AIDE: Ad-hoc Intents Detection Engine over Query Logs

Yunliang Jiang[1] Hui-Ting Yang[2] Kevin Chen-Chuan Chang[1] Yi-Shin Chen[2]
[1]Department of Computer Science, University of Illinois at Urbana-Champaign, USA
[2]Department of Computer Science, National Tsing Hua University, Taiwan
{jiang8, kcchang}@illinois.edu, huiting.yang@cazoodle.com, yishin@cs.nthu.edu.tw

ABSTRACT

While keyword queries have become the "standard" query language of web search and many other database applications, their brevity and unstructuredness make it difficult to detect what users really want. In this demonstration, we aim to detect such hidden query intents, which we define as the frequent phrases that users co-ask with the query term, by exploring query logs. Toward building an online search system *AIDE*, we offer users the function to detect *general* and *unique* intents using arbitrary ad-hoc queries at run time. We will also demonstrate the effectiveness of the system which achieves indexing and searching over 14M MSN query log records.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Query formulation, Search process; H.3.5 [**On-line Information Services**]: Web-based services

General Terms

Design, Experimentation, Performance

Keywords

Ad-hoc, intent, indexing, general, unique

1. INTRODUCTION

As the web has evolved as the ultimate information repository of our time, keyword queries have become the means for expressing our information needs to tap into the wealth of online information. While easy and intuitive to use, however, the brevity and unstructuredness of keyword queries have made it difficult to interpret what users really want. For instance, given a query term "chicago", what does the user want? Say, user may be interested in finding out "NBA" activities in Chicago, thus with a query intent "NBA".

As a result, detecting the hidden intents behind query terms becomes crucial for many applications such as formulating queries, enhancing result ranking and matching advertisements. As our objective, the *AIDE* system– ad-hoc intent detection engine– would detect intents for arbitrary query terms by mining the query log information on the fly.

We define an *intent* for a query instance e (such as "chicago") as an *n-gram* phrase such as "nba" or "white sox" that frequently associate with e. Thus, in a query log corpus, if the query "chicago white sox" (or "white sox chicago") occurs repeatedly, we may intuitively recognize "white sox" (a bigram) as possibly an intent for query instance "chicago".

However, given a query instance e, the notion of intents is context dependent– It not only depends on what e is, but also on what "group" (or "category", "class") that e belongs to. Thus we detect two kinds of intents below:

General Intents: On the one hand, we may ask about "chicago"– as a *city*– may share *common* intents with other cities. That is, what intents do several cities like {"chicago", "new york", "los angeles"} have? The following phrases: *craiglist*, *hotels*, *real estate*, *airport* may be discovered as such intents. In this case, "query" is a set of query instances T. We thus refer to those intents common to a target group $T = \{t_1, \ldots, t_n\}$ of query instances as *general intents* for T.

Unique Intents: On the other hand, we are also interested in finding what "chicago"– as a *particular* city– may have as its own intents which distinguish it from others? That is, in the context of cities like {"chicago", "new york", "los angeles"}, what intents does "chicago" have characteristically? we may discover *cubs*, *bears*, *tribune*, *sun times*, as such intents. We refer to those intents unique to an instance t, in the context of a background group $B = \{b_1, \ldots, b_m\}$ of query instances as *unique intents* for t.

We observe the dual notions of intents share a conceptually unified abstraction, namely, ***comparative intents***, which determines intents by comparison to a given context. The unification form can be simply described as: given a target group T and a background group B where $T \subset B$. How can we find the intents of T, relative to B?

In the *AIDE* system, after inputting a group of query instances, users would receive a ranked list of general intents, or unique intents of one specific instance among the group. Such function can turn into a core module of current web search engine in many mentioned applications: given the input query q, by automatically extracting a group of entities with similar domain [1] and then detecting the intents, the search system can formulate queries (suggesting matching intents), optimize the search results (biasing toward results matching top intents), or directly use such intents for keyword bidding in advertisement.

Our system is novel and unique in several aspects:

• In contrast to high-level taxonomy view of "query intents" [2] which has been widely adopted, we view query intents as "interesting aspects" of query instances. Thus we

- **Given**: Query log L, with intent vocabulary Ω and query vocabulary Σ.
- **Input**: $s = \mathcal{CI}(T{=}\{t_1, \ldots, t_n\}, B{=}\{b_1, \ldots, b_m\})$, where $T \subset B \subset \Sigma$.
- **Result**: $\forall\, x \in \Omega$: Rank all intent x by computing $Score(x|T,B)$.

(**1**) $Score(x|T,B) = p_o \cdot \log \frac{p_o}{p_r}$, **where**

$$(2) \quad p_o \equiv p(x|T, L_B) = \prod_{t \in T} p(x|t, L_B) = \prod_{t \in T} \frac{\mathcal{F}(x|t)+1}{\mathcal{F}(t)}$$

$$(3) \quad p_r \equiv p(x|T, L'_B) = \prod_{t \in T} p(x|L'_B) \times p(t|L'_B) = \prod_{t \in T} \frac{\mathcal{F}(x|B)}{\mathcal{F}(B)} \times \frac{\mathcal{F}(t)}{\mathcal{F}(B)}$$

- **General Intent**: $\mathcal{GI}(T{=}\{t_1, \ldots, t_n\}) \equiv \mathcal{CI}(T, \Sigma)$
- **Unique Intent**: $\mathcal{UI}(t, B{=}\{b_1, \ldots, b_m\}) \equiv \mathcal{CI}(\{t\}, B)$

Figure 1: Complete model

exploit not only *fine-grained* but also *dynamic* intents. Further, our intents are not identified in isolation, but rather in the context with respect to comparable query instances.

• In contrast to many "offline" query log mining tasks, we aim to provide online search with ad-hoc input queries– the first such search system, to the best of our knowledge.

In the demonstration, on the basics of the comparative intents analysis, the *AIDE* system will offer users the function of detecting *general intents* and *unique intents* using arbitrary queries at run time.

2. SYSTEM ARCHITECTURE

In this section, firstly we will show the conceptual model of the *AIDE* system. Before this we introduce *intent vocabulary* Ω, which defines the universe of all *candidate intents* that we should consider, and *query vocabulary* Σ, which contains all the query instances users may ask for. In the demonstration, Ω and Σ are both considered as all the unigrams and bigrams that appear in the query log L.

Figure 1 shows the complete model of the problem. For each intent x in Ω, we use $score(x|T,B)$ to measure the importance, which is decided by two major factors:

• x would be a frequent phrase co-occurred with the target group T, which is measured by p_o.

• x would relatively infrequently co-occur with the whole background group B, which is measured by p_r.

According to the model, *general intents* $\mathcal{GI}(T)$ is a special case of $\mathcal{CI}(T,B)$ where the background group B equals to the *query vocabulary* Σ. *unique intents* $\mathcal{UI}(t,B)$ is the case while the target group T contains only one query instance t.

Now we will discuss the architecture of the *AIDE* system. The core components are shown in Figure 2.

2.1 Offline Processing

Query Log extraction

In the query log, each record contains multiple information: the user query, the searching time, the click-through URL and the user ID. In the *AIDE* system, we only use the "query" part to detect the intents. After extracting the query part, Query Log Extractor would also analyze each query by removing stop words and non-standard characters.

Pre-compute statistics

Assume query instance $e \in \Sigma$, as the principle of indexing, our purpose is to pre-compute necessary statistics for e. To compute $Score(x|T,B)$, for $T{=}\{t_1 ... t_n\}$, $B{=}\{b_1, ... b_m\}$, we need four types of "frequency" as described in Figure 1.

• *Target Freq.* $\forall t \in T$, we need to acquire the value $\mathcal{F}(t)$,

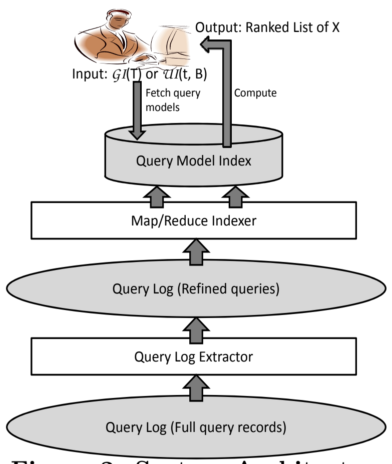

Output: Ranked List of X
Input: $\mathcal{GI}$(T) or $\mathcal{UI}$(t, B)
Fetch query models
Compute
Query Model Index
Map/Reduce Indexer
Query Log (Refined queries)
Query Log Extractor
Query Log (Full query records)

Figure 2: System Architecture

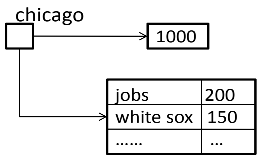


Figure 3: Query Model Index for 'chicago'

which represents the number of queries contain the query instance t.

• *Intent-Target Freq.* $\forall t \in T$, we need to acquire the value $\mathcal{F}(x|t)$, which means the number of queries containing the intent x and the query instance t.

• *Background Freq.* $\mathcal{F}(B)$ means the number of queries containing "any" background query instance b in B. To acquire it we need to do expensive union operations on each query set which contains b respectively. To simplify the problem we make an assumption based on the prior testing: in any B that users ask, each two b_i and b_j have so few overlaps that we can ignore. Thus we can approximately count: $\mathcal{F}(B) \approx |\sum_{b_i \in B} \mathcal{F}(b_i)|$

• *Intent-Background Freq.* $\mathcal{F}(x|B)$ means the number of queries containing one intent x and "any" background query instance b in B. Similar to above, $\mathcal{F}(x|B) \approx |\sum_{b_i \in B} \mathcal{F}(x|b_i)|$

In summary, for each $e \in \Sigma$, we need index the number of queries containing t. Simultaneously, we should index all e's co-occurring intent x_i and the number of queries which contain e and x_i.

Query Model Indexing

Having the above observation, for each $e \in \Sigma$: we will build *query model index* I_e to pre-store such information: the number of queries which contain e, each potential intent x_i which co-occurs with e, as well as the number of queries which contain e and x_i together. A snippet of *query model index* for "chicago" : $I_{chicago}$ is shown in Figure 3.

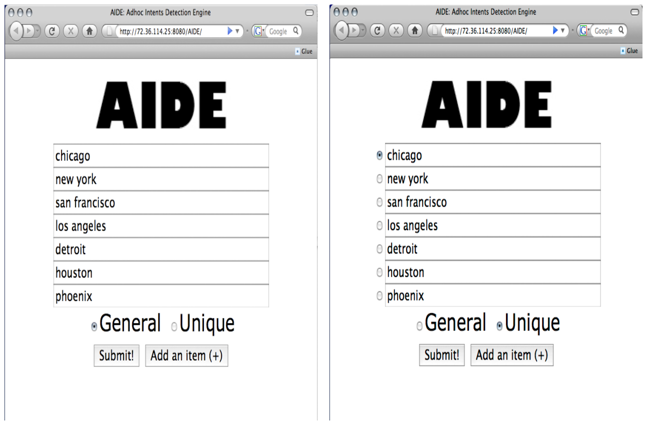
Map-Reduce Processing

Since *query vocabulary* Σ contains a large amount of instances, to build *query model index* for each $e \in \Sigma$ would be very expensive even offline. Thus we use Map-Reduce technique which supports parallel computations on clusters of computers to reduce the index space and time.

2.2 Online Computation

The online computing process will accept the arbitrary input and compute the score for each candidate intent x_i by fetching and combining the information from index.

For the conceptual model, given the target group T and


AIDE: Adhoc Intents Detection Engine
http://72.36.114.25:8080/AIDE/
AIDE
Submit!
Add an item (+)

chicago
new york
san francisco
los angeles
detroit
houston
phoenix
[✓ General] Unique

[✓ chicago]
new york
san francisco
los angeles
detroit
houston
phoenix
General [✓ Unique]

Figure 4: Input Interface

the background group B, firstly we would fetch query model index for each b in B ($T \subset B$) and collect all the items in each I_b as candidate intents. For each candidate intent x_i, we will compute $score(x_i|T, B)$ according to the formulas described in Figure 1. After all the candidate intents have been computed, we rank them and choose the top ones to be our results.

Specifically, in the case of *general intents*, the background group is the Σ. Since the whole query log L can be viewed as the set of queries containing Σ, computing p_r becomes different but more efficient. We have: $p_r = p(x|T, L') = \prod_{t \in T} \frac{\mathcal{F}(x)}{|L|} \cdot \frac{\mathcal{F}(t)}{|L|}$. All the factors could be fetched directly from each query model without additional combining. *Unique intents* is a special case of conceptual model while T contains only one query instance t, we can directly use the above algorithm to solve the unique intents problem.

3. DEMONSTRATION

Prototype System

We perform the demonstration by using a set of query log from Microsoft Live Search. This dataset is a sample over 4 months (Aug. 2007 to Nov. 2007), and contains 14M search impressions.

The whole system is built on six machines, each with the following environment: CPU: Pentium4 2.8G, RAM: 1G. We use Hadoop(version 0.19) to compute statistics and Lucene(version 2.3) to index all the query models.

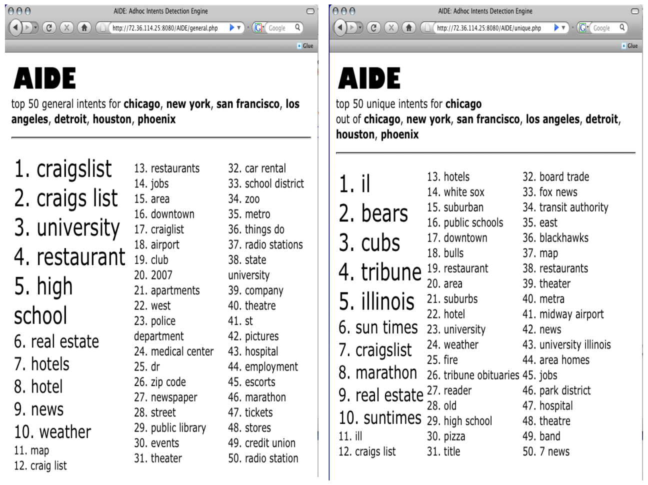
System Interface

As mentioned before, in the *AIDE* system, we would handle arbitrary queries for *general intents* and *unique intents* detection. Figure 4 shows users input interface.

The system offers users two functions: *general intent search* (left part) and *unique intent search* (right part). As shown in the left part of Figure 4, users can input a group of query instances T such as {“chicago”, “new york”, “san francisco”, etc} using the “Add an item” button. Similarly, in the right part, users can input a group of query instances B and choose one specific instance t such as “chicago” as the target instance using the radio box.

After users click the “Submit” button to offer the search task, the *AIDE* system would compute and return the top general intents or unique intents. Figure 5 shows the output interface which are the results for the two search tasks in Figure 4 respectively.

As shown in Figure 5, for both two search tasks, we will show top 50 intents according to the score and highlight the top 10 on which users focus mostly.


AIDE: Adhoc Intents Detection Engine
http://72.36.114.25:8080/AIDE/general.php
http://72.36.114.25:8080/AIDE/unique.php
AIDE

top 50 general intents for **chicago**, **new york**, **san francisco**, **los angeles**, **detroit**, **houston**, **phoenix**

1. craigslist
2. craigs list
3. university
4. restaurant
5. high school
6. real estate
7. hotels
8. hotel
9. news
10. weather
11. map
12. craig list
13. restaurants
14. jobs
15. area
16. downtown
17. craiglist
18. airport
19. club
20. 2007
21. apartments
22. west
23. police department
24. medical center
25. dr
26. zip code
27. newspaper
28. street
29. public library
30. events
31. theater
32. car rental
33. school district
34. zoo
35. metro
36. things do
37. radio stations
38. state university
39. company
40. theatre
41. st
42. pictures
43. hospital
44. employment
45. escorts
46. marathon
47. tickets
48. stores
49. credit union
50. radio station

top 50 unique intents for **chicago**
out of **chicago**, **new york**, **san francisco**, **los angeles**, **detroit**, **houston**, **phoenix**

1. il
2. bears
3. cubs
4. tribune
5. illinois
6. sun times
7. craigslist
8. marathon
9. real estate
10. suntimes
11. ill
12. craigs list
13. hotels
14. white sox
15. suburban
16. public schools
17. downtown
18. bulls
19. restaurant
20. area
21. suburbs
22. hotel
23. university
24. weather
25. fire
26. tribune obituaries
27. reader
28. old
29. high school
30. pizza
31. title
32. board trade
33. fox news
34. transit authority
35. east
36. blackhawks
37. map
38. restaurants
39. theater
40. metra
41. midway airport
42. news
43. university illinois
44. area homes
45. jobs
46. park district
47. hospital
48. theatre
49. band
50. 7 news

Figure 5: Output Interface

• **Query 1**: **chicago**, new york, san francisco, los angeles, detroit, houston, phoenix **General Intents**: craigslist, craigs list, university, restaurant, high school **Unique Intents**: il, bears, cubs, tribune, illinois
• **Query 2**: **turkey**, fish, rice, milk **General Intents**: recipes, recipe, pictures, cooking, wild **Unique Intents**: thanksgiving, trot, wild, ground, calories
• **Query 3**: **Purdue University**, Stanford University, MIT, Harvard University **General Intents**: athletics, libraries, press, faculty, library **Unique Intents**: calumet, indiana, owl english, west lafayette, online writing

Table 1: Example search tasks

Demo Scenarios

In our demonstration, users can input arbitrary query instances to detect intents. However, users are more likely to detect instances that share a similar concept(domain). For example, a group of city names or university names. In Table 1, we show several search samples and top 5 results. It is obvious that the samples focus on three domains: city, food and university. The query instances are from reference [3] which deals with a problem similar to finding general intents of a particular category, but in an offline mining mode.

As shown in Table 1, for general intent search, the “Query” part shows the group of query instances T. For unique intent search, it refers to the background group B and the specific instance t is in bold font. From the table we can observe that the *AIDE* system is highly effective: most of the top 5 results are reasonable intents that users may pursue.

4. REFERENCES

[1] Oren Etzioni, Michael Cafarella, Doug Downey, Stanley Kok, Ana-Maria Popescu, Tal Shaked, Stephen Soderland, Daniel S. Weld, and Alexander Yates. Web-scale information extraction in knowitall: (preliminary results). In *WWW*, pages 100–110, 2004.

[2] Andrei Broder. A taxonomy of web search. *SIGIR Forum*, 36(2), 2002.

[3] Marius Pasca. Organizing and searching the world wide web of facts – step two: harnessing the wisdom of the crowds. In *WWW*, pages 101–110, 2007.