# Mining the Web for IP Address Geolocations

Chen Chen
Chuanxiong Guo
Yunxin Liu
Helen J. Wang
Qing Yu
Yongguang Zhang
{v-chech, chguo, yunliu, helenw, qingyu, ygz} @ microsoft.com

October, 2007

Technical Report
MSR-TR-2007-139

**Microsoft Research-Asia**
Beijing Sigma Center
Zhichun Road
Beijing 100080, P.R. China

# Mining the Web for IP Address Geolocations

Chen Chen, Chuanxiong Guo, Yunxin Liu, Helen J. Wang+, Qing Yu, Yongguang Zhang
Microsoft Research-Asia, +Microsoft Research-Redmond
{v-chech, chguo, yunliu, helenw, qingyu, ygz}@microsoft.com

## ABSTRACT

In this paper, we observe that many Web pages contain geolocation information (address, zipcode, and telephone area code) and many of these geolocation items are directly related to the locations of the IP addresses that host the Web pages. We then design *Structon*, a system that mines Web pages for IP address geolocations. In Structon, we first extract geolocation information from every crawled Web pages, we then devise a serial of information clustering, false-information filtering, error-correction, and location inferring algorithms to map IP addresses to geolocations. We have run our algorithms on top of a set of 74M Chinese Web pages, from which we are able to identify the geolocations for 8.2M IP addresses, which contain addresses for not only Web servers but also client hosts. We have verified our result with an IP address location table of a major Chinese ISP, the verification shows that the accuracy of Structon is 94.4% at province level.

## 1. INTRODUCTION

The Internet depends on the Internet Protocol (IP) for information dissemination. IP address is one of the most important parts of the Internet Protocol, which is used not only for Internet routing, but also for end hosts identification. When the geolocaitons of IP addresses are known, many interesting location-aware applications can be provided. We give some examples as listed below:

- Automatic content providing. If a Web server knows the location of a customer, it can automatically provide contents that meet the customer's needs (e.g., local weather forecasting and location-aware advertisements).
- Resource ranking for Web search. The resources that are physically near the client should be assigned a higher score than those similar ones that are faraway from the client.
- Location-aware P2P overlay construction. In P2P file sharing and multimedia streaming, P2P nodes can take advantage of location information to select logical neighbors that are also their geographical neighbors. Location awareness can greatly improve user experiences (higher available bandwidth and reduced download time) and reduce P2P traffic in the network core.

Many schemes have been proposed to map IP addresses to geolocaitons, with their own pros and cons. See Section 5 for detailed discussion of these schemes.

In this paper, we propose a novel approach, which we call *Structon*, that mines the Web for IP address to geolocation mapping. We observe that many organizations put their contact information, which include many geolocation items, such as address, zipcode, and telephone area code, in their Web pages. We conjecture that the location of the organization is (statistically) related to the location of the Web server that hosts the Web pages. The reason is as follows. When one setups a Web server, she can setup the server inside her organization or she needs to use certain Web hosting service. Large organizations such as universities and governments may choose the first approach, in which the location of the Web server is exactly the location of the organization. When a user chooses the second approach, we argue that she still tends to choose Web hosting providers that are near to her, since 1) to place the Web server near the organization makes management and maintenance much easier; 2) people are more familiar with their local service providers than the remote ones. The results of this paper have validated this conjecture.

Motivated by the above observation and reasoning, in Structon, we first mine a Web data archive (includes 74M Chinese Web pages, which were crawled by the Web Search and Mining group of Microsoft Research Asia in the end of 2006) to extract location items (address, zipcode, and telephone area code) from each Web page; we then design a serial of information clustering, false-information filtering, error-correction, and location inferring algorithms to map IP segments to geolocations.

Using this 74M Web page pool, Structon is able to map IP address segments to {country, province, city} for 8.2M IP addresses in China. Structon covers 10% IP addresses in China by using less than 5% Web pages. Due to our information clustering and location inferring algorithms, Structon is able to identify locations for not only Web servers but also client hosts. We have verified our result with an IP address location table of a major Chinese ISP (which contains province information of its IP segments). The accuracy of Structon at province level is 94.4%. We also have compared our result with that of www.ip.cn, a grassroot site that manually collects IP address locations in China. The coherent ratio of Structon and www.ip.cn at city level is 86.1%.

To the best of our knowledge, Structon is the first approach that takes advantage of Web mining for IP address geolocation. Structon has the following properties: The data sources it uses are in public domain and may cover the whole world; the whole process is automatic without human intervene; and the result is of high accuracy.

| | |
|---|---|
| Number of Web pages | 74,300,723 |
| Web pages with location info | 20,590,612 |
| DNS names | 1,398,585 |
| DNS names with location info | 549,437 |
| Mined IP addr with location info | 96820 |
| Deduced IP addr with location info | 8,182,016 |
| Accuracy ratio (province level) | 94.4% |

**Table 1: The results we get after mined the Web data archive.**

The rest of the paper is organized as follows. We introduce the Web data extracting platform and the location extraction procedure in Section 2. We then devise algorithms for location information processing and inferring in Section 3. We validate the accuracy of Structon and briefly discuss its properties in Section 4. We discuss related work in Section 5 and conclude the paper in Section 6.

# 2. GEOLOCATION INFORMATION EXTRACTION

The Web data set we use has 74 millions Web pages (mainly from China) with a total size over 2.2 Tera-byes. We employ a two rounds approach to implement Structon. In the first round (Section 2), we extract location information (if any) for every archived Web pages on top of a specially designed cluster platform. In the second round (Section 3), we perform further information clustering, false information filtering, error-correction and location inferring, and finally map IP address segments to their geolocations.

A brief summary of Structon's results is given in Table 1.

## 2.1 The Platform

Geolocation extraction is performed on a cluster of 29 computers. Each of the 29 computers has a dual-core Intel 2.3 GHz Core2 processor, 4 GB DRAM, 1 TB hard disk, and a Broadcom 1 Gb/s Ethernet NIC. All the computers run Windows Server 2003 Enterprise x64 edition SP1 and are connected by a switch.

The structure of the platform is shown in Fig. 1. A distributed storage system is used to store and manage the crawled Web pages. The distributed storage system has similar properties with the Google File system [3]. It breaks large files into small pieces that are replicated and distributed across the local disks of the cluster.

On top of the distributed storage system, the Dryad [8] distributed execution engine is used to build the Structon routines. Dryad is a general-purpose, high performance distributed execution engine for coarse-grain data-parallel applications. With Dryad, developers are able to focus on their data processing logic while the Dryad execution engine handles many of the difficult issues of a large distributed, concurrent application: resources scheduling, concurrency optimization, communication and computer failures recovery , and data output. In Dryad, a task is scheduled to run at the computer which is the nearest to the stored data. This locality-awareness optimizes the network bandwidth usage and significantly increases the performance of the system.

## 2.2 Location Extraction

The geolocation extraction procedure is designed to extract location information from every crawled Web pages.

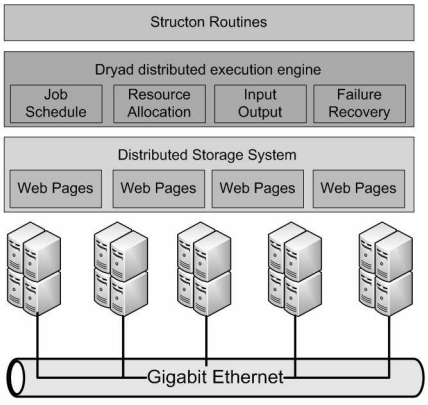


**Figure 1: The distributed system for location information extraction**

It is built on top of Dryad to take advantage of its parallel processing ability. The procedure is illustrated in Fig. 2. It uses MSNParser to parse each Web page and GRETA [4], an regular expression C$^{++}$ library, for location extraction.

**LocationExtraction**:

```
1  for (each url) { /* each url represents a Web page */
2     if (filterPage(url) == TRUE)
3        continue;
4     splitPage( );
5     for (int i = n; i > 0; i − −) {
6        addr = extractAddr(chunk[i]);
7        if (addr) listAddr.append(addr, i, n);
8        areacode = extractAreaCode(chunk[i]);
9        if (areacode) listArea.append(areacode, i, n);
10       zipcode = extractZipCode(chunk[i]);
11       if (zipcode) listZip.append(zipcode, i, n);
12    }
13    if (filterLocation(listAddr, listArea, listZip)==TRUE)
14       continue;
15    locations = (listAddr, listArea, listZip);
16    output(url, locations)
17 }
```

**Figure 2: The procedure to extract location information from all the archived pages.**

In Fig. 2, we first call *filterPage* to filter pages that contain 'blog', 'bbs', 'forum' in their urls. This is because the location information contained in these pages are very likely not the location of hosting Web server.

We then use *splitPage*, which in turn calls the MSNParser library, to divide the page into a serial of "chunks". A chunk here is approximately the content that contains in one html tag (we may combine several chunks to one to make sure that we do not loss location information). We then extract addresses, zipcodes, and telephone area codes from chunks by using *extractAddr*, *extractAreaCode*, and *extractZipCode*.

We observe that most of the location information start with "Addr:", "Tel:", "Fax:", and "Zipcode" and their similar variations. We therefore use these prefixes to simplify the location extraction algorithm.

In *extractAddr*, we first check if the content starts with "Addr:" or its variants (such as "Address:" and "Contact Addr:", etc.). If it does start with the desired prefixes,

we then try to extract locations by using regular expressions. We have established a small database that contains the province and city names, zipcodes and telephone area codes of China. We have 31 provinces (not including Taiwan, Hongkong, and Macao), 508 cities, 486 zipcodes, and 340 telephone area codes. In Chinese tradition, an address is start from province, then city, then street and building number. We take advantage of this to filter false-information. For example, in "Addr: Jiangsu province, Nanjing, Tibet road, No. 15", there are three locations: "Jiangsu" which is a province of China, "Nanjing" which is a city of Jiangsu, and "Tibet" which is a road name (but it is also a province of China). Since the location of "Tibet" appears behind "Nanjing" and "Jiangsu", we can safely filter "Tibet" in this case. (Similar rule can be applied to Western style Web pages, though the meaning of position needs to be re-interpreted.)

In *extractAreaCode*, if the content starts with "Tel:" or "Fax:" or their variations, we then try to match the telephone number using regular expressions that match the formats of a telephone number (we have designed 10 regular expressions to describe most of the telephone number formats).

Similar with *extractAreaCode*, *extractZipCode* extracts zipcode from the content.

After extracting all the locations from a page, we call *filterLocation*. We filter the page when the number of items in *listAddr*, or *listArea*, or *listZip* is larger than a threshold (10 in this paper). The reason is that when a page contains many address (zipcode, area code) items, it is very likely to be a Yellow page. The location information contained in Yellow pages are for dissemination or for advertising, and generally cannot be trusted for our purpose.

This location extraction algorithm is run on the cluster on top of Dryad. Dryad outputs the *(url, locations)* pairs into our local disks. All the rest computations described in Section 3 are performed on our local computer. The major challenge in designing the location extraction algorithm is the tradeoff between efficiency and accuracy: the algorithm should run fast enough (since we have Tera-Bytes of data to analyze) and the extracted location information should be accurate enough. We have tuned the extraction procedure so that it can be finished in 4 hours and 17 minutes in the cluster. The rest computations are not time-critical and are performed on a local PC.

# 3. GEOLOCATION INFORMATION PROCESSING

In this paper, We use the CIDR notation A.B.C.D/n to denote IP segments, where /n denotes the number of bits of the network prefix. Class C segments are therefore denoted as /24.

## 3.1 IP Segment to Location Mapping

The principle we adhere in this subsection is a majority voting principle: A segment is said to be in a location when most of the IP addresses in a segment say so. We take the following four sequential steps to finally map IP segments to geolocations:

1. For each page that is output from the previous location extraction procedure, we assign weights to locations based on their appeared position in the page.

| url \ location | BJ | FJ | LN | SH |
|---|---|---|---|---|
| dns_a/sub_url1 | 0.64 | - | 0.96 | 0.89 |
| dns_a/sub_url2 | 0.64 | - | 0.95 | 0.89 |
| dns_a/sub_url3 | - | 0.57 | 0.95 | 0.86 |
| LWV of dns_a | 0.43 | 0.19 | 0.95 | 0.88 |

**Table 2: Calculating the location weight vector (LWV) of a DNS name. The urls in the table share the same DNS name.**

2. For a DNS name that hosts many pages, we calculate a location weight vector from weights of the hosted pages.

3. We then resolve a DNS name to IP addresses (one DNS name may resolve to many IP addresses). All the IP addresses that are in the same class C IP segment are considered as only one independent IP address. All the independent IP addresses are then assigned the location weight vector of that DNS name.

4. We then cluster all the IP addresses that are in the same class C IP segment together and calculate a location probability distribution. The location of the class C IP segment is chosen as the one with the highest probability from the distribution.

In the first step, We use Fig. 3 to assign weights to locations for each page.

**WeightAssignment**:
```
1  for (each (url, locations) pair) {
2     addr = locations.listAddr.head;
3     area = locations.listArea.head;
4     zip = locations.listZip.head;
5     addr.weight = addr.chunk_id/n;
6        /*where n is the number of chunks of the page.*/
7     area.weight = area.chunk_id/n;
8     zip.weight = zip.chunk_id/n;
9     output (url, addr, area, zip);
10 }
```

**Figure 3: The procedure to assign weights to up to three locations in one page.**

In Fig. 3, we take the first items from *listAddr*, *listArea*, and *listZip*, respectively. These items are the last address, area code, and zipcode that appear in one Web page. The reason we choose the items appear in the end of a page is that users tend to put their contact information at the bottom of their Web pages. We then assign weights to these three geolocation items based on their positions in the page. The larger their chunk_id, the higher the weight. Note that *addr*, *area*, and *zip* may describe a same location. In this case, the weight of that location is the sum of the three weights. If the three locations are not the same, then the weight of the page is distributed to different locations. After the first step, for each url, we get at most 3 locations and their corresponding weights.

In the second step, we list all the pages that have the same DNS name into a same table to calculate a Location Weight Vector (LWV). We use the example as illustrated in Table 2 to illustrate how this step works. In this example, all the three urls share the same DNS name *dns_a*.

| IP \ location | Chengdu | NJ | Sanya | SH | SY |
|---|---|---|---|---|---|
| 61.155.111.42 | 0.003 | 0.004 | 0.003 | 0.24 | - |
| 61.155.111.44 | - | 0.02 | - | - | - |
| 61.155.111.70 | - | 0.77 | - | - | 0.13 |
| Location PDF | 0.26% | 68% | 0.26% | 20.5% | 11% |

**Table 3: Location weight vectors of IP addresses in the same 61.155.111.0/24 segment and the location probability distribution function (PDF) of the segment.**

As to the example in Table 2, the mean weights of dns_a that assigned to BeiJing (BJ), FuJian (FJ), LiaoNing (LN), and ShangHai (SH) are $(0.64 + 0.64)/3$, $(0.57)/3$, $(0.96 + 0.95 + 0.95)/3$, and $(0.89 + 0.89 + 0.86)/3$, respectively. The location weight vector is therefore {0.43, 0.19, 0.95, 0.88} for these four geolocations. After that, we then get a list of {dns name, location weight vector} for all the DNS names that appear in the Web archive.

In the third step, we first resolve DNS names into IP addresses. One DNS name may be resolved to multiple IP addresses. We treat all the IP addresses that are in the same Class C IP segment as one independent IP address (this is because a Web site may use multiple servers for reliability and load-balancing purposes, IP addresses in the same segment of a DNS name should not increase the weights of the site). For each independent IP address, an IP address to location weight vector mapping is then created.

In this stage, we also carry out IP address filtering by leveraging information from BGP routing table. From the BGP table [15], we get the origin Autonomous Systems (AS) number of an IP address. And from the Whois [2] database, we get from which country the AS number is registered. Then if the mined location of an IP address says that it is located in a country $X$, whereas the BGP table tells that this IP address is actually located in country $Y$, we can safely discard this mined location information for this IP address.

IP addresses are allocated in segments (to reduce the size of the routing table). The IP addresses of an allocated segment are in the same location. In the fourth step, we arrange the IP addresses that are in a same IP segment into a table as illustrated in Table 3, and then calculate a location probability distribution function for the segment. The problem here is how to decide the segment size. In this paper, we found /24 (i.e., class C) is a good (and conservative) choice at least for our dataset. We note that we actually can dynamically adjust the segment granularity. For example, when the majority IP addresses in the bottom half of a /24 segment say they are located in $X$, whereas the IP addresses in the top half say they are in $Y$, we can divide this segment into two /25 segments.

We calculate the location probability distribution function (PDF) of the IP segment by normalizing the location weight vectors of all the IP addresses in the segment. As to the example in Table 3, the probabilities that the segment 61.155.111.0/24 is located in CD (Chengdu), NJ (Nanjing), Sanya, SH (Shanghai), and SY (Shenyang) are 0.003/1.17, 0.794/1.17=0.68, 0.003/1.17, 0.24/1.17, and 0.13/1.17, respectively.

After that, we take the location that has the highest probability as the location of the IP segment. In this example, we decide that 61.155.111.0/24 is in NJ (which is exactly the case). We therefore map all the IP segments (that appeared in our Web data archive and have location information in Web servers) into their locations.

## 3.2 Self-Error-Correction and Location Inferring

The algorithms presented below show Structon's self-error-correction and location inferring abilities. More sophisticated algorithms can be developed based on this majority voting idea.

For self-error-correction, we cluster the class C IP segments we get into larger segments (in this paper, we choose the size of a larger segment to be /18 each with 64 class C segments). If most of the class C segments are located in a same location $L_m$ and only a very small fraction of segments are in other locations. We then conclude that these small fraction of segments are also located in $L_m$. The procedure is illustrated in Fig. 4. $IPSegList$ is the ascendant sorted list of the mined class C segments that are in the same /18 segment. $IPSeg_b$ and $IPSeg_e$ are the first and last Segments in $IPSegList$, $N_a$ is the number of class C segments that appear in $IPSegList$. $N_m$ is the number of Class C segments that are located in $L_m$.

**ErrorCorrection**:
```
1  for (each /18 IP segment) {
2     if (IPSeg_b not in L_m or IPSeg_e not in L_m)
3        continue;
4     flag = 0;
5     if ( N_a ≥ 30 and N_m/N_a ≥ 0.8)
6        flag =1;
7     else if (N_a ≥ 20 and N_m/N_a ≥ 0.85)
8        flag =1;
9     else if (N_a ≥ 10 and N_m/N_a ≥ 0.9)
10       flag =1;
11    if (flag ==1)
12       map all segments in IPSegList to L_m;
13 }
```

**Figure 4: The self-error-correction procedure.**

We use different thresholds ($N_m/N_a$) for different $N_a$. The larger the $N_a$, the smaller the threshold. This is because we need to be more cautious when the data set is small. We are conservative in that we require both the beginning ($IPSeg_b$) and the end ($IPSeg_e$) segments to be located in $L_m$.

Table 4 shows 11 class C IP segments in 59.64.128/18. Since 10 segments except 59.64.136.0/24 say they are located in BJ, based on Fig. 4, we determine that 59.64.136.0/24 is in BJ instead of HEB (the capital of HLJ province).

Based on the self-error-correction procedure, we further devise a location inferring heuristic. We observe that when all segments in $IPSegList$ are in the same location, it is very likely that all the segments in [$IPSeg_b$ - $IPSeg_e$] are in that location. The inferring algorithm is the same as Fig. 4 except that we replace line 12 to: "map all segments in [$IPSeg_b$ - $IPSeg_e$] to $L_m$;".

Using this inferring heuristic, as to the example in Table 4, we deduce that all the 55 class C segments are in BJ. We therefore are able to deduce locations for 44 segments that we originally do not know their locations! Note that in location inferring, we again are conservative: When segments in $IPSegList$ agree on their location, we treat only the segments that are in $IPSeg_b - IPSeg_e$ instead of the whole 59.64.128.0/18 segments to be located in $L_m$.

| IP segment | location [→*corrected location*] |
|---|---|
| 59.64.128.0/24 | BJ |
| 59.64.133.0/24 | BJ |
| 59.64.136.0/24 | HEB-HLJ →*BJ* |
| 59.64.137.0/24 | BJ |
| 59.64.140.0/24 | BJ |
| 59.64.144.0/24 | BJ |
| 59.64.149.0/24 | BJ |
| 59.64.154.0/24 | BJ |
| 59.64.156.0/24 | BJ |
| 59.64.160.0/24 | BJ |
| 59.64.182.0/24 | BJ |

**Table 4: An example to show how self-error-correction and location inferring work.**

### 3.3 The Result

By mining the 74M Web pages, Structon identifies the locations for a set of 19264 class C IP segments (we call this set the 'original set') which spans from 58.16.32.0 to 222.248.238.0. Using the self-error-correction procedure, we are able to 'correct' the locations for 374 IP segments (a.k.a., the 'corrected set'). After the location inferring procedure, we get locations for 31961 IP segments with 12697 new ones that originally do not appear in the Web archives (a.k.a., the 'inferred set'). We therefore are able to identify the locations for 31961 segments (or 8.2M IP addresses). As to the end of 2006, the total IP addresses allocated to China is about 82M, and the number of Web pages in China is expected to be much larger than 1.6 billion (internal reference). We therefore are able to cover 10% of the IP addresses with less than 5% Web pages. We still do not know how the coverage ratio will increase as the numbers of Web pages and DNS names increase. At the time when this paper is written, we are preparing much larger Web data archives.

## 4. VALIDATION AND DISCUSSION

### 4.1 Validation

We first verify the accuracy of Structon by comparing our result with a set of IP segments which we know their exact province information (a.k.a., the 'test set'). This set of IP segments is from a major Chinese ISP and contains 50976 class C IP segments. The number of the overlapped segments of our original set and the test set is 3919. The overlapped segments are distributed across all the 31 provinces of China Mainland (which demonstrates the geolocation diversity of our validation). For these overlapped segments, Structon correctly identifies the locations for 3429 of them. The accuracy ratio is therefore 87.5%. After running the self-error-correction procedure, Structon is able to correctly identify the locations for 3525 of them, and the accuracy ratio raises to 90%. After location inferring, since more segments are added, the number of overlapped segments become to 7033. Structon correctly identify the locations of all the inferred segments (which means that our location inferring algorithm may be overly conservative and may have large room for further improvements) and the accuracy ratio raises to 94.4%.

We also have compared our result with www.ip.cn, a grass-root site that manually collects IP location information contributed by end users at city and province levels. At city level, the 'original set' has 17936 segments overlapped with www.ip.cn, with coherent ration 80.7% (i.e., 14473 segments are mapped to the same cities in both sources). The coherent ratios (overlapped segments) are 82.8% (17748) and 86.1% (22322) for the 'corrected' and 'inferred' sets, respectively. At province level, the coherent ratios (overlapped segments) are 87% (19206), 89% (19206), and 93.2% (31815) for the 'original', 'corrected', and 'inferred' sets, respectively. Though a coherent ratio cannot tell us the exact accuracy ratio, a high coherent ratio nonetheless indicates a high accuracy ratio.

The high accuracy of Structon is therefore validated by both of the two validation studies.

### 4.2 Discussion

Since Structon mines Web pages for IP address locations, one might doubt that Structon can only identify locations for Web servers. Since for many location-aware applications, locations of client hosts may be more useful, one may therefore doubt the usefulness of Structon. This observation, however, is not true due to:

- When the location of one IP address is identified, the location of the whole segment is also determined. And it is very unlikely that the whole segment contains only Web servers.
- Most importantly, Structon has the ability to infer locations for segments that do not host any Web servers. Our result shows that even very conservative inferring algorithm can discover significantly more IP segments (12697 segments). For example, Structon can identify 218.69.110.255/24 is in TianJin (TJ) via location inferring. An offline check shows that this IP segment is assigned to ADSL users.

At current stage, though we do not know whether Structon is able to cover the whole IP address space when all the Web pages of the world are available, Structon surely is able to cover many client IP addresses and to provide a huge pool of highly available passive landmarks with accurate location information for the whole networking community.

## 5. RELATED WORK

There are two categories of related work for geolocation mapping, one is delay-based, and the other is information-retrieval-based.

### 5.1 Delay-based

There are many schemes that first measure delays to landmarks, then calculate the geolocation (or virtual coordinates) of an IP address based on the measured delays from the end-host to the landmarks [5, 9, 10, 11, 14, 16, 17, 18]. GeoPing [14] maps a host to one of its landmarks based on the measured delays between the landmarks and the host. CBG [5] improves GeoPing by using the measured delays as constrains. The location derived from CBG need not to be the locations of the landmarks. TBG [9] further improves the delay-based approach by taking advantage of the topology information. In Octant [17], not only positive, but also negative measurement constrains are considered. Since the geographical distance and network delay is only moderately correlated (due to detour routing and queueing and transmission delays), delay-based approaches generally result in hundreds or even thousands of kilometers error distance.

There are also approaches that calculate Internet distance between end-hosts based on certain end-hosts coordinates [1, 13]. The basic operation used by these approaches is also to measure delays between end-hosts.

Structon can be complimentary to the delay-based approaches. The locations of the Web servers determined by Structon can be used by the delay-based approaches as passive landmarks. Since Structon can easily discover the locations of millions of Web servers, the number of landmarks used in delay measurement can be increased in many magnitudes. This will increase the accuracy of the delay-based approaches significantly, since "the error of the class of delay-based algorithms to be strongly determined by the distance to the nearest landmark" [9].

## 5.2 Information-retrieval-based

Structon is an information retrieval-based approach. Information retrieval-based approaches get geolocation information from certain sources that contain location information. In [12], the authors mined the Whois [2] database for geolocation. The major issue of using the Whois database is that the location information may be outdated or even incorrect.

In GeoCluster [14], the authors used the IP location information collected by a large Web portal. The location information were input by end users when they were asked to provide their location information by certain location-aware services (such as weather forecasting). The accuracy of GeoCluster therefore depends on the correctness of user input. Another difference between GeoCluster and Structon is that Structon uses publicly available Web pages instead of proprietary data sources.

As compared with previous information-retrieval-based approaches, Structon is more active in that by actively crawling the Web, it can detect location changes for existing IP segments and discover locations for new IP segments.

There are also many Web sites that provide IP location query service, such as [6, 7]. The technologies they use are commercial secrets, hence it is difficult to compare them with Structon. Their data sources may be, 1) from Whois database; 2) collected from ISPs; 3) collected using grassroot methods (e.g., establish a Web 2.0 site and let users input their IP addresses and locations). Collecting data from ISP does not scale since it is difficult if not totally impossible to work with all the ISPs in the world. The grassroot methods have 2 issues: 1) to attract people in the world to participate is quite difficult; 2) it is difficult to filter malicious input data.

# 6. CONCLUSION AND FUTURE WORK

In this paper, we have presented *Structon* for IP address to geolocation mapping by mining the Web. Structon is able to cover 8.2M IP addresses with 74M Web pages in China. The data sources used by Structon are all crawled from the public domain and the whole process is automatic without human intervene. By introducing a serial of information clustering, false information filtering, self-error-correction, and location inferring algorithms, Structon achieves high IP-to-location mapping accuracy at both province and city levels.

Structon is our first data-centric approach to show that the information contained in the Web can help us better understand the network infrastructure itself. In our future work, we plan to: 1) run Structon on larger Web dataset; 2) research on methods to cover more IP addresses for client hosts; 3) extend Structon to provide a network distance service: given two IP addresses, tell the network latency between them.

# 7. REFERENCES


[1] Frank Dabek, Russ Cox, Frans Kaashoek, and Robert Morris. Vivaldi: A Decentralized Network Coordinate System. In *Proceedings of SIGCOMM'04*, 2004.

[2] L. Daigle. Whois protocol specification, September 2004. RFC 3912.

[3] S. Ghemawat, H. Gobioff, and S. Leung. The Google file system. In *Proc. ACM SOSP'03*, 2003.

[4] The GRETA Regular Expression Template Archive. http://research.microsoft.com/projects/greta/.

[5] Bamba Gueye, Artur Ziviani, Mark Crovella, and Serge Fdida. Constraint-Based Geolocation of Internet Hosts. *IEEE/ACM trans. Networking*, 14(6), Dec 2006.

[6] IP2Location. http://www.ip2location.com/.

[7] IP Inquery. http://www.ip.cn.

[8] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks. In *Proc. ACM EuroSys'07*, Lisboa, Portugal, March 2007.

[9] Ethan Katz-Bassett, John P. John, Arvind Krishnamurthy, David Wetherall, Thomas Anderson, and Yatin Chawathe. Towards IP Geolocation using Delay and Topology Measurements. In *Proceedings of IMC'06*, 2006.

[10] Jonathan Ledlie, Paul Gardner, and Margo Seltzer. Network Coordinates in the Wild. In *Proceedings of NSDI 2007*, Cambridge, MA, April 2007.

[11] Harsha V. Madhyastha, Thomas Anderson, Arvind Krishnamurthy, Neil Spring, and Arun Venkataramani. A Structural Approach to Latency Prediction. In *Proceedings of IMC'06*, 2006.

[12] David Moore, Ram Periakaruppan, Jim Donohoe, and k claffy. Where in the World is netgeo.caida.org? In *Proceedings of INET'00*, 2000.

[13] T. S. Eugene Ng and Hui Zhang. Predicting Internet Network Distance with Coordinates-Based Approaches. In *Proceedings of infocom'02*, 2002.

[14] Venkata N. Padmanabhan and Lakshminarayanan Subramanian. An Investigation of Geographic Mapping Techniques for Internet Hosts. In *Proceedings of SIGCOMM'01*, 2001.

[15] University of Oregon Route Views Project. http://www.routeviews.org/.

[16] Liying Tang and Mark Crovella. Virtual Landmarks for the Internet. In *Proceedings of IMC'03*, 2003.

[17] Bernard Wong, Ivan Stoyanov, and Emin Gün Sirer. Octant: A Comprehensive Framework for the Geolocalization of Internet Hosts. In *Proceedings of NSDI 2007*, Cambridge, MA, April 2007.

[18] Artur Ziviani, Serge Fdida, José Ferreira de Rezende, and Otto Carlos Muniz Bandeira Duarte. Improving the accuracy of measurement-based geographic location of Internet hosts. *Computer Networks, Elsevier Science*, 47(4):503–523, March 2005.