

# PAIR: Person-Attribute Identification and Retrieval

Yi-Ting Chou, Shui-Lung Chuang, Xuanhui Wang

Computer Science Department  
University of Illinois at Urbana-Champaign  
201 N. Goodwin Avenue, Urbana, IL 61801  
{chou1,schuang2,xwang20}@uiuc.edu

## Abstract

The Web has become the most major information source of our daily activities. In this project, we consider a special information need: finding the instance-attribute information from the Web, and particularly, we focus on the instance domain of person. Examples include the contact information (attribute) and research interests (attribute) of faculty or students (person). Most of such personal information already exists today as conventional, unstructured pages because people find it easier to create them that way. Such type of information need is very common but cannot be directly supported by current keyword-matching-based search engines. People get used to using a two-phase search scenario: locating the candidate pages first and then getting into them for the desired information piece. Based on the stimulation of such human search behavior, we design a block-based retrieval engine, upon the general search engines, to help the finding of person-attribute information from the Web. The experiment on several faculty members has shown the feasibility of the approach. It is believed that the approach can benefit not only the finding of such information by users, but also various extraction applications.

## 1 Introduction

The Web has become the most major information source of our daily activities. To help users finding information on the Web, search engines are developed, e.g., Google<sup>1</sup>. They constantly crawl Web pages, and provide the retrieval of relevant pages for large amounts of free text queries. Although such general search engines have reduced a lot of information overload for the users, there are still many special information needs not directly supported. In this project, we

are considering one of such special information needs: finding the instance-attribute information from the Web, that is, we want to get the value of an attribute about an instance. Examples of instance-attribute pair include the price of a book and the phone number of a person. Such type of information needs raise usually in our daily life, but cannot be directly answered by current search engines. To make our study focused, we particularly focus on the instance domain of person.

**Example 1:** Consider that we are going to construct a profile list for UIUC CS faculty. A profile would consist of several fields (or attributes), e.g., name, title, email, postal address, research interests, publication list, etc. Such people profiles would be very useful in many aspects. For example, we can enhance search capability. A prospective student may look for faculty in a particular research field, or ask the detailed contact information about a specific faculty without looking at his/her homepage. We may also perform some mining tasks to discover the relationship among the faculty, e.g., who join the same project. ■

Filling such profiles needs some labor-intensive work, such as manually collecting the information faculty-by-faculty. This is an annoying job. It is still tractable when dealing with UIUC CS faculty because of the locality of faculty. However, such a task becomes harder when considering the profile construction for committee members in a conference, since they may distribute over the world. It would be beneficial to have a tool that can assist the profile constructor.

Most of such personal information already exists today as conventional, unstructured pages because people find it easier to create them that way. Some information may also be duplicated in several places. For example, publications of a researcher may be listed in his/her homepage, and also available in some public resources, such as DBLP, ACM, and IEEE digital libraries. Hence, to fill a specific person-attribute slot, e.g., the school (attribute) that Prof. ChengXiang Zhai (person) received his Ph.D. degree, the profile constructor can utilize current search engines. First, he/she may use search engines to locate the pages that

<sup>1</sup><http://www.google.com>

may contain the target information (usually the homepage of the target person), and then look into the candidate pages to find out the desired information (usually located in some passage of the pages). This two-phase process (i.e., locating the candidate pages and getting into them for the desired information piece) seems becoming the major search behavior of users using the general search engines.

To further reduce the search efforts, it would be beneficial to have a search engine that can directly responds to some questions whose answers are composed of short messages. Web information extraction (IE) aims to identify and extract named entities and their relations from the Web. However, there are tremendous domains of instances. Even when we focus on the domain of person, there are still many types of attributes. Each type of attribute requires different kind of extraction method (think of the difference between identifying a person’s address and graduating school), and most of them are still not mature and need a lot of further research efforts. Hence, following the traditional Web IE approach to constructing a person-attribute database is hard to be scalable and hence, suffers from the inability to answer huge variety of requests. (Analogy examples include AskJeeves which can not answer most of questions because of the shortage of the underlying knowledge base.)

Our idea is to translate the extraction paradigm of such tasks into a retrieval framework. Current search engines retrieve the information in page level, and return a list, each of whose items contains a page url and a short summary (usually the surrounding text of the matched keywords). Because of their convenient usage and irreplaceable ability to search huge amounts of pages, users get used to using them with evolved two-phase search behavior. Since the users’ demands are widely various, it is hard to design a system that can directly satisfy all kinds of search needs or tasks. It is the generality and simplicity of usage that make the general search engines so successful to be a general tool for users to perform their tasks based on them. Hence, we are considering the design of a system that can perform the retrieval of instance-attribute information from the Web without relying on the complicated extraction. Such a system would also benefit many Web IE and NLP systems that are going to use the Web as the corpus for their tasks.

Our design evolves from two observations: (1) Most of the information is contained in some text segment inside the page. Instead of performing the retrieval in page level, can we perform it in a more fine-grained regularity level? (2) Before looking into a page, users usually have a quick look at the matched snippet of the page in the returned result list so as to justify the relevance of a page at the very early stage. Can this matched snippet be replaced by some more accurate information, which directly answer or is closer to our

desired target attribute?

Stimulated from the above two observations, we design an engine, named PAIR, based on the human’s two-phase searching behavior. PAIR is built upon the current search engines and use them as the backend to perform the page retrieval. (In our system, we use Google.) First, PAIR uses the given person name to retrieve several candidate pages. Then, it segments each page into blocks with different level of granularity. Further, it uses the given attribute to re-rank the text blocks. Finally, it returns a ranked list of items, composed of the page urls and high-ranked blocks.

In the rest of this report, we first provide an overview of the proposed approach. Then, the design of each involved component is presented in detailed, followed by the experiments and some statistical analyses on the results. Finally, we review some related works and draw the conclusions.

## 2 Overview of the System

In this section, we provide a brief overview of our system. Some of the technical details of the components will be described in the following sections.

Towards our goal of person-attribute retrieval, we identify the following challenges:

1. The information of a person may be distributed in several places on the Web. How to identify those most-relevant pages containing the target information is a problem to be addressed.
2. A page usually contains various kinds of information, such as the contact information and biography in the homepage, and they are not organized in a structured formats. The difficulty in identifying the target information is twofold.
  - How to segment the page into basic semantic blocks?
  - How to identify which block most likely contains the target information?

Figure 1 depicts the operation of our framework. First, our system accept two string inputs: a person name and an attribute name. For example, the person name is “ChengXiang Zhai” and the attribute name is “graduating school.” Our system performs the following steps.

1. We use the person name as a query to the backend search engine, and collect top-ranked pages as the candidates that contain the information of our target attribute.
2. The information in the Web pages is usually organized as blocks to facilitate user browsing. By applying existing block analysis techniques, such as Vision-based Page Segmentation (VIPS) [6], we decompose each page into a hierarchy of blocks,

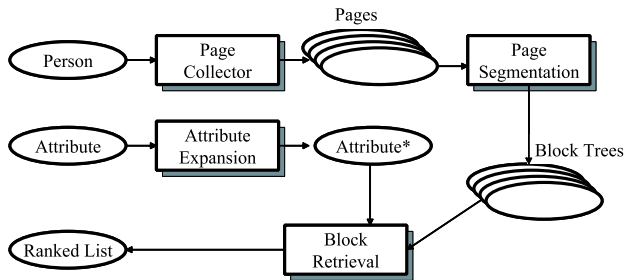


Figure 1: A diagram showing the operations behind the proposed approach.

from coarse to fine-grained blocks. Each of them are treated as an complete semantic unit (please refer to Figure 2 for a rough idea of the document block-tree).

- Since an attribute name is usually too short and may not be effective to retrieve the target information, we perform a blind query expansion on the given attribute name. The attribute name is sent to the backend search engine, and all the snippets in the result page are used as the complement text to enrich the representation of the attribute.
- Finally, we use the augmented attribute to compute a relevance score on each node in the block trees of documents through a *tf-idf* weighting scheme. The blocks with high scores are selected and returned as a ranked list to the users.

Step 1 addresses our challenge 1, and Steps 2–4 address our challenge 2. Through the above steps, we can obtain a list of blocks ranked according to the possibility that they contain the information for the target attribute. It would be useful for further processing, such as applying NLP techniques to identify noun phrases or any specific entity, e.g., the address. But, here we only consider providing a block-retrieval engine to this instance-attribute search problem.

### 3 Technical Details

From the diagram shown in Figure 1, there are several components in our approach. We describe each of them below with more details.

#### 3.1 Page collector

The purpose of the page collector is to gather the related pages about the given person name. In our current design, it performs the following steps:

- The person name is submitted to the backend search engine to retrieve the top-ranked pages, which are treated as the initial page set for further processing.

- As the information is usually organized in a hyperlinked structure on the Web, the target information may locate in some sub-hierarchy pages. Hence, for each candidate page, we also collect its out-linked pages for further processing.

Since identifying whether a page is really relevant to a person is non-trivial, we decide to rely on current search engines. A given person name is usually more unique and specific, and current search engines are good in performing such kind of yellow-page-based search (e.g., finding the homepage of a named person, a company, or an institute). We decide to trust those top-ranked pages as the most possible pages that contain the target information of that person. Hence, further processing is based on these top-ranked pages. This simplifies the design of our current system. Our further work will consider relaxing this assumption.

#### 3.2 Visual-based Page Segmentation (VIPS)

The target information in our project is not a whole web page but a semantic coherent text snippet. Thus our method employs page segmentation algorithms, and we adopt the Visual-based Page Segmentation (VIPS) proposed by Cai et al [6].

VIPS aims to segment a web page into semantic structure based on its visual presentation. Such semantic structure is a tree structure; each node in the tree corresponds to a block. Each block will be assigned a value (Degree of Coherence) to indicate how coherent of the content in the block based on visual perception, the bigger is the DoC value, the more coherent is the block. The algorithm takes a top-down segmentation procedure to build the tree structure. The depth of the tree is controlled by a predefined DoC value. A typical way is to use all the leaf blocks as the segmentation results.

The VIPS algorithm could segment the page into meaningful blocks. However, the granularity is controlled by the predefined DoC value. If the predefined value is too large, the leaf block is too small to contain meaningful information. On the other hand, if the value is too small, the leaf block may be too large and the information in it is not so coherent. To predefine an appropriate DoC for all the pages is very difficult because of the diverse structures of the web pages.

Instead of using the leaf blocks, we consider the hierarchy structure to capture the containment relationship between blocks with different granularities. Figure 2 show an example. The root corresponds to our target person. Each of the second-level nodes corresponds to a page related to the person, and its sub-hierarchy is the result by applying VIPS on the page. In our project, we predefine a larger DoC value so as to get a fine-grained segmentation and select the appropriate blocks from the whole tree (including both leaf nodes and non-leaf nodes).

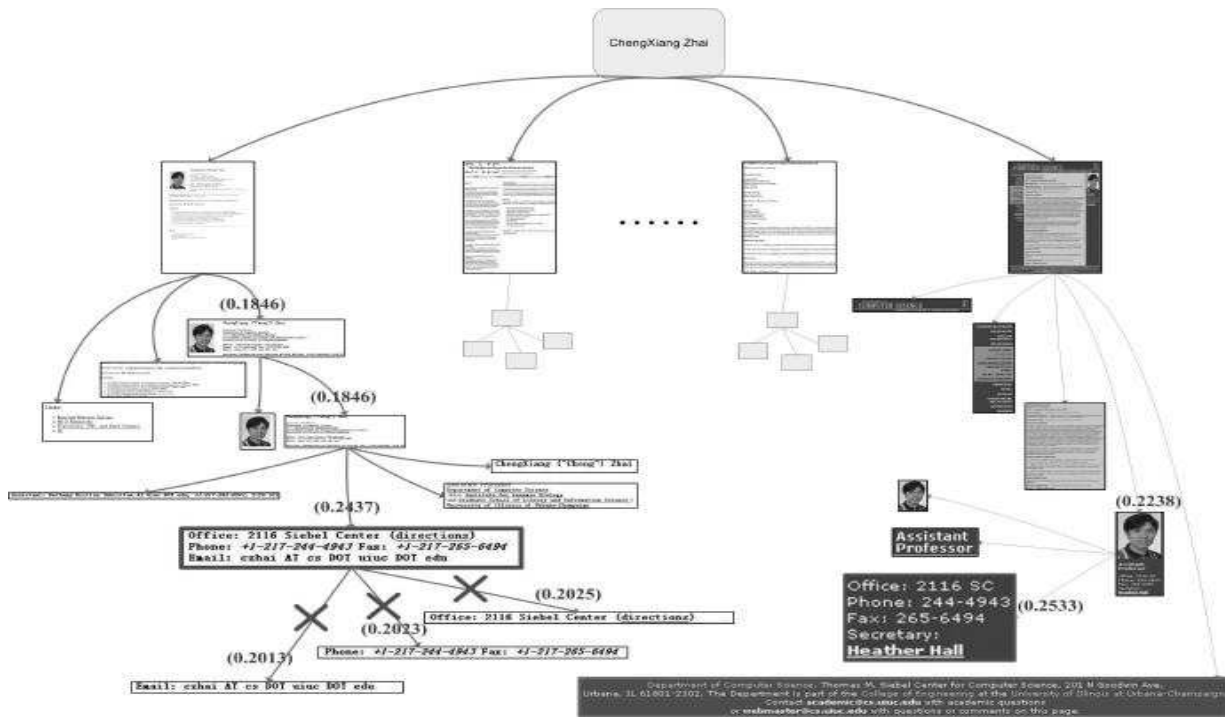


Figure 2: An example block-tree of pages for a person.

There are other approach to getting a hierarchical structure of a Web page. The reason we choose VIPS as our segmentation is onefold. Although the HTML language is simple, the composition of a Web page is usually very complicated. There are many different ways to format a page, from the use of the basic tags (e.g., table related tags are overwhelming used in convention) to the use of the Cascading Style Sheets (CSS). This increases the challenge of analyzing page structure through directly analyzing its source HTML. Hence, through vision-based approach, we can get a more consistent segmentation of the page.

### 3.3 Attribute expansion

An attribute name is usually too short and may not be effective to retrieve the target information. The block containing the target information may not contain the attribute name in its content. For example, the contact information shown in a personal homepage may just contain the address, phone number, and email address (all of these are easily understandable for human readers) without an indicating head “contacts.” Or, the indicating head may be some equivalent expression (such as synonym) of the given attribute: For example, the page author may use “way to reach me” instead of “contacts.” Thus, using only the given attribute name to adjust the relevance of the candidate blocks may be infeasible.

An ideal approach to addressing this problem is to gather some example text blocks that contain the target attribute information (other people’s), and then

to learn a concept of that attribute. For example, if we have some sample blocks of contact information from other people, we can learn that it usually contains some indicating words, such as “phone” and “email”, and some common words used in the value of attribute, such as “street,” “avenue”, etc. However, there does not exist any search engine that can retrieve the samples of various attributes.

Hence, we use an approximate solution. For the given attribute name, we perform a blind query expansion on it. The attribute name is sent to the back-end search engine, and all the snippets in the result page are used as the complement text to enrich the representation of the attribute. Sometimes, the snippets contain some fragments of the attribute sample from other people. From these fragments, we still can learn a partial concept of the attribute. For example, the attribute “contacts” can be enriched with “office,” “phone,” “fax,” “email,” or some address snippets “st.,” “ave.,” which would identify the block containing contact information better. In our current implementation, we use the top 100 snippets from the Google search engine.

### 3.4 Block retrieval

After the page segmentation and attribute expansion, we have a block hierarchy about the pages related to the person and the expanded text for the attribute. The remaining task is to determine the relevance between the blocks and the expanded attribute. First, we traverse the block hierarchy. At each node, the text

contained in that block is compared with the expanded attribute to determine a relevance score. Second, we determine the appropriate blocks to retrieve. If the score of a node is higher than the score of every its child, then its children are pruned. For example, referring to Figure 2, for the attribute “contacts,” the leaf blocks containing individual information about office, email, and phone are pruned and their parent block remains. This depends on the target attribute. If the attribute is “phone number,” then the leaf block may survive because the parent block contains more irrelevant information. After this pruning stage, the remaining leaf blocks in the hierarchy form the basic elements to retrieve, and are put into a ranked output list according to their scores.

The core of this process is how to compute a relevance score between a node and the attribute. Here, we adopt the vector-space model for this purpose. The text (either of the block or of the expanded attribute) is converted into a bag of feature terms by applying normal text processing techniques, e.g., removing stop words<sup>2</sup> and stemming. Some easily recognized patterns, i.e., email, phone number, and date, are also added into the feature set if the text contains such pattern. For example, if the text contains an email address (e.g., user@dept.univ.edu) or a phone number (e.g., (217) 444-3333), the higher-level feature about email or phone will be added. These patterns are expressed by regular expressions, and incorporated into the text processing step. With simple processing, a text  $b$  can be represented as a term vector  $v_b$ , where  $v_{b,t}$  is the weight term  $t$  in  $v_b$ . The term weights in this work are determined according to one of the conventional *tf-idf* term weighting schemes [11], in which each term weight  $v_{b,t}$  is defined as

$$v_{b,t} = (1 + \log_2 f_{b,t}) \times \log_2(n/n_t), \quad (1)$$

where  $f_{b,t}$  is the frequency  $t_i$  occurring in  $v_b$ 's corresponding feature term bag,  $n$  is the total number of leaf blocks in the original block hierarchy, and  $n_t$  is the number of leaf blocks that contain  $t$  in their corresponding bags of feature terms. The similarity between a block  $b$  and the attribute  $a$  is computed as the cosine of the angle between the corresponding vectors, i.e.,

$$\text{sim}(a, b) = \cos(v_a, v_b).$$

In addition to this basic similarity measure, we also consider several heuristics to enhance the relevance measure:

#### *Background noise.*

Since we blindly adopt all snippets to augment the attribute, a lot of irrelevant terms will be added into the attribute's feature set. Intuitively, this will make

<sup>2</sup>In this work, stop words are determined by a stop-word list, and the one we used is obtained from the Smart system, available at <ftp://ftp.cs.cornell.edu/pub/smart/>.

our similarity measure unreliable. Hence, we consider building a background model to capture these common used terms.

To obtain this background model, we submit 485 stop words in our stop-word list to the backend search engine and each of them achieve 100 snippets. This forms a text set of 48500 snippets. For each term  $t$  appearing in this set, we compute an *idf* value for it:

$$bg(t) = \log_2(m/m_t)$$

where  $m$  is the total number of snippets and  $m_t$  is the number of snippets containing term  $t$ .

To use this background model, the term weight in Equation 1 is changed as:

$$v'_{b,t} = (1 + \log_2 f_{b,t}) \times \log_2(n/n_t) \times bg(t).$$

Hence, if a term is commonly used in the Web environment (e.g., “search,” “information,” etc), its corresponding weight will be degraded. The similarity between a block  $b$  and the attribute  $a$  becomes

$$\text{sim}_{bg}(a, b) = \cos(v'_a, v'_b).$$

#### *Rank of the pages.*

In Google search engine, the ranks of the pages in the returned list implicitly represent the relevance of the pages to the query. Given a person name query, the page with small rank can be considered with good quality to answer our target information. Under this assumption, we consider the rank of the pages when select the text snippet (blocks). In our method, we define a decay function along with the ranks to weigh each page. The higher the rank is, the lower the weight is. In this way, the page at top is emphasized to avoiding the result being biased by other noise pages. Let the rank of the page containing the block  $b$  is  $r_b$ , our decay function is heuristically defined as:

$$\text{rank}(b) = 1 - (r_b - 1) \times \alpha,$$

where  $\alpha$  is a constant parameter for the decrement of importance of the block per rank increment. In our experiment, we set  $\alpha$  as 0.5.

To incorporate this heuristics, the similarity between a block  $b$  and the attribute  $a$  is

$$\text{sim}_{rank}(a, b) = \text{sim}(a, b) \times \text{rank}(b).$$

#### *Length of the block.*

The length factor is always significant for information retrieval [9]. When using cosine to measure the similarity, short documents are preferred than long documents by this measure. When we calculate the similarity between blocks and the query, we want to disfavor the too short documents because it always does not contain enough information. On the other hand, we

also do not want too big blocks which may contain irrelevant information. Thus, we introduce the heuristic to deal with the length problems. Let  $l_b$  be the length of the text contained in block  $b$ , we define a membership function to weigh different length:

$$\text{len}(b) = \begin{cases} l_b/200 & l_b < 200, \\ 1 & 200 \geq l_b \leq 500, \\ 0 & 500 < l_b. \end{cases}$$

To incorporate this heuristics, the similarity between a block  $b$  and the attribute  $a$  is

$$\text{sim}_{\text{len}}(a, b) = \text{sim}(a, b) \times \text{len}(b).$$

To incorporate all of the above heuristics, the similarity between a block  $b$  and the attribute  $a$  can be changed as:

$$\text{sim}_{\text{all}}(a, b) = \text{sim}_{\text{bg}}(a, b) \times \text{rank}(b) \times \text{len}(b).$$

In our experiment, we will give a comparison among these heuristics. But, we will not further explore whether the parameter setting for each individual heuristics is appropriate (e.g., the decay factor  $\alpha$  in the rank of pages and the membership function in the length of the block).

## 4 Experiments

There is no standard benchmark for this task. To evaluate our performance, we design to manually create several person and attribute testing pairs, and evaluate the top-ranked results of block retrieval by human.

### 4.1 Data set

We selected 20 faculty from UIUC CS department, and consider 3 attributes: (1) contacts, (2) graduating school, and (3) interests. For each person, we performed our approach on the top five pages returned from Google. For each person-attribute pair, we ran experiments on the following five heuristics strategies: (1) no heuristics (None), (2) the rank heuristics (Rank), (3) the background noise removal heuristics (Bg), (4) the length filter heuristics (Len), and (5) all heuristics (All=Rank+Bg+Len). The result set contains totally 300 ranked block lists. For each rank list, we manually record the rank of the desired block.

The Web may not contains every (attribute) information of a given person. Therefore, it is reasonable to treat such person-attribute pairs as outliers and excludes them from our sample. Moreover, since some algorithms might be significantly worse than the other in particular person-attribute pair, we set their rank to be the number of persons with valid results given that attribute to avoid over-penalizing effect from the outliers. Table 1 shows the top ranked desired block of different heuristic strategies for each person-attribute

pair. In Table 1, there are six major columns. The first major column contains the names of the testing faculty. The rest five major columns correspond to each heuristic strategy. Each of the rest five major columns contains three sub-columns, each of which corresponds to the result on one attribute. Notice that if the desired information is not found in the result, an N is recorded in the corresponding slot; otherwise, the rank of the target block is recorded.

### 4.2 Discussion on Attributes

In this part, we will discuss how the various attributes and heuristic strategies affect our result performance. In order to test these two factors without counting the interaction effect of these two factors, we design our experiment with the 2-way analysis of variance (ANOVA).

ANOVA is a statistical model to test the significance among several different factors simultaneously. In our experiment, the first factor is the heuristic strategies, and the second factor is the attribute. For each block, we have 10–20 samples, depending on the number of valid data. Table 2 is the summary of our test result.

We may notice that the p-value of attribute factor is less than 4%. This is because our attribute of query is not predefined, and every attribute has different property. So the result performance is highly correlated with the attribute.

### 4.3 Testing on Heuristics

The result in Table 2 shows that the p-value of Heuristic factor is about 40%, which means adding the heuristics is not statistically improving our performance. In this part, we would like to argue that different heuristic strategies will affect the result performance statistically significantly.

In 2-way ANOVA model, we assume  $Y_{ij} = \mu + \alpha_i + \varepsilon_{ij}$ , where  $Y_{ij}$  is the  $j$ th observation of the  $i$ th factor,  $\mu$  is the overall mean,  $\alpha_i$  is the differential effect of the  $i$ th factor, and  $\varepsilon_{ij}$  is the random error in the  $j$ th observation under the  $i$ th factor. The  $\varepsilon_{ij}$  in 2-way ANOVA is assumed to be independent, normally distributed with mean zero and fix variance.

However, given a person-attribute pair, every heuristic strategy will find desired blocks from the same set of pages. Therefore, it is not accurate to assume all  $\varepsilon_{ij}$  are independent from a common normal distribution. Moreover, the p-value of the interaction between two factors is high ( $> 88\%$ ), it is reasonable to ignore the effect of different attributes.

Here, we focus on the effect of each heuristic. We want to perform hypothesis testing between no-heuristic strategy and other four heuristic strategies. Since given a person-attribute, every strategy share the same set of pages, the samples from two distribution are paired. Therefore, we may model our samples

Table 1: The result on 20 faculty and 3 attributes with various heuristic strategies.

Name	Heuristic Strategy (Contacts/Graduating School/Interests)														
	None			Rank			Bg			Len			All		
AnHai Doan	2	5	4	2	5	6	1	3	1	2	5	2	1	2	1
ChengXiang Zhai	1	2	2	1	2	1	1	2	1	1	2	1	1	1	1
Dan Roth	2	1	N	2	1	N	1	1	N	1	1	N	1	1	N
Darko Marinov	1	10	N	1	12	N	1	13	N	1	7	N	1	10	N
Edgar Ramos	1	6	10	1	5	10	1	2	10	1	2	10	1	2	6
Eyal Amir	2	4	2	2	4	2	1	2	1	2	3	5	1	2	1
Haiyun Luo	1	3	N	1	4	N	1	3	N	1	2	N	1	3	N
Indranil Gupta	5	2	N	5	2	N	1	2	N	1	2	N	1	2	N
Jean Ponce	1	N	N	1	N	N	1	N	N	1	N	N	1	N	N
Jiawei Han	1	2	6	1	2	7	1	3	7	1	3	4	1	3	2
John Hart	1	N	10	1	N	10	1	N	4	2	N	5	1	N	3
Jose Meseguer	1	N	N	1	N	N	1	N	N	1	N	N	1	N	N
Josep Torrellas	1	N	N	1	N	N	1	N	N	1	N	N	1	N	N
Kevin Chang	3	1	2	3	1	2	1	1	4	1	1	1	1	1	6
Klara Nahrstedt	2	N	3	1	N	4	1	N	1	1	N	4	1	N	1
Marco Caccamo	1	N	N	1	N	N	1	N	N	1	N	N	1	N	N
Marianne Winslett	1	7	1	1	8	1	1	9	1	1	13	4	1	11	1
Michael Garland	4	1	N	4	1	N	4	1	N	2	1	N	2	1	N
Ralph Johnson	1	3	5	1	3	5	1	3	2	1	3	5	1	3	1
Steven LaValle	1	N	N	1	N	N	4	N	N	1	N	N	1	N	N

The number is the rank of target information in the result list;  
If the information of the desired person-attribute pair does not exist, an N is recorded.

Table 2: The ANOVA between the heuristic strategies and attributes.

Factor	d.f.	SS	F	p-value
Heuristic	4	31.3023	1.4641	0.3955
Attribute	2	264.6178	24.7540	0.03958
Interaction	8	26.1145	0.6107	0.8838
Error	200	1068.9884		
Total	214	1391.0232		

by  $D_i = X_i + Y_i$ , where  $X_i$  is  $i$ th sample from no-heuristic sample set, and  $Y_i$  is  $i$ th sample from the sample set of our target heuristic strategy. Instead of comparing the distribution of  $X_i$  and the distribution of  $Y_i$ , we compare the distribution of  $D_i$  with 0. By this way, we reduce the sample variance by the extra information that every heuristic strategy of a given person-attribute pair will retrieve same set of pages. We assume  $D_i$  are independently from a normal distribution  $N(\mu, \sigma^2)$ . We perform paired t-test to show if the heuristic improve the result with the following hypotheses:  $H_0 : \mu = 0$  and  $H_A : \mu > 0$ .

Each of the testings between no-heuristics strategy and every of page rank strategy, background noise removal strategy, length filter strategy, and all-heuristic strategy contains 43 samples. Table 3 is the summary of our testings.

*Page rank strategy* has very high p-value, which means it has negative contribution to the precision of our result. However, if there are several persons with the same name, this strategy will more likely to help us to find the exact person.

*Background noise removal strategy* has p-value 1.3%. Therefore, we have high confidence to say this heuristic improve our performance statistical significantly.

*Length filter strategy* has p-value 12.2%. This is be-

Table 3: Comparing different heuristics through paired t-test.

Heuristic	$Mean(D_i)$	$Var(D_i)$	t	p-value
PageRank	-0.1162	0.5859	-1.3013	0.8998
BackNoise	0.6279	1.7865	2.3047	0.0130
Length	0.3255	1.8089	1.1802	0.1222
All-Heuri	0.9069	1.9977	2.9770	0.0024

cause our current filter is a step function defined manually. If we apply more sophisticated parameter-tuning distribution, e.g. Gamma distribution, we might get this heuristic with higher confidence being beneficial.

*All heuristic strategy* has p-value 0.24%, beating all other single heuristic significantly. Moreover, we can claim that with these heuristics, we can get higher precision with very high confidence.

#### 4.4 Comparing with Google

There are very few similar applications or previous works, so it is hard to have a comparison between our system and other related works. Hence, we only compare our approach with an intuitive person-attribute search strategy using Google. For a person-attribute pair, we use the concatenation of the person name and the attribute name as the query string, and compare our rank list with the ranked snippets list returned from Google. Most of the desired information even cannot be found in any snippets returned from Google; while our system can find the desired block with high rank in most of the cases.

One reason is that the page containing our target information may not contain the attribute name in its content. For example, the contact information shown in a personal homepage may just contain the address, phone number, and email address (all of these are eas-

ily understandable for human readers) without an indicating head “contacts.” Another reason is a bit unobvious. Since the current keyword-matching-based search engines cannot distinguish the relation between the input query words, the query words are treated equally. However, the instance and attribute have relation analogous to that between the master and slave. Usually, the instance would be more specific (e.g., a person name) and the attribute would be more general and topic-oriented (e.g., phone, school, research interests, etc). Combining them together sometimes distort the page searching and ranking of current search engines. For example, when submitting “Marco Caccamo<sup>3</sup> graduating school” to Google, the top-1 page is titled as “Cape Coral High School,” which definitely stands far away from our original intention of this request. Relying on the co-occurrence of instance name and attribute name using current search engines is not feasible.

## 5 Related Work

PAIR aims to provide a block-based retrieval engine to benefit person-attribute extraction and various related applications. Here, we link it with related works in following aspects: information extraction, passage retrieval, and query expansion.

Information extraction (IE) aims at extracting and organizing the relevant information from text, and has been studied extensively in the literature [1, 10]. As Web becomes biggest repository of data recently, many researches on IE seeks their applications or tasks using the Web as the corpus. For example, Brin [2] extracted tuples from Web-scaled corpus through bootstrapping based on the notion of duality of extraction patterns and target tuples. Ciravegna et al. [4] proposed a similar idea by integrating various kinds of information available on the Web to perform extraction. Our work is not directly addressing the extraction task. Instead, we use the instance (i.e., person name in this work) combined with top search engines to retrieve potential pages and then use the attribute to re-rank the potential blocks [5] that contain our target information, which can be treated as a mediate step towards various IE tasks. Hence, our project aims to provide a generic, systematic supporting infrastructure for these various IE applications.

Block retrieval is analogous to passage retrieval in traditional IR works [8]. Different from conventional text documents, Web pages have visual clues which enable us to perform more sophisticated segmentation of “passages,” which are usually referred as blocks in the Web pages [5, 6, 7].

Also, our learning of an attribute concept is performed through the retrieved snippets for the corresponding attribute name. This idea can be linked

to query expansion, particularly the work on pseudo-relevance feedback, which is used to improve the retrieval performance with expansion terms extracted from the top-ranked documents [13, 14, 3]. In our approach, the Google search engine is used to expand the terms. The main difference between our approach and the traditional query expansion is that we use external documents rather than internal ones.

Entity Retrieval (ER) is proposed by Sayyadian et al. [12] to find the information of an entity (e.g. researcher) from both the structured and text data in an integrative way. The paper displayed the potential to help the retrieval in plain text by database and vice versa. The first step of our method is similar to ER in finding Web page which contain the information of an entity. However, we employ a relative simple method. Go beyond this, our goal is to find the relevant text snippet which contains our target information.

## 6 Conclusion

In this project, we develop a system for person-attribute retrieval, based upon the current state-of-the-art search engine and the techniques of vision-based page segmentation, blind query expansion, and block retrieval. An initial experiment on several faculty is performed and analyzed. The positive results show that our approach is applicable.

For future work, we are considering the following two directions:

1. Each component in our system still needs improvement. In our current stage, we use only a few of top-ranked pages from Google as the candidate pages for further processing. However, the target information may be contained in some lower-ranked pages. How to select better candidate pages needs further investigation. Running vision-based page segmentation is time-consuming, and this is the reason that we process just a few of top-ranked pages. We may need to explore a faster and light-weighted segmentation algorithm. Our current attribute expansion is very simple. Although it does help the retrieval of the target information, it brings a lot of noisy terms. Our experiment results show that the incorporation of a background model improves the performance a lot. How to handle these noisy terms is worth of further exploration. In block retrieval, we briefly investigate several scoring heuristics to improve the relevance measurement. The results show that most of them are useful. However, we still need to explore each of them as well as discovering other useful heuristics, and to study a more principle and systematic way for incorporating them.
2. We are considering to extend our framework to other instance domains. Currently, we focus on

---

<sup>3</sup>A faculty of UIUC CS department

the person domain. There are still other instance domains, such as book, movie, car, etc. From our initial analysis, the major difference is the backend search engine. There are some good sources for other domain, such as Amazon.com for books. Is it possible to replace the backend search engines so that the whole framework can be directly adapted to other domains? This would be an interesting direction to explore.

## References

- [1] Douglas E. Appeltand and David J. Israel. Introduction to Information Extraction Technology. IJCAI99-Tutorial, 1999.
- [2] Sergey Brin. Extracting patterns and relations from the World Wide Web. WebDB Workshop at EDBT'98.
- [3] C. Buckley, G. Salton, and J. Allan. Automatic retrieval with locality information using smart. In *Proceedings of the First Text REtrieval Conference (TREC-1)*, pages 59–72, 1992.
- [4] F. Ciravegna, A. Dingl, S. Chapman, and Y. Wilks. Integrating information to bootstrap information extraction from Web sites. IJCAI 2003 Workshop on Information Integration on the Web.
- [5] Deng Cai, Shipeng Yu, Ji-Rong Wen, Wei-Ying Ma. Block-based Web Search. SIGIR'04.
- [6] Deng Cai, Shipeng Yu, Ji-Rong Wen and Wei-Ying Ma. VIPS: a Vision-based Page Segmentation Algorithm. Microsoft Technical Report (MSR-TR-2003-79), 2003.
- [7] Deng Cai, Xiaofei He, Ji-Rong Wen, Wei-Ying Ma. Block-Level Link Analysis. SIGIR'04
- [8] James P. Callan. Passage-Level Evidence in Document Retrieval. SIGIR'94, pp. 311-317.
- [9] Hui Fang, Tao Tao, and ChengXiang Zhai. A formal study of information retrieval heuristics. SIGIR'04
- [10] Tom Mitchell. Extracting targeted data from the Web. SIGKDD'01.
- [11] G. Salton and C. Buckley. Term weighting approaches in automatic text retrieval. *Information Processing and Management*, 24:513–523, 1988.
- [12] Mayssam Sayyadian, Azadeh Shakery, AnHai Doan, ChengXiang Zhai. Toward Entity Retrieval over Structured and Text Data. WIRD'04, the first Workshop on the Integration of Information Retrieval and Databases (WIRD'04).
- [13] Jinxi Xu and W. Bruce Croft. Query expansion using local and global document analysis. SIGIR'96.
- [14] ChengXiang Zhai and John Lafferty. Model-based feedback in the language modeling approach to information retrieval. CIKM'01.