

# 國立交通大學

## 資訊科學與工程研究所

### 碩士論文



WebPuzzle: 網際網路中的真實物件搜尋

WebPuzzle: Object Discovery in World Wide Web

研究生：蔡尚樺

指導教授：彭文志 教授

中華民國九十七年六月

WebPuzzle: 網際網路中的真實物件搜尋  
WebPuzzle: Object Discovery in World Wide Web

研究生：蔡尚樺

Student : Shang-Hua Tsai

指導教授：彭文志

Advisor : Wen-Chih Peng

國立交通大學  
資訊科學與工程研究所  
碩士論文



Submitted to Institute of Computer Science and Engineering

College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer Science

June 2008

Hsinchu, Taiwan, Republic of China

中華民國九十七年六月

## 摘 要

物件搜尋技術已被證明在網際網路中，不直接檢視每一個頁面，而是直接搜尋如電話號碼與電子郵件地址等實際物件之功能十分有用，該技術透過將網路中的文件模型轉換為物件模型使得直接搜尋物件變的更加直覺。然而，由於過去的物件搜尋系統中的限制，對於搜尋多種物件之要求，唯有曾出現在同一頁面之物件群能夠被搜尋。在這篇論文中，我們將物件搜尋的問題建置成一個解謎的模型，本系統將對網際網路之文件進行分析後，創造一個宏觀的觀點並建立每一個物件之間的關係圖，藉由物件之關係圖來搜尋無直接關聯性之物件群，我們使用真實世界之資料作為實驗依據，並經由實驗證明本論文提出之系統在物件搜尋之效率與準確性。

關鍵字：物件搜尋，資訊擷取。



## Abstract

Entity search has been proved to be handy to search data “entities” such as phone number and email without looking them indirectly from individual pages. The technique transforms the web from the document view to the entity view, which enable more holistically search. However, due to the limitation of previous models, the resulting entities are limited in the same pages. In this paper, we model the entity search problem as a puzzle problem. The framework digests all pages and builds a global view on how the entities should be combined. By utilizing the entity graph, the framework is able to composes entities into tuples even they are not directly related. We evaluate our system using real world web pages and show that the system is efficient for searching entity tuples.

*Keywords* — Entity search, information retrieval

## 致 謝

首先要感謝我的指導教授彭文志教授與伊利諾大學香檳分校的張振川教授，因為有老師們的積極指導，才能做出如此有趣的題目，在撰寫論文的過程中，讓我學習到了做研究的方法和追求創新的精神。此外，還要感謝口試委員李素瑛教授與陳宜欣教授給予我許多寶貴的意見，使得本篇論文的部分缺失得以改善。

在實驗室的日子讓人難忘，感謝博士班的學長洪智傑，自我入學以來給予我許多幫助，使我能順利度過碩士生涯，博士班的學長姐魏綾音、廖忠訓、江孟芬與碩士班的學長姐黃正和、游敦皓、李柏逸、周佳欣對我的照顧讓我十分感激，碩士班同學駱嘉濠、傅道揚、蔣易杉、郭員榕也是我的良師益友，與你們的討論使我獲益良多，實驗室的學弟妹黃琬婷與王琮偉，祝福你們能在研究的路上一帆風順，各位帶給我的溫暖與歡樂，是我碩士班美好的回憶。

最後要感謝我的父母，始終站在我的背後支持我，給予我莫大的精神力量。因為有你們的付出，讓我能無後顧之憂的專注於研究，感謝你們為我作的一切。

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Preliminaries</b>	<b>5</b>
2.1	Data Model . . . . .	6
2.2	Entity Search Characteristics . . . . .	7
2.3	Our Propose: Puzzle Solving . . . . .	8
<b>3</b>	<b>Entity Graph</b>	<b>10</b>
3.1	Document-centric Analysis . . . . .	11
3.2	Corpus-centric Analysis . . . . .	12
3.3	Entity Graph Construction . . . . .	13
3.4	Tuple Composition . . . . .	17
<b>4</b>	<b>Entity Search</b>	<b>19</b>
4.1	Concept-Driven Tuple Ranking . . . . .	19
4.2	The Entity Index . . . . .	21
4.3	Tuple Expansion by A* Search . . . . .	22
<b>5</b>	<b>Experiments</b>	<b>25</b>

5.1	Precision . . . . .	26
5.2	Study of Tuple Size . . . . .	28
5.3	Tuple Listing . . . . .	29
<b>6</b>	<b>Related Works</b>	<b>32</b>
<b>7</b>	<b>Conclusions</b>	<b>35</b>



# List of Tables

3.1	Relations in Figure 3.1 . . . . .	14
5.1	Number of entities annotated . . . . .	26
5.2	Ranking for case\tupleSize . . . . .	30
5.3	Different Tuples for Common Surnames . . . . .	31





# List of Figures

1.1	An example of cross-page tuple . . . . .	2
2.1	Entity search framework architecture . . . . .	9
3.1	Example web pages . . . . .	11
3.2	Sub-trees of entity graph for documents in Figure 3.1 . . . . .	17
4.1	Relate tuple to query . . . . .	21
5.1	Satisfied query percentage under various ranks . . . . .	27
5.2	Satisfied query percentage under various number of entity viewed . . . . .	28
5.3	Satisfied query percentage under various ranks for different size of tuple scheme . . . . .	29
5.4	Number of valid tuples under various ranks . . . . .	31

# Chapter 1

## Introduction

The web is an ultimate information repository, and it is still growing rapidly. In order to search information in the unorganized and unstructured web, several search models has been introduced. The document-based search engines provide users a page view of the web. The basic unit in the page view is a document, which is good enough for some kind of user intentions such as “Find blog posts which is related to a topic” and “Find related information about a person”. However, the document-based search model is far from intuitive for users who want to find specific entities; for example, “What is the phone number and email address of someone”. Although users can narrow down the search results by providing more precise keywords to the search engine, they still have to digest the pages to find the exact entity they are looking for.

Due to the desire of a more intuitive way to retrieve structured information from the web, vertical search, which search the target domain in a more organized way, has gain focus in recent years. Although there are general frameworks to build vertical search, building vertical search engines for arbitrary domains is still a very complicated task due



Figure 1.1: An example of cross-page tuple

to the cost of data integration. As a result, the entity search, which provide user an entity view of the web, is proposed. Instead of pages, users search for specific entity tuples in the entity search engine. It is more efficient to specify what we are looking for directly rather than digest the a set pages ourselves. Consider the following scenarios which compare the page-based search with the entity-based search:

- Scenario One:** An user who wants to find all universities that has the computer science department may issue the query "university computer science" in the page-based search engine and try his best to view all the pages and digest the desired tuples. Yet in the entity-based search engine, the user is able to issue the query "#University #Department:'computer science'" to list all tuples directly.
- Scenario Two:** An user wants to list all {zip code, address, phone number} tuples in California, it is almost impossible to do this in keyword-based search model. However, in the vertical search engine, one can issue the query "#State:California #Zipcode #Address #Phone" to list the desired contact information.

The previous work of entity search[21] has studied the ranking criteria of tuples very well; however, the system is limited by the proximity window size and page boundary. As a result, the system is only able to find entity tuple which has appeared together in at least one page; for example, the previous system can not find tuples in Figure 1.1.

In this paper, we study a different approach to construct the entity search framework. The framework is designed to search entity tuples of arbitrary sizes and consider the relationship between different entities in a corpus-centric view in order to find entity tuples across multiple pages.

To serve an entity search query, the system has to grab a set of entities from the corpus and rank the composed tuples. The process is like solving a puzzle problem in which the solver has a lot of puzzle pieces and want to compose the pieces into the desired form.

In the puzzle problem, each piece provides the solver some information about how this piece can be combined with others. The same thing holds for the entity search problem, in which each entity is a piece of the desired tuple. Due to the redundancy of the web, the same information would appear several times in different websites. In the web puzzle problem, each page provides the system information that how to compose entities together. However, the answer of the web puzzle problem varies from query to query; the tuple scheme defines what kind of pieces (entities) the solver need to compose the answer (tuple) and the keywords give the solver hints about what is a good answer.

In summary, our contributions are as follows:

- We design the entity search framework in a new approach, which offers better precision and functionality.
- We model the entity search as a puzzle problem utilizing the redundancy of web and study the inference process to search entity tuples across page boundary.
- The framework is able to handle web scale data by paralleling the off-line relation recognition with the map/reduce programming model and utilizing the distributed

index for on-line query serving.

We start in Section 2 to the conceptual model to solve the entity search problem using the puzzle solving approach. Section 3 presents the entity graph and Section 4 materializes the tuple composition and the ranking criteria. We study the related works in Section 5. Section 6 evaluate our prototype system using real world data set.



# Chapter 2

## Preliminaries

Looking for entity tuples from the web is like a puzzle problem. Each page provide some hints about the solution tuple. When a user search for information, the user actually tries to solve the web puzzle. Each page offers some evidence about how the entities should be combine into the answer tuples. By surfing the pages in the partial web, the user digests the relation between entities and composes the answer tuples based on his intention. However, digesting all the pages in the web is an impossible task for a single user and there is no guarantee that no information is missing by surfing only the limited number of pages. As a result, the entity search system is crucial for such web data analysis tasks.

The entity search framework solves the following problem:

- **Given:** A set of documents  $\{d_1, d_2, \dots, d_n\}$ .
- **Input:** The tuple scheme  $\{E_1, E_2, \dots, E_m\}$  and keywords to narrow down the returned tuples.
- **Output:** Ranked entity tuples in  $\{E_1 \times E_2 \times \dots \times E_m\}$ .

## 2.1 Data Model

The web is a collection of documents  $D = \{d_1, d_2, \dots, d_n\}$  and each document  $d_i$  is a collection of terms  $\{t_1, t_2, \dots, t_{|d_i|}\}$ . In order to search entity holistically, we have to model the web in the entity view. An entity is a  $(value, type)$  pair and each document is annotated as a collection of entities. We will use  $type(e)$  to represent the type of the entity  $e$  and use  $E_t$  to represent all entities of type  $t$ . For example, an contact page of a company may be annotated as  $\{(012-345-6789, \#phone), (contact@example.com, \#email), \dots\}$ .

To relate entities with concept, keywords are also annotated as a type of entity. Instead of page, the basic unit in the entity search system is entity. Given the document repository, the system have to transform the documents into a set of entities and recognize their relationship in order to support the tuple composition process.

An entity search query contains two parts: concept and tuple scheme. The concept is in the form of keywords in the page-based search system while the tuple scheme is a set of entity types. Each entity type is prefixed by the  $\#$  symbol. For example, the query “Turing Award  $\#$ Person  $\#$ University  $\#$ Email” searches for {person name, university, e-mail} tuples related to the concept “Turing Award”. It is also possible to narrow down the search results by limiting the value in an entity type. For example, “Computer Science  $\#$ Person  $\#$ University  $\#$ State:California” lists only the {person name, university, state} tuples related to the concept “Computer Science” and the value of the state entity is “California”.

## 2.2 Entity Search Characteristics

Since the characteristics of an entity-based search system is different from the page-based search system, the difference drives the different system requirements and design issues between the two. Given the concept, which is represented in keywords, the desired entity tuples are directly returned in entity-based search system while users have to digest the results in page-based search system. Therefore, the entity search system should not only be able to annotate entities but also to compose those entities into correct tuples. Moreover, since not all entities in the resulting tuples in directly related to the keywords; in the other words, they may not even appears in the same page with the keywords. As a result, the system should consider the relation between entities in a global view rather than being limited by only pages containing the keywords.

The major difference comes from that the page-based search system requires only the returned pages to match the given concept and entity-based search requires the entities to be annotated and composed into tuples. There are three major requirements in an entity-based search system:

- **Annotation:** The system should be able to annotate entities in the unstructured documents.
- **Composition:** Entities should be composed into tuples based on their relation in the corpus.
- **Ranking:** The entity tuples should be ranked based on their correlation to the given concept.

The entity annotation is well studied in information extraction [4][8][3]. As a result



we will focus on tuple composition and the ranking criteria of entity tuples in this paper.

## 2.3 Our Propose: Puzzle Solving

A very special characteristic of the web is the redundancy, which means the same information may appear several times in different form. The redundancy introduce the following hypothesis.

**Hypothesis 1.** *An entity tuple exists in the real-world if and only if there is a way to discover all entities in the tuple starting from one of the entities.*

The hypothesis actually matches the way how we discover information as well as how we solve a puzzle problem. To solve a puzzle, we start from one piece of the puzzle; see how it matches to the other pieces; add more pieces; and get to the answer after several iterations. We know which piece to add because the piece itself gives us some information about how it is related to other pieces. The same situation happens to the entities in the web. Each pages provide us information about how entities are correlated. Suppose whenever we see entity  $e_i$ , we also see  $e_j$ ; we probably know that if a tuple contains  $e_i$ , it may also contains  $e_j$ . The appearance of the entity itself gives us, the searcher, how it is related to other entities.

In our framework, we first analysis the whole corpus and build the relation between any two entities. The relation is represented in the conditional probability form.  $P(e_{post}|e_{pri})$  is the probability that when  $e_{pri}$  is in the tuple,  $e_{post}$  is also in the tuple. Given the concept, we can choose some entities as the starting points and discover other entities based on the relation.

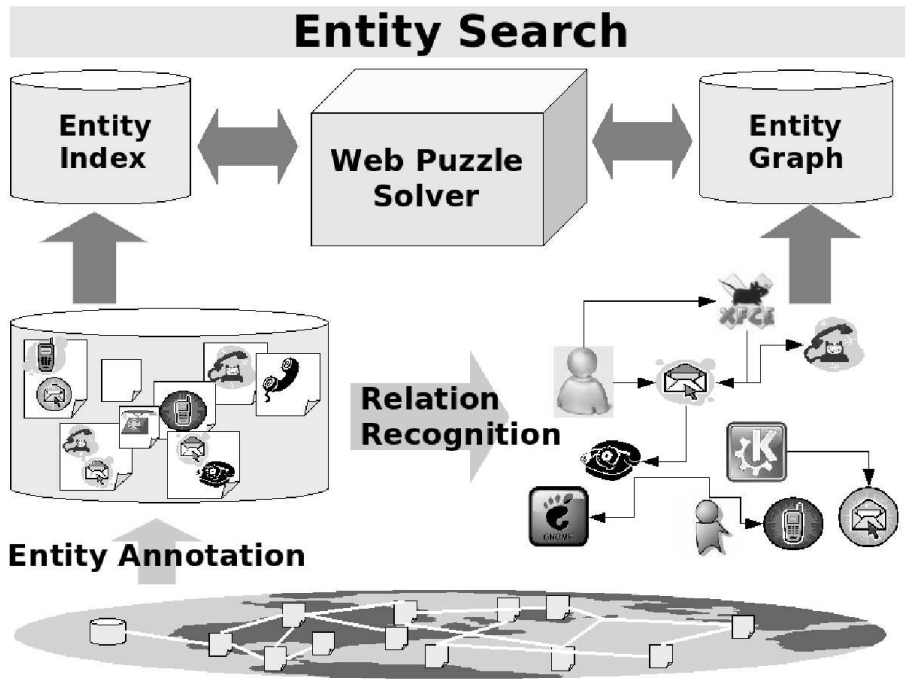


Figure 2.1: Entity search framework architecture

The framework architecture is shown in Figure 2.1. The web pages is first annotated and materialized into the entity graph. Whenever there is a user query, we choose some entities in the entity graph and discover other entities based on the tuple scheme and their relation with the concept.

# Chapter 3

## Entity Graph

To catch the global relation between entities, the system has to analysis the whole corpus construct a global view of entities. The global is stored as an entity graph  $G = (V, E)$ , where  $V$  is the set of all entities and  $E$  is the correlation between entities. The entity graph is a directed graph and the weight of an edge from  $e_i$  to  $e_j$  is the probability that an user can discover  $e_j$  by starting from  $e_i$ . By utilizing the entity graph, the system is able to compose entities into tuples based on their correlation.

The analysis process is generalize into two steps: document-centric analysis and corpus-centric analysis. Each document is analyzed and provides the system some evidence. The evidences from each documents is aggregated as a global view and the entity graph is constructed. Since the recognition function can be varies for any two different types, we study only the basic probability model which consider only co-occurrence of entities but not context in the documents. However, the recognition can be easily replace by arbitrary model based on domain knowledge.

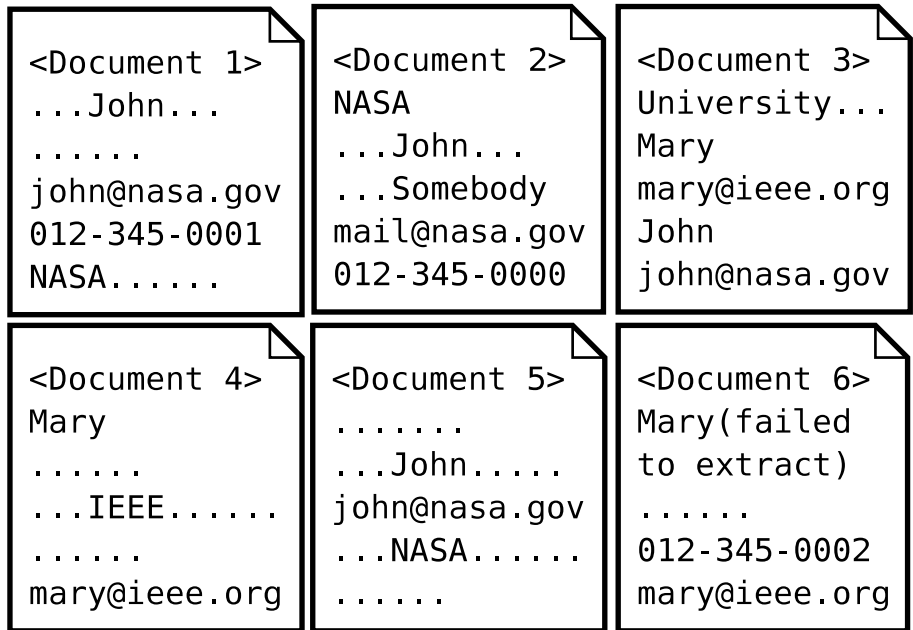


Figure 3.1: Example web pages

### 3.1 Document-centric Analysis

Consider the documents in Figure 3.1, each document gives us some information about the entities in them even without seeing other context in the document. Suppose there is only one document 1, it is very possible the person name “John” is highly correlated to the email “john@nasa.org”.

The confidence comes from the presentation of entities in the document. The more complex a document, the lesser information it provides for an entity pair. For example, in document 3, we are not so sure that which email is related to “John” because there are two person name entity and two email entity in the document. Consider only the co-occurrence of entities, we formulate the confidence estimation function for any two entities  $e_i$  and  $e_j$  appears in document  $d$  as follow:

$$C_d(e_j|e_i) = \frac{|\{(e_i, e_j) \in d\}|}{|\{(x, y) \in d | x \in \text{type}(e_i), y \in \text{type}(e_j)\}|}$$

The above function consider only the occurrence of entities in the document. The more entities of same type in a document, the lower its confidence to an entity pair contains entity of the type. It is also possible to consider other factors such as word distance or the number of entities of the types between two entities.

$$C_d(e_j|e_i) = \frac{1}{\text{distance}(e_i, e_j)}$$

One can simply use the occurrence-based function or optimize the document-centric analysis function based on domain knowledge. It is also possible to choose different function for different entity types. However, to make the system simply to bootstrap, we use the occurrence-based function by default. A more detailed analysis on document-centric analysis function is studied in the experiment section.

## 3.2 Corpus-centric Analysis

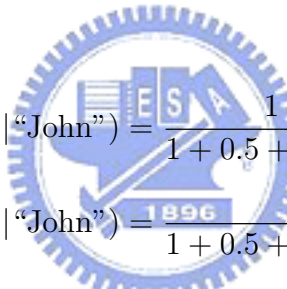
Given an entity, one can discover other entities of different types by surfing documents in the corpus. With unlimited time and patient, one can summarize the impression and analysis the relation between entities.

**Definition 1.** *Given two entities  $e_{prior}$  and  $e_{posterior}$  of different types.  $P(e_{posterior}|e_{prior})$  is the probability that one discover  $e_{posterior}$  from  $e_{prior}$  rather than another entity of type  $\text{type}(e_{posterior})$  by randomly surfing the corpus.*

The corpus-centric analysis function, which aggregate the document-based confidence, is defined as follow:

$$P(e_{post}|e_{pri}) = \frac{\sum_{d \in D} C_d(e_{post}|e_{pri})}{\sum_{e_j \in type(e_{post})} \sum_{d \in D} C_d(e_j|e_{pri})}$$

Each document in the corpus contribute some confidence to the estimation; however, the final probability estimation is determined by the overall impression in the whole corpus. Consider the example of six documents in figure 3.1, the person name “John” has co-occured with three different emails. There are five (#Email | “John”) pairs with different confidence estimation in the corpus. The probability that discover an email entity from the person entity “John” are as follows:



$$P(\text{“john@na...”} | \text{“John”}) = \frac{1 + 0.25 + 1}{1 + 0.5 + (0.25 + 0.25) + 1}$$

$$P(\text{“mail@na...”} | \text{“John”}) = \frac{0.5}{1 + 0.5 + (0.25 + 0.25) + 1}$$

$$P(\text{“mary@ie...”} | \text{“John”}) = \frac{0.25}{1 + 0.5 + (0.25 + 0.25) + 1}$$

We can construct the entity graph by estimating the discovery probability between all entity pairs in the corpus. All entity pairs with discovery probability greater than 0.6 are shown in Table 3.1.

### 3.3 Entity Graph Construction

With the millions or even billions of entities in the web, constructing the entire entity graph is a non-trivial task. As a result, the system should be able to parallel the entity annotation as well as the relation recognition process. In the prototype system, we use

$e_i$	$e_j$	$P(e_j e_i)$
John	john@nasa.gov	0.75
Mary	mary@ieee.org	0.833
Somebody	mail@nasa.gov	1
John	012-345-0001	0.667
John	NASA	0.833
Mary	IEEE	0.667
john@nasa.gov	John	0.9
mary@ieee.org	Mary	0.833
john@nasa.gov	012-345-0001	1
mail@nasa.gov	012-345-0000	1
mary@ieee.org	012-345-0002	1
john@nasa.gov	NASA	0.8
mail@nasa.gov	NASA	1
mary@ieee.org	IEEE	0.667
012-345-0001	John	1
012-345-0000	mail@nasa.gov	1
012-345-0001	john@nasa.gov	1
012-345-0002	mary@ieee.org	1
012-345-0000	NASA	1
012-345-0001	NASA	1
NASA	John	0.833
IEEE	Mary	1
NASA	john@nasa.gov	0.667
IEEE	mary@ieee.org	1

Table 3.1: Relations in Figure 3.1

the Gate[9] toolkit for entity annotation. The entity graph construction process is implemented in the Map/Reduce programming model[11] using the open source Hadoop[1] framework.

The mapper and the reduce is shown in Algorithm 1 and Algorithm 2 respectively. In the mapper function, each document is annotated based on the given tuple scheme and a set of entities is produced. The document-centric analysis function  $C_d$  is also done in the mapper function right after the entities are annotated. The function can be arbitrary function that output the a confidence value between 0 to 1 given the annotated document. Moreover, it is possible to set a threshold value for any two types in order to prune less confident entity pairs. After the analysis is done, each entity pair is emitted to the reducer to perform the corpus-centric analysis.

---

**Algorithm 1** Entity graph constructor mapper

---

**Input:** document  $d_i$  in corpus  $D$ , object scheme

**Output:**  $(\{e_{pri}, type\}, \{e_{post}, confidence\})$  pairs

- 1: Entities  $E = \text{annotate}(d_i)$
  - 2: **for all**  $(e_i, e_j) \in E$  **do**
  - 3:    $c = C_d(e_j|e_i)$
  - 4:   **if**  $c > \text{threshold}[type(e_i), type(e_j)]$  **then**
  - 5:      $\text{emit}(\{e_i, type(e_j)\}, e_j, c)$
- 

Before passing to the reducer function, the system group the entity pairs by the value of  $e_{pri}$  and the type of  $e_{post}$ . The confidence value for each entity is aggregated by the corpus-centric analysis function. We introduce the null matrix  $M_{null}$  here to handle noise from the document-centric analysis function. The element  $m_{ij}$  in  $M_{null}$  denote the



probability that an entity of type  $t_i$  can not discover any entity of type  $t_j$ . Conceptually, we create a virtual entity with null value for each type. Each entity in the corpus has a virtual edge link to the null entities with probability  $m_{ij}$ . All entities with corpus-centric confidence value less than  $m_{ij}$  are pruned to eliminate the noise. Moreover, the pruning saves the space to store the huge entity graph. The values in  $M_{null}$  can be assigned based on domain knowledge or any reasonable value that properly shrink the entity graph.

---

**Algorithm 2** Entity graph constructor reducer

---

**Input:**  $(\{e_{pri}, type\}, \{e_{post}, confidence\})$  pairs

**Output:** Edges  $(e_{pri}, \{e_{post}, confidence\})$

```

1:  $E = \text{new set}()$ 
2: for all  $e, c \in \{e_{post}, confidence\}$  pairs do
3:   if  $e \notin E$  then
4:      $E.\text{put}(e)$ 
5:      $e.confidence = c$ 
6:   else
7:      $e.confidence = e.confidence + c$ 
8:  $total = \sum_{e \in E} e.confidence$ 
9: for all  $e \in E$  do
10:   $e.confidence = e.confidence \div total$ 
11:  if  $e.confidence > M_{null}[e_i.type, e_j.type]$  then
12:     $\text{emit}(\text{new Edge}(e_{pri}, \{e, e.confidence\}))$ 

```



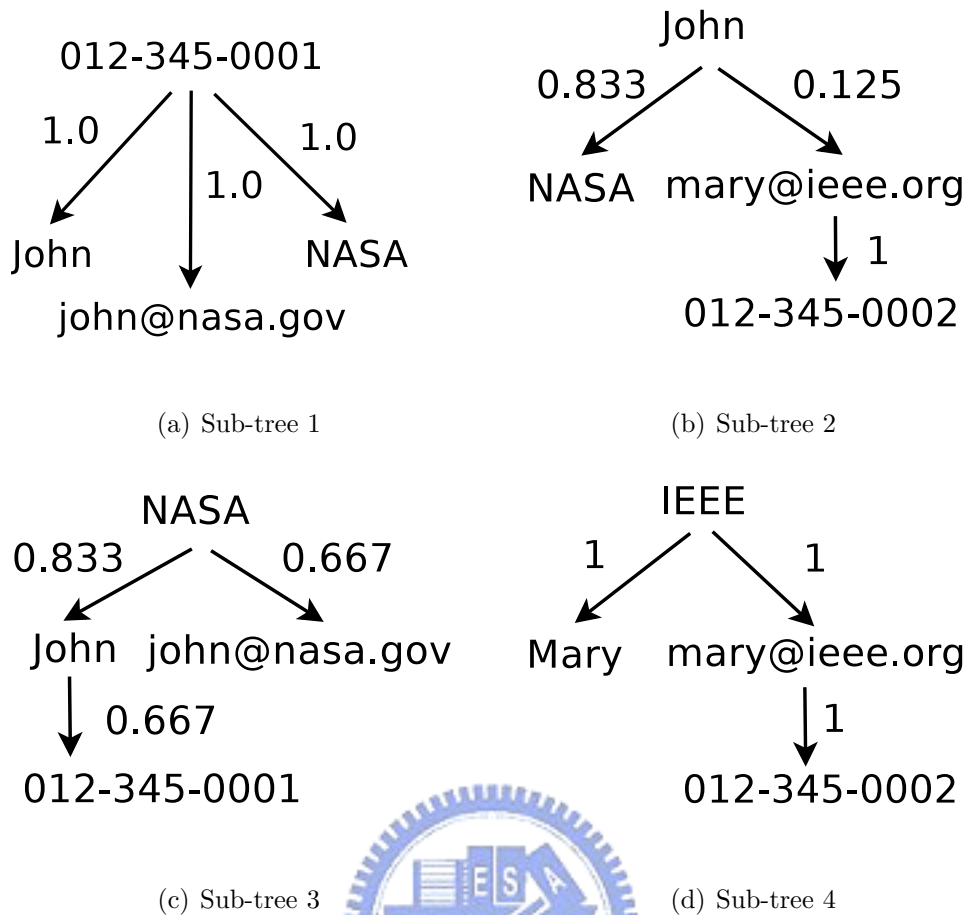


Figure 3.2: Sub-trees of entity graph for documents in Figure 3.1

### 3.4 Tuple Composition

As in our hypothesis, if a tuple exists in real world, there should be a way to find all entities in it from one of the entity. The entity graph provides the system about how to “discover” entity one by one. The tuple composition process is like traverse the entity graph in random walks. Suppose the user starts from knowing one entity, the root entity, and want to discover the other entities of different types. With his knowledge about the known entities, the weight of the edge gives him some hints about which entity to go next. He may go back and forth in the entity graph and finally collect the desired tuple.

Figure 3.2 shows some sub-trees in the entity graph of documents in Figure 3.1.

The sub-trees are possible combination of entities and represents the discovery path of entities in the tuple. Given all sub-trees which contains one entity for each type in the tuple scheme, some of them are more likely to be discovered by random walking in the entity graph starting from random entity. For example, it is more likely to discover tuple in Figure 3.2(a) and 3.2(d) rather than the one in Figure 3.2(b). Suppose the we start from entities that match the types in tuple scheme and only discover one entity for each type in the scheme. We can define the confidence of a tuple in the following equation:

$$C(tuple) = \max_{\text{sub-tree } t \rightarrow tuple} \prod_{\text{edge } e \in t} weight(e)$$

Consider starting from the root entity as the known entity and discover other entities by following the edge in the entity graph; whenever a new entity is discovered by edge  $e$ , we have a probability  $weight(e)$  to continue the discovery rather than another discovery process. As a result, the probability of accepting a sub-tree is the product of the weight of its edges. Since there may be multiple sub-trees represents the same tuple, such as the two sub-trees in Figure 3.2(a) and 3.2(c). We choose the one with maximum probability to represent the confidence of the tuple to avoid the noise in tuple composition since there may be a lot of low confidence sub-trees represents the same tuple.

# Chapter 4

## Entity Search

We will concretely materialize the entity search framework in this section. As the requirements for entity search, the entity tuples should not only be valid but also be relevant to the user given keywords. The keywords in the query guide the system the starting points to search in the entity graph and the system rank the tuples found based on their relevance with the keywords. To enable the framework, we adapt the inverted list index into entity index and search the entity graph with A\* search algorithm.

### 4.1 Concept-Driven Tuple Ranking

Since there are a lot of entity tuples matching the tuple scheme, the entity search provide a function to narrow down the search results by introducing the keywords. The keywords describe the concept which the resulting tuples should be related to.

Given the query  $q$ , we can transform  $q$  as an entity of unique type in the entity graph by computing the edges between  $q$  and other entities in the graph. With the keyword index, we can obtain the subset  $D_q$ , which contains query  $q$ , in the corpus  $D$ . With the

entity index, we can obtain the subset  $D_e$ , which contains the entity  $e$ , in the corpus. The inlink and outlink between an entity  $e$  and query  $q$  can be calculated by  $P(e|q)$  and  $P(q|e)$  respectively. The detail of computing  $P(e|q)$  and  $P(q|e)$  utilizing both keyword index and entity index is discussed in the next sub-section.

$$P(e|q) = \text{document frequency}(e, D_q)$$

$$P(q|e) = \text{document frequency}(q, D_e)$$

Suppose we have the concept in mind, we can find the  $D_q$  document set by the keyword index. By browsing all documents in  $D_q$ , we can get an impression that how all entities are related to the concept. However, the entities of the desired tuple may not all appear in  $D_q$  so we still have to traverse the entity graph to compose the tuple. Whenever we discover a new entity in the graph, in addition to check the confidence of the edge, we also check the confidence of the entity  $e$  given the query  $q$ . We have the probability  $C(q, e)$  to continue the discovery process rather than re-starting the discovery. The probability  $C(q, e)$  is defined as follow:

$$C(q, e) = \min(\max(P(q|e) + P(e|q), P_{base}), 1)$$

$P_{base}$  is the base probability for those entities that does not co-occur with the query in any document.  $P(e|q)$  may be much lower than  $P(q|e)$  since the concept can be very general and matches much more documents than the entity. The function can be explained as the confidence of an entity  $e$  given  $q$  is proportional to  $P(q|e)$  and breaking tie by  $P(e|q)$ .

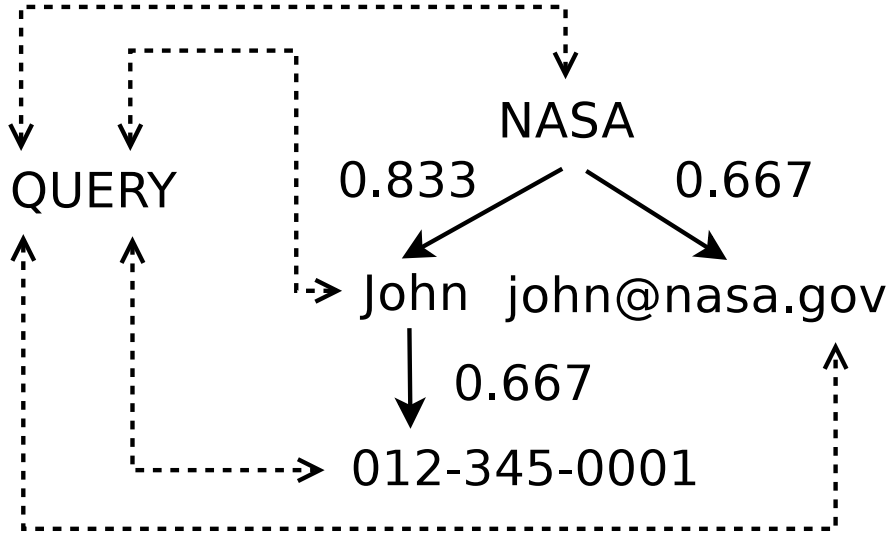


Figure 4.1: Relate tuple to query

Given the concept-driven query  $q$ , the tuples discovered in the entity graph are ranked by the following ranking function.

$$Score(q, t) = C(t) * \prod_{e \in t} C(q, e) \quad (4.1)$$

We traverse the entity graph to compose the entity tuples. When we select the root node, we have a probability to give up this traversal; whenever we expand the tuple, we check the path so far as well as the confidence of the newly added entity. The score is the probability that the discovery process from the root entity has successfully completed. For example, we evaluate the tuple in Figure 4.1 by all its edges and the confidence of each entity in it give the query.

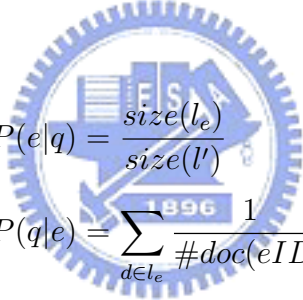
## 4.2 The Entity Index

To enable entity search, we store the entity graph in a memory cached table for fast look-up. Every entity is encode into a 8-bytes ID and the graph is stored in distributed

machines. Moreover, we construct an entity-based inverted list for each type of entity with the following structure.

$$\#TYPE \rightarrow \{\text{docID}_1, \{(eID, \frac{1}{\#doc(eID)}), \dots\}, \dots\}$$

The  $eID$  is the id of the entity and  $\#doc(eID)$  is the number of documents containing the entity with  $eID$ . By utilizing the entity index, we can join the inverted list of an entity type with keyword-based inverted lists. Given the inverted list  $l_q$  of query  $q$  and the inverted list  $l_E$  of entity type  $E$ , we can join the two to obtain the joined list  $l'$ . For each entity  $e$ , we can get its sub-list  $l_e$  in  $l'$ . Then we can calculate the confidence of any entity  $e$  given query  $q$  by:



$$P(e|q) = \frac{\text{size}(l_e)}{\text{size}(l')}$$

$$P(q|e) = \sum_{d \in l_e} \frac{1}{\#doc(eID)}$$

### 4.3 Tuple Expansion by A\* Search

The algorithm that serves entity search query is shown in Algorithm 3. Given the query  $q$  and the tuple scheme  $S = \{E_1, E_2, \dots, E_n\}$ , the system first checks the index and constructs the set of entities which are co-occurred with  $q$  in the documents containing  $q$ ; then, it constructs size one tuples rooted by each entity in the set. The initial tuple set is expanded based on the entity graph by A\* search.

Given a partial tuple  $t$ , the cost function  $g(t)$  and the hypothesis function  $h(t)$  are defined as follows:

$$g(q, t) = \prod_{entity \in t} C(q, entity) \times \prod_{edge \in t} weight(edge)$$

$$h(q, t) = \prod_{E_i \notin t} \max_{entity \in E_i} C(q, entity)$$

The score of an incomplete tuple  $t$  is estimated by:

$$Score'(q, t) = g(q, t) \times h(q, t)$$

The system aims to find tuples with top- $k$  scores. The hypothesis function  $h(q, t)$  is admissible since the weight of each edge is always less equal than 1 and the function utilize the max confidence of entities in each type. As a result, given the tuple  $t'$  expanded from  $t$ , the following equation holds and the algorithm satisfies the property of  $A^*$ .

$$g(q, t') \times h(q, t') \geq g(q, t) \times h(q, t)$$



---

**Algorithm 3** Entity search

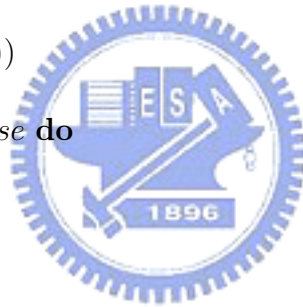
---

**Input:** Query  $q$  and scheme  $S = \{E_1, E_2, \dots, E_n\}$

**Output:** Ranked entity tuples

```
1: Get  $l_q$  and  $l_{E_i}, \forall E_i \in S$  from the index
2:  $tuples$  = priority queue ranked by  $Score'(q, t)$ 
3: for all  $E_i \in S$  do
4:    $l'_i = \text{join}(l_q, l_{E_i})$ 
5:   for all  $e \in l'_i$  do
6:      $tuples.add(\text{new tuple}(e))$ 
7:   while  $tuples.empty() = false$  do
8:      $t = tuples.pop()$ 
9:     if  $size(t) = n$  then
10:       $results.add(t)$ 
11:     if  $size(results) = k$  then
12:       return
13:      $T' = \text{expand}(t)$ 
14:     for all  $t' \in T'$  do
15:        $tuples.add(t')$ 
```

---



# Chapter 5

## Experiments

We evaluate the framework by building a prototype system using 0.5M web pages under `uiuc.edu` domain crawled on April, 2008. We use the GATE framework to annotate entities by both statistical and dictionary-based methods. The number of entities annotated including false-positive entities are shown in Table 5.1. The entity graph is constructed using the Hadoop platform and is stored in a cluster of memory cache to enable fast look-up. The entity index is distributed on two machines and the  $C(q, e)$  function is computed by aggregating the index look-up in a portal machine to show that the system can be linearly scale out. After aggregating the  $C(q, e)$  function, the machine performs tuple composition by querying the cluster which stores the entity graph.

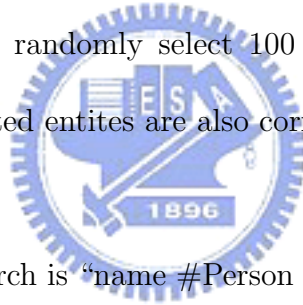
In the entity graph construction process, we set each element in the threshold matrix and each element in  $M_{null}$  as 0.2 since it is quite reasonable to filter out noise. Moreover, to isolate the annotation error from performance evaluation, we do not consider entity tuples containing any false-positive annotated entities.

Type	#Entities
Person Name	632,511
Email	34,408
Phone Number	29,731
Zipcode	38,656
State	103
University	85

Table 5.1: Number of entities annotated

## 5.1 Precision

In the precision evaluation, we randomly select 100 person names which is correctly annotated. Moreover, their related entities are also correctly annotated at least one time in the corpus.



The query used in entity search is “name #Person #Email #Phone”, which searches the complete person name, email, and phone number related to a person name string.

We compare the web puzzle powered entity search with the other two competitors:

- **keyword search:** The keyword search is implemented based on the open source search framework Lucene[2] and all pages are boosted by their Pagerank[15]. The query used in keyword search is “name AND (phone OR email)”. In the keyword search, we suppose that the user will not miss any entities appear in the result pages and combine the entities he has seen correctly.
- **Naive Count:** Simply count the number of co-occurrence of entities in the inverted

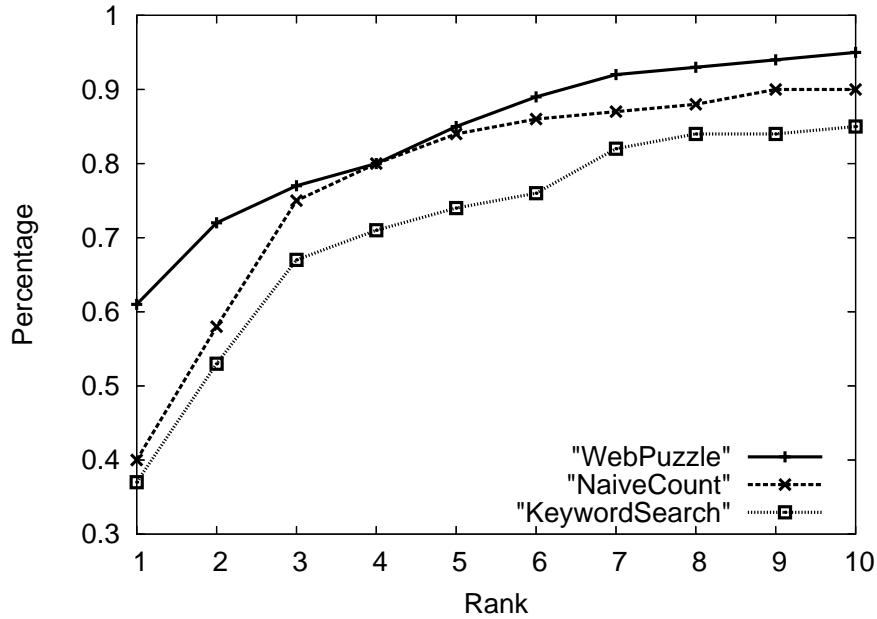


Figure 5.1: Satisfied query percentage under various ranks

list matching the query. Since there is no composition in naive counting, we suppose the user can compose the entities in the top ranks correctly. The rank is considered as the maximum rank among the three entities in the query.

In Figure 5.1, we compare the number of search results viewed in order to satisfy the query. The keyword-based search is sometimes misled by the other phone number or e-mail texts such as contact information of the organization. The same problem occurs in the naive counting approach since the common shared entities appears more frequently than the personal phone number or email. The web puzzle framework utilize the conditional probability in tuple composition thus avoid the problem.

In Figure 5.2, we compare the number of different entities viewed in the top results to satisfy the query. It shows that keyword search will show more ambiguous entities comparing to the web puzzle framework because there are some directory listing pages that list personal information for a lot of people. The naive counting beats the web

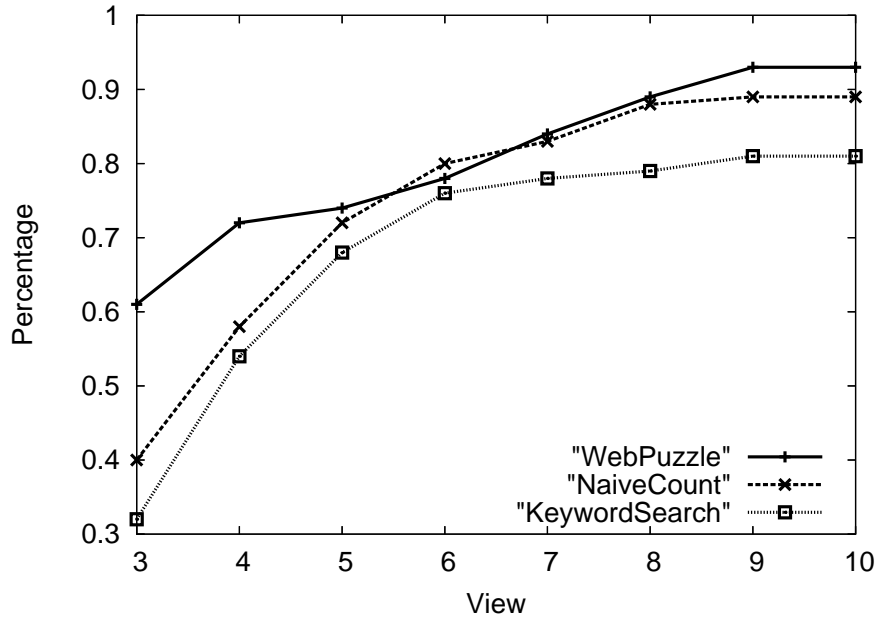


Figure 5.2: Satisfied query percentage under various number of entity viewed

puzzle framework in rank 6 because the naive counting approach consider only entities appearing in pages that also contains the query keywords; thus, less noise was introduced to the result.

Although we suppose that the user can compose all entities seem in both keyword search and naive counting approaches, it is impossible without spending a sufficient amount of time. In addition to the precision, the web puzzle framework satisfies more queries in top ten.

## 5.2 Study of Tuple Size

To evaluate the system with different tuple size, we re-use the 100 person names in the previous experiment and issue the query “name #Email #Person #Phone #University #State #Zipcode” by adding one entity type at a time. The result is shown in Figure

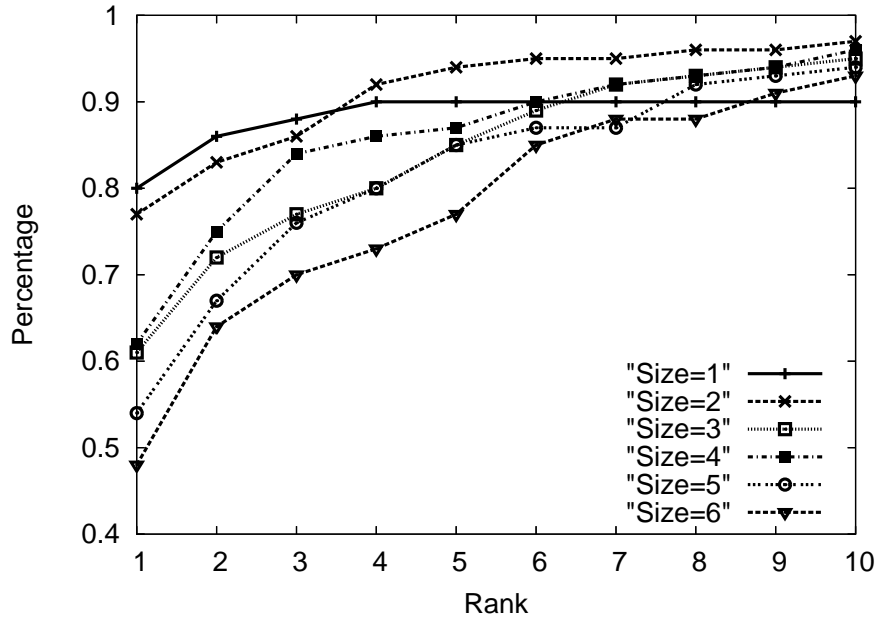


Figure 5.3: Satisfied query percentage under various ranks for different size of tuple scheme

### 5.3.

Querying for less entity types gives better precision in top results; however, when it goes to top-10, there is no much different. 4 of the 100 query cases is shown in Table 5.1. It shows that querying for more entity types may mislead the tuple composition as in case 12 and 24. However, in case 7 and 16, it shows that some entity may not be directly found by the keywords without traversing the entity graph. In summary, the accuracy of overall result is stable even the number of entity type required increases.

## 5.3 Tuple Listing

In the third experiment, we exam the system's ability to list different entity tuples given a fuzzy concept. 10 of 100 common surnames in United States are selected and make sure

C\TS	1	2	3	4	5	6
7	X	8	7	3	2	X
12	1	1	5	9	8	4
16	X	3	6	3	8	6
24	1	2	9	6	10	7

Table 5.2: Ranking for case\tupleSize

there are at least 5 different  $\{\#Person \#Phone \#Email\}$  tuples related to the surname in the corpus. The query “surname #Person:surname #Phone #Email” is issued to the prototype system and we count the number of correct tuples referring to different real-world instances in top ranks.

Table 5.2 shows the detailed result and Figure 5.4 shows the average number of different correct entity tuples in top ranks. The experiment shows that the web puzzle framework is able to show different tuples without introducing extra noises.

Case\Rank	1	2	3	4	5	6	7	8	9	10
Smith	1	2	3	4	4	5	5	6	7	7
Johnson	1	2	3	4	5	5	5	5	5	5
Williams	1	2	2	3	4	4	4	5	5	5
Brown	1	1	2	2	3	4	4	5	5	5
Miller	1	2	2	3	4	4	4	4	4	5
Anderson	1	1	1	2	3	4	4	5	6	6
Taylor	1	1	2	2	3	4	5	6	6	7
Thomas	1	2	3	4	5	5	6	7	8	9
Martin	1	2	2	3	3	4	5	6	7	7
Scott	1	2	3	4	5	6	7	7	7	8

Table 5.3: Different Tuples for Common Surnames

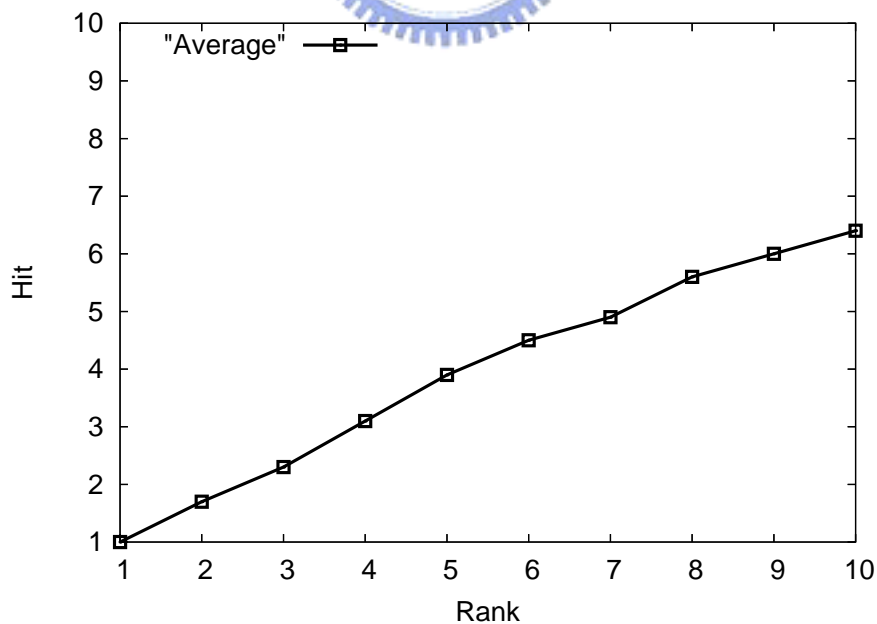


Figure 5.4: Number of valid tuples under various ranks



# Chapter 6

## Related Works

The objective of the web puzzle framework is to search entity tuples that match the tuple scheme and the given concept from the web. The system provide the possibility to search the web with entity view, which matches the emerging trend towards searching with entity and relationship over unstructured corpus collection [7][23]. Unlike most similar works, our system simulate the process that human discover knowledge by utilizing the entity graph. In this section, similar works are compared with our system in both conceptual model and system design aspects.

Like most entity-based search system, entity annotation is the basic component in the web puzzle framework. Information extraction techniques have been widely studied and [4][8][3] have summarized the state of the art. There are many information extraction works in the web domain [13][18][22][19]. Open source frameworks such as GATES[9], which we used in our system, and UIMA[10] are public available. The web puzzle framework utilizes the excellent existed information extraction works and further expand them to build the entity graph to support entity search.

Question answering [12][5][16] system are widely applied for knowledge discovery. The AskMSR QA system [12] rank answers based on entity occurrence. Closeness to keywords is the main criteria for the Mulder system [5] to rank their answer candidates; the system also clustering similar candidates for voting to strength the precision. The Aranea system [16] uses a scoring function from the candidate frequency and keyword inverse document frequency in the candidates. However, the question answering system still provides page-based model rather than search entity holistically.

To support an “object view”, ExDB [17] offers a SQL-like query language to extract singular objects and binary predicates of the web. Libra [20] model the web objects as records with attributes, which is similar to our presentation. However, due to the information granularity difference, its language retrieval model is quite different from ours. The above approaches, which tried to search fully extracted entities and relationships over the web, is heavily depends on the precision of entity and relation recognition while our system utilize the redundancy nature of the web and a degree of information extraction error can be tolerant. Thus, the performance of web puzzle framework is mainly depends on the large-scale analysis rather than precision of individual document.

Efficient index structure is an important part to support on-line entity-based search. BE [6] utilize a special index, “neighborhood index”, to efficient processing phrase queries, “interleaved phrase”. In the entity index, we utilize a similar structure to inverted keyword index; as a result, we can integrate the entity search function with existed system with less cost. Chakrabarti et al. [14] introduce a class of text proximity queries and study scoring function and index structure optimization for such queries. Local proximity information within document is the main concern of its scoring function while our

system focus on global relation among entities which is more suitable for the information distribution of web objects.

ObjectFinder [24] compose “object” from a collection of documents that are related with it. As what we do in our system, it aggregate object score over document scores. However, our approach is more entity-centric, where document scores is aggregated per entity relation. The score is mainly based on its structure and the entity in the tuple.



# Chapter 7

## Conclusions

In this paper, we model the entity search problem as the web puzzle problem. By build the entity graph and entity index, the framework extend entity search to provide better accuracy and data coverage. The design of the framework and internal algorithm supports scale horizontally which meets the requirement of a search service. The experiments show that our framework is able to discover entities which do not directly co-occur with the keyword query. Moreover, the system is able to list entity tuples related to a fuzzy concept based on the overall impression of the web.

# Bibliography

- [1] Hadoop, <http://hadoop.apache.org/core/>.
- [2] Lucene, <http://lucene.apache.org/>.
- [3] R. Ramakrishnan A. Doan and S. Vaithyanathan. Managing information extraction: state of the art and research directions (tutorial). In *SIGKDD*, 2003.
- [4] E. Agichtein and S. Sarawagi. Scalable information extraction and integration (tutorial). In *SIGKDD*, 2006.
- [5] O. Etzioni C. C. T. Kwok and D. S. Weld. Scaling question answering to the web. In *WWW*, pages 150–161, 2001.
- [6] M. Cafarella and O. Etzioni. A search engine for large-corpus language applications. In *WWW*, 2005.
- [7] S. Chakrabarti. Breaking through the syntax barrier: searching with entities and relations. In *ECML*, pages 9–16, 2004.
- [8] W. Cohen. Information extraction (tutorial). In *SIGKDD*, 2003.
- [9] H. Cunningham, D. Maynard, K. Bontcheva, and V. Tablan. Gate: A framework and graphical development environment for robust nlp tools and applications. In

*Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics*, 2002.

- [10] F. David and L. Adam. Uima: an architectural approach to unstructured information processing in the corporate research environment. *IEEE Nat. Lang. Eng.*, 10:327–348, 2004.
- [11] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. In *OSDI*, pages 137–150, 2004.
- [12] S. Dumais E. Brill and M. Banko. An analysis of the askmsr question-answering system. In *EMNLP*, 2002.
- [13] S. D. et al. Semtag and seeker: bootstrapping the semantic web via automated semantic annotation. In *WWW*, 2003.
- [14] J. Han K. Chakrabarti, V. Ganti and D. Xin. Ranking objects based on relationships. In *SIGMOD*, 2006.
- [15] R. Motwani L. Page, S.y Brin and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project, 1998.
- [16] J. J. Lin and B. Katz. Question answering from the web using knowledge annotation and knowledge mining techniques. In *CIKM*, pages 116–123, 2003.
- [17] D. Suci M. Cafarella, C. Re and O. Etzioni. Structured querying of web text data: a technical challenge. In *CIDR*, 2007.

- [18] D. Downey S. Kok A.-M. Popescu T. Shaked S. Soderland D. S. Weld O. Etzioni, M. Cafarella and A. Yates. Web-scale information extraction in knowitall. In *WWW*, 2004.
- [19] F. Chen A. Doan P. DeRose, W. Shen and R. Ramakrishnan. Building structured web community portals: a top-down, compositional, and incremental approach. In *VLDB*, pages 399–410, 2007.
- [20] K. Puniyani S. Chakrabarti and S. Das. Optimizing scoring functions and indexes for proximity search in type-annotated corpora. In *WWW*, pages 717–726, 2006.
- [21] X. Yan T. Cheng and K. C.-C. Chang. Entityrank: searching entities directly and holistically. In *VLDB*, pages 387–398, 2007.
- [22] S. Raghavan S. Vaithyanathan T. S. Jayram, R. Krishnamurthy and H. Zhu. Avatar information extraction system. *IEEE Data Eng. Bull.*, 29(1):40–48, 2006.
- [23] G. Weikum. Db & ir: both sides now. In *SIGMOD*, 2007.
- [24] S. Shi J.-R. Wen Z. Nie, Y. Ma and W.-Y. Ma. Web object retrieval. In *WWW*, 2007.