. Bar-Yossef, I. Keidar, and U. Schonfeld. Do not crawl in the dust: Different urls with similar text. *ACM Trans. Web*, 3(1):1–31, 2009.

[7] L. Becchetti, C. Castillo, D. Donato, and A. Fazzone. A comparison of sampling techniques for web characterization. In *Workshop on Link Analysis (LinkKDD)*, August 2006.

[8] T. Bennouas and F. de Montgolfier. Random web crawls. In *Proc. WWW'07*, pages 451–460.

[9] P. Berkhin. Survey of clustering data mining techniques. Technical report, Accrue Software, San Jose, CA, 2002.

[10] K. Bharat and M. R. Henzinger. Improved algorithms for topic distillation in a hyperlinked environment. In *Proc. SIGIR'98*, pages 104–111.

[11] A. Bonato and W. Laurier. A survey of models of the web graph. In *Proc. CAAN'04*, pages 159–172.

[12] R. Cai, J.-M. Yang, W. Lai, Y. Wang, and L. Zhang. irobot: an intelligent crawler for web forums. In *Proc. WWW'08*, pages 447–456.

[13] S. Chakrabarti, M. van den Berg, and B. Dom. Focused crawling: a new approach to topic-specific web resource discovery. *Computer Networks (Amsterdam, Netherlands: 1999)*, 31(11–16):1623–1640, 1999.

[14] D. H. Chau, S. Pandit, S. Wang, and C. Faloutsos. Parallel crawling for online social networks. In *Proc. WWW'07*, pages 1283–1284.

[15] J. Cho, H. Garcia-Molina, and L. Page. Efficient crawling through url ordering. In *Proc. WWW'98*, pages 161–172. Elsevier Science Publishers B. V.

[16] A. Dasgupta, R. Kumar, and A. Sasturkar. De-duping urls via rewrite rules. In *Proc. KDD'08*, pages 186–194.

[17] M. Diligenti, F. Coetzee, S. Lawrence, C. L. Giles, and M. Gori. Focused crawling using context graphs. In *Proc. VLDB'00*, pages 527–534, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[18] D. Donato, L. Laura, S. Leonardi, and S. Millozzi. The web as a graph: How far we are. *ACM Trans. Interet Technol.*, 7(1):4, 2007.

[19] Flickr API. `http://www.flickr.com/services/api/`.

[20] L. Goodman. Snowball sampling. In *Annals of Mathematical Statistics 32*, pages 148–170, 1961.

[21] J. Han, J. Pei, Y. Yin, and R. Mao. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data Min. Knowl. Discov.*, 8(1):53–87, 2004.

[22] D. D. Heckathorn. Respondent-driven sampling: A new approach to the study of hidden populations. In *Social Problems*, 1997.

[23] M. Henzinger. Finding near-duplicate web pages: a large-scale evaluation of algorithms. In *Proc. SIGIR'06*, pages 284–291.

[24] M. R. Henzinger, A. Heydon, M. Mitzenmacher, and M. Najork. On near-uniform url sampling. In *Proc. WWW'00*, pages 295–308.

[25] C.-C. Hsu and F. Wu. Topic-specific crawling on the web with the measurements of the relevancy context graph. *Inf. Syst.*, 31(4):232–246, 2006.

[26] J. M. Kleinberg, R. Kumar, P. Raghavan, and A. S. Tomkins. The web as a graph: Measurements, models and methods. In *Proc. COCOON'99*.

[27] H.-T. Lee, D. Leonard, X. Wang, and D. Loguinov. IRLbot: scaling to 6 billion pages and beyond. In *Proc. WWW'08*, pages 427–436.

[28] G. Manku, A. Jain, and A. Das Sarma. Detecting near-duplicates for web crawling. In *Proc. WWW'07*, pages 141–150.

[29] F. Menczer, G. Pant, P. Srinivasan, and M. E. Ruiz. Evaluating topic-driven web crawlers. In *Proc. SIGIR'01*, pages 241–249.

[30] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee. Measurement and analysis of online social networks. In *Proc. IMC'07*, pages 29–42.

[31] Myspace.com: site information from alexa. `http://www.alexa.com/data/details/main/myspace.com`.

[32] Global faces and networked places: A Nielsen report on social networking's new global footprint, March 2009.

[33] J. Rennie and A. McCallum. Using reinforcement learning to spider the web efficiently. In *Proc. ICML'99*, pages 335–343.

[34] P. Rusmevichientong, D. M. Pennock, S. Lawrence, and L. C. Giles. Methods for sampling pages uniformly from the world wide web. In *AAAI Fall Symposium on Using Uncertainty Within Computation*, pages 121–128, 2001.

[35] U. Schonfeld and N. Shivakumar. Sitemaps: above and beyond the crawl of duty. In *WWW '09: Proceedings of the 18th international conference on World wide web*, pages 991–1000, New York, NY, USA, 2009. ACM.

[36] P. Singla and M. Richardson. Yes, there is a correlation: - from social networks to personal behavior on the web. In *Proc. WWW'08*, pages 655–664.

[37] Web stats from Statbrain.com. `http://www.statbrain.com`.

[38] R. T. F. Tim Berners-Lee and L. Masinter. Uniform resource identifiers (uri): Generic syntax. In *Internet RFC 3986*, 2005.

[39] J. S. Vitter. Random sampling with a reservoir. *ACM Trans. Math. Softw.*, 11(1):37–57, 1985.

[40] Y. Wang, J.-M. Yang, W. Lai, R. Cai, L. Zhang, and W.-Y. Ma. Exploring traversal strategy for web forum crawling. In *Proc. SIGIR'08*, pages 459–466.

[41] C. Xiao, W. Wang, X. Lin, and J. X. Yu. Efficient similarity joins for near duplicate detection. In *Proc. WWW'08*, pages 131–140.

[42] Q. Xu and W. Zuo. First-order focused crawling. In *Proc. WWW'07*, pages 1159–1160.

[43] M. Yannakakis. Node-and edge-deletion np-complete problems. In *Proc. STOC'78*, pages 253–264.