Building Taxonomy of Web Search Intents for Name Entity Queries

Xiaoxin Yin
Microsoft Research
One Microsoft Way
Redmond, WA 98052
xyin@microsoft.com

Sarthak Shah
Microsoft Corp.
One Microsoft Way
Redmond, WA 98052
sarthaks@microsoft.com

ABSTRACT

A significant portion of web search queries are name entity queries. The major search engines have been exploring various ways to provide better user experiences for name entity queries, such as showing "search tasks" (Bing search) and showing direct answers (Yahoo!, Kosmix). In order to provide the search tasks or direct answers that can satisfy most popular user intents, we need to capture these intents, together with relationships between them.

In this paper we propose an approach for building a hierarchical taxonomy of the generic search intents for a class of name entities (e.g., musicians or cities). The proposed approach can find phrases representing generic intents from user queries, and organize these phrases into a tree, so that phrases indicating equivalent or similar meanings are on the same node, and the parent-child relationships of tree nodes represent the relationships between search intents and their sub-intents. Three different methods are proposed for tree building, which are based on directed maximum spanning tree, hierarchical agglomerative clustering, and pachinko allocation model. Our approaches are purely based on search logs, and do not utilize any existing taxonomies such as Wikipedia. With the evaluation by human judges (via Mechanical Turk), it is shown that our approaches can build trees of phrases that capture the relationships between important search intents.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval – *search process.*

General Terms

Algorithms, Measurement, Experimentation.

Keywords

Web search intent, query clustering.

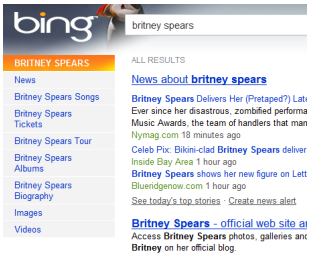
1. INTRODUCTION

Until a few years ago, most major search engines return only ten result snippets to user, and let the user find useful results by reading the snippets. Although this has been very successful, it costs much user effort in reading snippets, and it is not always possible for the search engine to accurately capture the user's intent when it is not clearly indicated in the query. This problem is more obvious for *name entity queries*, since different users may search for different aspects of a name entity using the same query, and it is very difficult for the search engine to infer the exact search intent. Name entity queries are a most popular type of queries. According to an internal study of Microsoft, at least 20-30% of queries submitted to Bing search are simply name entities, and it is reported 71% of queries contain name entities [14]. For simplicity we will use the word "entity" to refer to name entity.

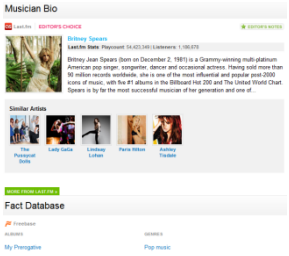
Recently there are some new methods used by different search engines to help users find the right information faster and more effectively for entity queries. The first method is to provide relevant "search tasks" (Bing search). For example, when the user searches for an entity, Bing usually shows several popular tasks for this entity. As shown in Figure 1(a), for query {Britney Spears}[1] Bing shows five tasks: Songs, Tickets, Tour, Albums, and Biography, together with three result snippets for each task. The tasks are all generic ones for a certain class of entities (e.g., musicians), which are found through mining the search logs. The second method is to show "direct answers" for what the user might be searching for, which saves the efforts of inspecting result snippets. Kosmix and WolframAlpha show only such "direct answers", while Yahoo! and Google mix such information with regular results. Given the query {Britney Spears}, Kosmix shows her biography, similar artists, albums, genres, etc. (Figure 1(b)), and Yahoo! shows her image, videos, official site, songs, and links to albums, lyrics, photos and videos (Figure 1(c)).

Although these methods bring more convenience to users, there exist two limitations. First, the direct answers only work well for certain classes of entities. For example, for query {university of washington}, Yahoo! provides no direct answer, while Kosmix shows a simple list of direct answers without most popular query intents such as sports, admissions, etc.

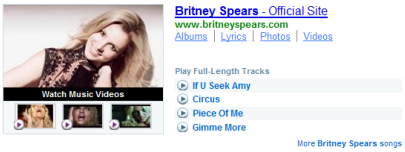
Second, these approaches put the major generic search intents



(a) Search tasks of Bing (left pane)



(b) Top part of Kosmix result page



(c) Top part of Yahoo! result page

Figure 1: The result pages containing search tasks or different aspects of information by Bing, Kosmix, and Yahoo!

[1] We use "{x}" to represent a web search query x.

WWW 2010, April 26-30, 2010, Raleigh, North Carolina, USA.
ACM 978-1-60558-799-8/10/04.

for a class of entities into a simple flat list. But these intents actually form a taxonomy, as some intents are related and some are sub-concepts of others. Take Britney Spears (or any musician) as an example. "Albums", "songs", "lyrics", and "music videos" are all about her music, "biography" and "profile" are about information of her life and career, and "tours" and "tickets" are about her concerts. Showing a flat list of search tasks or direct answers that mix up different aspects of information makes it difficult for users to find what he wants or get an overview of the entity.

The purpose of our study is to organize the generic search intents for a class of entities into a taxonomy according to the relationship between different intents. A class of entities is a set of entities that are usually considered to be of the same type, such as musicians, movies, car models, cities, etc. We work on classes of entities instead of individual ones, because the sparseness and noises in the data prevent us from accurately inferring the relationships between intents of different queries involving a single entity. By aggregating data involving many entities we can better capture the relationships between intents. An example taxonomy of major generic search intents for musicians is shown in Figure 2, which is generated by our approach.

Our work can help search engines in three ways. First, it can help to organize the search tasks by selecting tasks for representative intents and avoiding redundancy. With the taxonomy of intents we can also easily create a "tree of search tasks", so that a user can navigate in the tree to find the right task. Second, our work can help to better organize the direct answers, so that their layout is consistent with the taxonomy of user intents. The direct answers can also be organized into a tree according to the taxonomy, whose nodes can be dynamically shown to users when necessary. Third, our work can help web developers and users better understand queries containing name entities, and provide an overview of different search intents for a class of entities.

Outline and contributions

Baeza-Yates et al. propose the concept of query relationships, including equivalence, IS-A, and overlapping relationships [2]. However, only equivalence relationship is studied in [2], which is also studied in [3][5][24]. In this paper we study both equivalence and IS-A relationships between search intents, and how to build taxonomy of intents based on such relationships.

The first challenge is how to identify generic search intents for a class of entities. For each class of entities, we can find the generic search intents from user queries. For example, if many users search for {[musician] lyrics} for many different musicians, we can know "lyrics" is an important search intent for musicians.

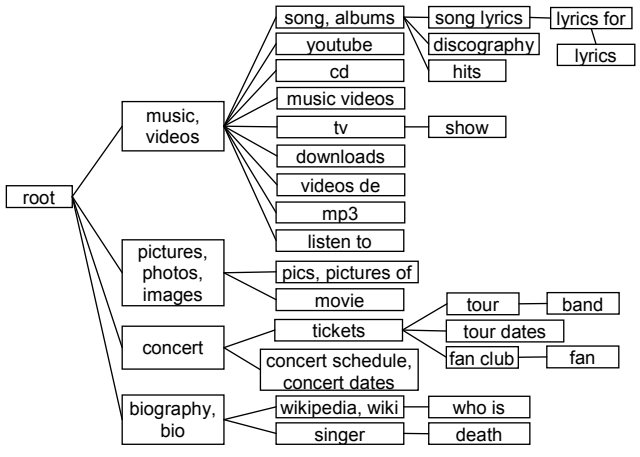


Figure 2: Result intent tree of musicians (most popular phrases only)

There are usually many users explicitly indicating their search intents in queries, and thus we can capture almost every major generic search intent for a class of entities. We represent such generic intents using words and phrases co-appearing with entities in user queries. If a word or phrase co-appears with a number of entities in user queries (e.g., "lyrics", "biography"), it usually represents a generic intent, and we call it an "*intent phrase*".

The second challenge is how to infer the relationship between two intent phrases. Some intent phrases carry the same intent, such as "pictures", "pics", "photos", "images" in queries for musicians. Some intents are sub-concepts of some other intents. For example, "wallpapers" are a type of "pictures", and "wikipedia" is source of "biography". We propose a method for inferring the relationship between two intent phrases based on user clicks, which indicate their intents. Although it is often difficult to accurately infer the relationship between two queries, there are usually many entities that co-appear with two intent phrases in queries, and we can aggregate the relationships between these queries to infer the relationship between the two intent phrases. Our experiments show such aggregation improves accuracy significantly.

The third challenge is how to organize intent phrases into a tree, so that each node contains intent phrases corresponding to the same intent, and the child nodes of a node correspond to sub-concepts of this node. We propose three algorithms for this task. The first algorithm is based on Directed Maximum Spanning Tree [6][10]. The second algorithm is adapted from Hierarchical Agglomerative Clustering [8][21]. The third algorithm is based on Pachinko allocation models [18] for building hierarchical topic models for documents. These algorithms are evaluated with human-labeled data on ten classes of entities.

The rest of this paper is organized as follows. We discuss related work in Section 2. Section 3 presents the problem formulation, and Section 4 describes how to infer the relationships between intent phrases. We present approaches for building intent trees in Section 5, and empirical study in Section 6. This study is concluded in Section 7.

2. RELATED WORK

There are some studies using existing taxonomy to generate related queries, or organize queries or information into an existing taxonomy. Ariel et al. [11] utilizes query-click logs and the taxonomy of Open Directory Project (ODP, www.dmoz.org) to generate related keywords for advertisers. In [20] Paşca and Alfonseca utilize WordNet and Wikipedia to organize the information extracted from Web into a hierarchy. Hu et al. [16] identify search intents of queries by considering Wikipedia categories and articles as possible search intents. Our approach is different from above approaches as we do not utilize any existing, manually edited taxonomies. This is because these taxonomies are very different from user intents. For example, the ODP taxonomy for "Music" is shown in Figure 3. Neither this taxonomy, nor the taxonomies from Wikipedia or WordNet, can be used to represent the search intents for musicians. Therefore, we purely rely on user behaviors to build hierarchies of generic search intents.

Baeza-Yates et al. study relationships between queries [2], in which "IS-A" relationship is mentioned but not studied or computed. Wen et al. [24] study query clustering by their similarities. In [4] Chuang et al. use hierarchical agglomerative clustering to organize queries into hierarchical clusters. These papers study how to organize queries using their similarities. But they do not infer the "IS-A" relationships between queries or search intents, and do not build taxonomies that reflect such relationships, which are the goals of our study.

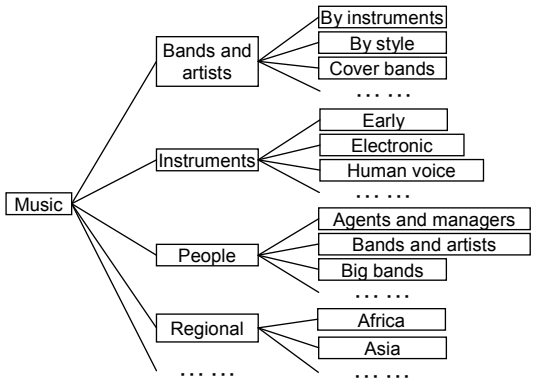


Figure 3: ODP taxonomy for music

Another related category of studies is how to find related queries as query suggestions, which has been studied from may perspectives. Cui et al. [7] uses the clicked URLs of queries to find related queries. In [5] Chien and Immorlica find related queries using the temporal query frequency information. Cao et al. [3] first group queries into clusters in an offline process, and then utilize query sequences to find related queries. In [22] Wang et al. propose an approach for generating the broad intent aspects for query suggestions, based on search session data. In [19] Paşca studies extracting different classes of entities, and top attributes for each class. Our study is different from these studies because we study a different problem of building taxonomy of query intents. We infer relationships between generic intents for a class of entities, which is more feasible than inferring relationships between queries, and does not suffer from ambiguity of keywords.

3. PROBLEM FORMULATION

Our study is about the generic intents on certain classes of entities. A class of entities is a set of entities that people usually consider them to be of the same type. Examples include musicians, soccer athletes, presidents of U.S., movies, car manufacturers, cell phone models, superstores, TV networks, cities, etc. Almost every entity can be categorized into some classes. It is usually easy to get a class of entities. For most classes we can think about, there is corresponding categories or lists on Wikipedia, such as en.wikipedia.org/wiki/Category:Lists_of_musicians for musicians. Such lists can also be gathered by approaches such as [19].

Because a large portion of web search queries are entity queries and many others contain names of entities, improving user experiences for entity queries is very important for search engines. The goal of our study is to find generic search intents associated with a class of entities, identify relationships between different generic intents, and organize them into a hierarchical structure. Similar to studies on query suggestion [5][7][22], we represent a specific search intent by a query, such as {britney spears biography}.

We represent generic intents by *intent phrases*. For a query containing an entity name as a substring, we define the query excluding the entity name as an intent phrase. For example, for query {britney spears biography} with "britney spears" being an entity, "biography" is an intent phrase. We consider an intent phrase carrying the same generic intent no matter whether it appears before or after the entities in queries.

Different intent phrases may represent the same generic intent. Take intent phrases for musicians as an example. "Biography", "bio", "biography of" all represent the same generic intent. Some intent phrases may indicate generic intents that are sub-concepts of others. For example, "history", "life", and "death" represent sub-concepts of "biography".

For each class of entities, our approach performs three tasks: (1) Find all important intent phrases. (2) Through the query-click behaviors of users, determine the relationships between intent phrases, i.e., whether one intent phrase is equivalent to, unrelated to, or represents a sub-concept of another intent phrase. (3) Based on the relationships, organize the intent phrases into a hierarchical structure (like the one in Figure 2), so that each node represents a generic intent and contains all intent phrases for it, and the child nodes of each node represent sub-concepts of that node. The correctness of the intent trees will be evaluated by human judges.

Non-Goals

In this paper we do *not* study the following problems: (1) How to find a class of entities, which can usually be done through parsing Wikipedia categories/lists, or using automatic approaches such as that proposed by Paşca [19]. (2) Ranking search tasks or generic intents for a specific entity, which can be done by analyzing query frequencies, query refinements, and clicked URLs of queries involving a specific entity. Many existing approaches on ranking query suggestions can be applied here.

4. INTENT PHRASES AND THEIR RELATIONSHIPS

4.1 Intent Phrases

We hope to find all intent phrases that represent generic search intents involving entities of a certain class. For each generic intent (e.g., lyrics of a musician), there are usually many users expressing the intent explicitly in their queries, and thus we can capture most generic intents using all intent phrases. On the other hand, to make sure an intent phrase represents a generic intent for different entities, the intent phrase should appear with at least θ entities in queries ($\theta = 5$ in our study). We exclude a small set of *stop phrases* that do not indicate clear intents, such as "web", "online", "2009", and inappropriate phrases such as "nude" and "sex". There are 185 stop phrases, including misspelled terms. Table 1 shows the top intent phrases for some classes of entities, sorted by the number of entities they co-appear within queries. We can see they represent very important intents for each class of entities. Some of them are redundant (e.g., "jobs" and "employment"), and we will study how to infer relationships between intent phrases.

4.2 Relationships between Intent Phrases

Before defining the relationships between intent phrases, we need to first define the relationships between the intents of different queries. We say the intents of query q_1 belong to those of q_2, if the information (or web pages) that can satisfy the needs of q_1 can also satisfy the needs of q_2. For example, a user searching for {Seattle} is usually looking for tourism, general information, or government of Seattle. Therefore, the intents of queries like {Seattle tourism} and {Seattle government} belong to those of {Seattle}. Queries q_1 and q_2 are likely to be equivalent if the intents of each of them belong to those of the other.

The search intent of a query is inherently subjective and varies

Table 1: Top intent phrases for certain classes of entities

Actors	*Cities*	*Musicians*	*Universities*
actor	city	lyrics	library
photos	city of	music	employment
biography	news	youtube	jobs
pictures	real estate	wikipedia	bookstore
imdb	hospital	songs	address
bio	apartments	Wiki	athletics
wikipedia	jobs	discography	alumni
movies	map	biography	tuition

for different users. As shown by Lee et al. [17], the intent of a query can usually be indicated by clicks of the user. It is also proposed by Baeza-Yates et al. [2] that relationships between queries can be defined with clicked URLs. Therefore, we propose an approach for deciding the relationship between intents of two queries by their clicked URLs.

For a query q submitted by a user, a search engine returns a ranked list of result URLs. There are three possibilities for each result URL: (1) Clicked, (2) skipped, i.e., a URL ranked below it is clicked, and (3) neither clicked nor skipped. According to the studies of Joachims et al. [15], the assumption that a clicked URL is more relevant than a skipped URL can provide a large number of preferences between pairs of URLs with reasonable accuracy.

In order to determine the relationships between intents of different queries through their clicked URLs, we need to be able to estimate the relevance of a URL for a query based on the search logs. We adopt a model similar to the one proposed in [13]. Suppose we are given the query-click logs of a search engine in a period of time. We say a URL is viewed for a query if it is either clicked or skipped. Consider a query q in the logs, and a URL u that is viewed for q. Let $click(q, u)$ be the number of times a user clicks u for q, and $skip(q, u)$ be the number of times a user skips u for q. We define the relevance of u for q as

$$rel(q,u) = \frac{click(q,u)}{click(q,u) + skip(q,u) + \varepsilon}. \quad (1)$$

$rel(q, u)$ is the estimated probability for u being clicked when it is viewed for q. ε is a smoothing factor, which reflects the fact that we are not confident about the estimated relevance when we do not see many clicks/skips for a URL. (ε is set to 1.0 in our study.) As discussed in [13][23], the measure based on probability of being clicked for viewed URLs is quite resistant to positional bias.

For two queries q_1 and q_2, we want to estimate the degree of the search intents of q_1 being included in those of q_2. This is a challenging problem as search intent is implicitly implied by user behaviors. Because clicked URLs indicate query intents, distribution of clicks on the clicked URLs of a query is a good indicator of distribution of query intents. For example, the major clicked URLs of four queries involving "Seattle" are shown in Figure 4, together with the percentage of Bing search users clicking on each URL for each query. We can see the major user intents for query {Seattle} are about tourism, official site of city, and hotels/attractions/restaurants. Most users querying for {city of Seattle} and {Seattle tourism} click on URLs frequently clicked for query {Seattle}, which indicate the intents of {city of Seattle} and {Seattle tourism} are included in that of {Seattle}.

DEFINITION 1 (Relationship between intents of queries). Let $U(q)$ be the set of URLs clicked for q. For two queries q_1 and q_2, the degree of q_1's intents being included in q_2's is defined as

$$d(q_1 \subseteq q_2) = \frac{\sum_{u \in U(q_1)} click(q_1,u) \cdot rel(q_2,u)}{\sum_{u \in U(q_1)} click(q_1,u)}. \quad (2)$$

$d(q_1 \subseteq q_2)$ is the average relevance to q_2 of each clicked URL of q_1, and $d(q_1 \subseteq q_2) \in [0,1)$. If we could assume a user only clicks on URLs that satisfy his search intent and only skips URLs that do not, then $rel(q, u)$ is the estimated probability of a user who submits query q considers URL u to satisfy his intent. In this way $d(q_1 \subseteq q_2)$ is the estimated probability that a clicked URL for q_1 can satisfy the search intent of a user querying for q_2. This will be a good estimation of the probability that q_1's intents are included in q_2's intents, if each URL does not satisfy multiple different intents. In reality some URLs can satisfy different (but usually related) intents. For example, www.seattle.com contains information about Seattle's hotels, restaurants, attractions, etc., which may confuse our algorithm by mixing up two weakly related or even unrelated intents.

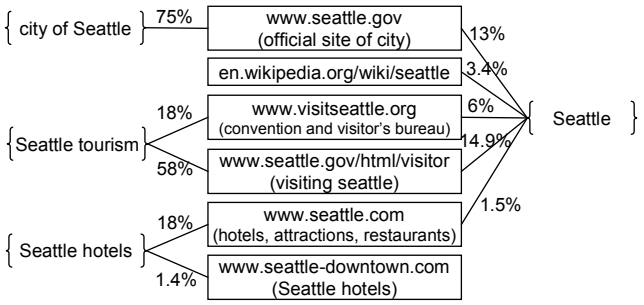


Figure 4: Percentage of users clicking each URL for four queries involving "Seattle"

Fortunately our goal is to identify the relationships between different intent phrases, and each intent phrase usually co-appears with many different entities in many queries. We can achieve higher accuracy by aggregating the relationships between many pairs of queries involving different entities.

DEFINITION 2 (Relationship between intent phrases). Let $f(q)$ be the query frequency of q, and let $e+w$ be the query consisted of entity e and intent phrase w (for simplicity we do not distinguish the queries in which w appears before e or after e). For a class of entities E, and two intent phrases w_1 and w_2, the degree that the search intents of w_1 are included in those of w_2 is defined as

$$d(w_1 \subseteq w_2) = \frac{\sum_{e \in E, f(e+w_1)>0} d(e+w_1 \subseteq e+w_2) \cdot f(e+w_1)}{\sum_{e \in E, f(e+w_1)>0} f(e+w_1)}. \quad (3)$$

$d(w_1 \subseteq w_2)$ is the weighted average degree of the intents of a query containing w_1 being included in those of a query containing w_2, if these two queries contain the same entity. $d(w_1 \subseteq w_2) \in [0,1)$ for any w_1 and w_2. If both $d(w_1 \subseteq w_2)$ and $d(w_2 \subseteq w_1)$ are high, then the intents of w_1 and w_2 should be very similar, and w_1 and w_2 are likely to be equivalent. If $d(w_1 \subseteq w_2)$ is high but $d(w_2 \subseteq w_1)$ is low, then w_1 should correspond to a sub-concept of w_2. We will use such relationships between intent phrases to organize the intent phrases into a tree.

5. ORGANIZING INTENT PHRASES

In this section we describe our approaches for organizing intent phrases into intent trees. Each node of the intent tree contains one or more intent phrases with same or similar search intents. The child nodes of a node n should contain intent phrases that represent sub-concepts of the intent phrase(s) of n. There is a root node that contains no intent phrase. An example intent tree has been shown in Figure 2.

We will present three approaches that solve this problem from different angles. The first approach is based on directed maximum spanning tree [10]. The second one is based on hierarchical agglomerative clustering [8]. The third one is a baseline approach based on Pachinko Allocation models [18].

5.1 Method Based on Directed MST

Given the relationships between different intent phrases for a class of entities, a good intent tree should have the following property: For all pairs of intent phrases (w_1, w_2) that are in the same node or w_1 in a child node of w_2, $\sum_{(w_1,w_2)} d(w_1 \subseteq w_2)$ is high. This problem is similar to finding a maximum spanning tree

(MST) in the directed graph containing a node for each intent phrase and an edge between each pair of related intent phrases. Directed MST has been studied in [6][10]. (These papers study how to find Minimum Spanning Tree, which is equivalent to finding Maximum Spanning Tree.) In order to apply such algorithms, we first define the "goodness" of an intent tree.

DEFINITION 3 (Score of intent tree). Given the intent phrases $w_1,\ldots,w_K$ for a class of entities, and an intent tree T with nodes $n_1,\ldots,n_L$ containing $w_1,\ldots,w_K$, we define the score of T as

$$s(T)=\sum_{\forall i,j,n_i\in child(n_j)}\max_{w_k\in W(n_i),w_l\in W(n_j)} d(w_k\subseteq w_l)+\sum_{\forall i,n_i\in child(root)}\beta + \sum_{i=1}^{L}\sum_{(w_k,w_l)\in MST(W(n_i))} d(w_k\subseteq w_l)+d(w_l\subseteq w_k) \quad (4)$$

child(n) is the child nodes of n. $W(n)$ is the intent phrases of node n. β is the base weight of each node. For a node n, if $d(w_i\subseteq w_j)<\beta$ for any w_i of n and w_j of some other node, then we do not consider n represent a sub-concept of any other node, and n should be a child of the root. *MST*($W(n)$) is the undirected maximum spanning tree constructed for a graph built from $W(n)$, with a node for each phrase w_k and an edge with weight $d(w_k\subseteq w_l)+d(w_l\subseteq w_k)$ between phrases w_k and w_l. □

$s(T)$ contains three terms. The first and second terms are for relationships between all pairs of parent and child nodes, and the third term is for the inherent consistency of phrases within each node. If we consider the graph with each intent phrase as a node, and the relationship between each two intent phrases as an edge, then $s(T)$ is defined based on a spanning tree on this graph. We will show that the intent tree with maximum score can be built by finding the directed maximum spanning tree in a graph.

We hope to build an intent tree from the graph of intent phrases, so that each tree node may contain multiple equivalent intent phrases. However, this is not allowed in the tree generated by a maximum spanning tree (MST) algorithm. To use a MST algorithm, we define a *semi intent tree*, which can be generated by an MST algorithm and can be converted into an intent tree. Each non-roof node in a semi intent tree contains a single intent phrase. Between each pair of nodes n_1 and n_2, there is zero or one directed edge. Each edge (n_1, n_2) is either a *belonging edge*, which indicates n_1 is a child of n_2; or an *equivalence edge*, which indicates n_1 and n_2 contain equivalent intent phrases. Each non-roof node has a single parent node. To convert a semi intent tree into an intent tree, we simply merge all nodes connected by equivalent edges into a single node.

Given the intent phrases $w_1,\ldots,w_K$ for a class of entities, we build a semi intent tree T' of nodes $n_1,\ldots,n_K$, with n_i containing phrase w_i. The score of T' is measured as

$$s(T')=\sum_{\forall i,j,n_i\in child(n_j)} d(w_i\subseteq w_j)+\sum_{\forall i,j,n_i\in equivalent(n_j)} d(w_i\subseteq w_j)+\sum_{\forall i,n_i\in child(root)}\beta \quad (5)$$

, where *equivalent*(n) is the equivalent nodes of n.

Building Optimal Semi Intent Tree

In order to build an optimal semi intent tree, we first convert the intent phrases and their relationships into a directed graph G. For each phrase w_i, we create a node n_i containing w_i. For each two phrases w_i and w_j with $d(w_i\subseteq w_j)>0$, we add a belonging edge from n_i to n_j with weight $d(w_i\subseteq w_j)$. We add two equivalent edges between n_i and n_j (in both directions) with weight $d(w_i\subseteq w_j)+d(w_j\subseteq w_i)$, if $d(w_1\subseteq w_2)>\beta$, $d(w_2\subseteq w_1)>\beta$, $d(w_1\subseteq w_2)>\alpha\cdot d(w_2\subseteq w_1)$, and $d(w_2\subseteq w_1)>\alpha\cdot d(w_1\subseteq w_2)$. α is a parameter between 0 and 1, and a larger α indicates more strict criterion for considering two nodes as equivalent. If $\alpha=1$, then no nodes are considered equivalent.

Given the graph G, we apply Edmond's algorithm [10] to build a directed maximum spanning tree. Edmond's algorithm is for building directed minimum spanning tree, which is equivalent to building directed maximum spanning tree. The main idea of Edmond's algorithm is as follows: (1) Select the edge with smallest weight that points to each node. (2) If no cycle formed, stop. (3) For each cycle formed, contract the nodes on the cycle into a pseudo node, and modify the weight of each edge pointing to this pseudo node according to weights of edges in the cycle. (4) Select the edge with smallest weight pointing to the pseudo node to replace another edge. (5) Go to step (2) with the new graph.

For a graph with n nodes and m edges, Edmond's algorithm takes $O(mn)$ time. A more efficient algorithm is presented in [12], which takes $O(m + n\cdot\log n)$ time. Since we only care about intent phrases that co-appear with at least a certain number of entities, the number of intent phrases is usually limited, and Edmond's algorithm is efficient enough. The maximum spanning tree on graph G is a semi intent tree, which can be easily converted into an intent tree by merging equivalent nodes.

Proof of Optimality

Here we give the sketch of the proof that the above algorithm can generate the optimal intent tree T^*, so that $s(T^*)\geq s(T)$ for any valid intent tree T.

LEMMA 1. For any intent tree T, there exists a semi intent tree T' so that $s(T)=s(T')$, and T' can be converted into T.

PROOF SKETCH. Select all pairs of phrases w_k and w_l so that $d(w_k\subseteq w_l)$ is involved in computing $s(T)$, and add edge between their corresponding nodes to construct a semi intent tree T' (adding equivalent edges for phrases of same node in T). In this way $s(T)=s(T')$ and T' can be converted into T. □

THEOREM 1. The intent tree T^* converted from the semi intent tree T'^* built by directed maximum spanning tree algorithm is the intent tree with maximum score.

PROOF. If T^* is not optimal, then there exists intent tree T so that $s(T)>s(T^*)$. By Lemma 1 there exists semi intent tree T' so that $s(T')>s(T'^*)$, which contradicts with T'^* being the maximum semi spanning tree. □

5.2 Method Based on Hierarchical Agglomerative Clustering

The above algorithm can find the optimal intent tree defined based on the maximum spanning tree. Its disadvantage is that the algorithm cannot merge nodes or edges during the procedure for finding maximum spanning tree. When we merge multiple equivalent intent phrases into a single node, or put one node as a child of another node, we should consider all intent phrases in the merged node and descendant nodes as a whole, and measure its overall relationships with other nodes. To accomplish this task, we modify the hierarchical agglomerative clustering (average-link) algorithm [8], so that it can run on a directed graph and perform two types of merging operations: (1) Putting one node as a child of another node, and (2) merging two nodes into one. Other hierarchical agglomerative clustering algorithms (Single-link and complete-link) are not used because they are highly vulnerable to noises, and the addition or removal of one edge can often significantly change the result clusters.

Our algorithm is described in Algorithm 1. Lines 1-6 are for initializing a graph whose nodes represent the intent phrases and edges represent relationships between intent phrases. There are two types of edges. A type 1 edge (n_i, n_j) indicates putting node n_i

as a child of n_j. A type 2 edge (n_i, n_j) indicates merging n_i as an equivalent node of n_j, which means merging the phrases and child nodes of n_i into those of n_j, and removing n_i. Let $d(n_i \subseteq n_j)$ represent the average degree of any intent phrase of n_i belongs to that of n_j. We add a type 1 edge from n_i to n_j if $d(n_i \subseteq n_j) > 0$, and add a type 2 edge if $r \cdot \left(d(n_i \subseteq n_j) + d(n_j \subseteq n_i)\right) > \max\left(d(n_i \subseteq n_j), d(n_j \subseteq n_i)\right)$, where r is a parameter between 0.5 and 1 controlling whether to merge two nodes into one or put one as a child of another. If r=0.5, we only put nodes as children of other nodes; if r=1, we only merge two nodes into one.

Lines 7-16 are for repeatedly merging nodes. In each iteration the edge with highest weight is selected. If it is a type 1 edge, we put one node as a child of the other; otherwise we merge the two nodes into one. After merging node n_i into n_j in either way, we need to update the weights of edges involving n_j. For every other node n_k, suppose the original weight of edge from n_j (or n_i) to n_k is d_{jk} (or d_{ik}). Let $f(w)$ be the total frequency of all queries consisted of intent phrase w and an entity in the specific class, and $f(n) = \sum_{w \in W(n)} f(w)$, where $W(n)$ is the set of phrases of n. The new weight from merged node n_j to n_k is an average of d_{ik} and d_{jk}, weighted by $f(n_i)$ and $f(n_j)$, as follows

$$d'_{jk} = \frac{f(n_i) \cdot d_{ik} + f(n_j) \cdot d_{jk}}{f(n_i) + f(n_j)}. \quad (6)$$

It can be easily proved that if d_{jk} (or d_{ik}) is the weighted average degree of any intent phrase in n_j (or n_i) belonging to that of n_k, then d'_{jk} is that for the newly merged node n_j. After updating the edges, the algorithm extracts the next edge with highest weight, and performs the merging.

Lines 17-19 create the final intent tree and output it.

This algorithm is different from traditional average-link algorithm in two aspects. First, in the input of our algorithm there can be two directed edges between each two objects, indicating bi-directional relationships. In contrast, average-link algorithm only handles simple, undirected similarity between objects. Second, our algorithm can put one node as a child node of another, besides merging two nodes into one as in average-link algorithm. Comparing to the DMST-based algorithm described in Section 5.1, this algorithm updates the relationships between different nodes after any node is modified. When deciding which nodes to be merged, it considers the relationship between all descendants of each node, and all intent phrases on these descendants. This makes it more accurate than the DMST-based algorithm according to our experiments, although it is a greedy algorithm.

It is sometimes necessary to select a representative intent phrase within a node. We define the query frequency of an intent phrase w as the sum of frequency of all queries consisted of w and an entity in the corresponding class. We find the intent phrase with highest query frequency is usually a good representative for a node, which is shown in the example intent trees in Section 6.4.

Algorithm 1. Average-link for intent trees

Input: Intent phrases $w_1,\ldots,w_K$ for a certain class of entities. $d(w_i \subseteq w_j)$ for $1 \le i, j \le K$.
Output: An intent tree whose nodes contain $w_1,\ldots,w_K$.
Procedure:
1: build graph G with a node n_i containing each phrase w_i
2: **for each** $d(w_i \subseteq w_j) > 0$
3: add edge (n_i, n_j) with weight=$d(w_i \subseteq w_j)$ and type=1
4: **if** $r \cdot \left(d(w_i \subseteq w_j) + d(w_j \subseteq w_i)\right) > \max\left(d(w_i \subseteq w_j), d(w_j \subseteq w_i)\right)$
5: add edge (n_i, n_j) with weight=$r \cdot \left(d(w_i \subseteq w_j) + d(w_j \subseteq w_i)\right)$ and type=2
6: $E \leftarrow$ set of all edges in G
7: **while** $\max_{e \in E} e.\text{weight} >$ min_sim
8: (n_i, n_j) $\leftarrow$ edge in E with maximum weight
9: **if** n_i or n_j has been merged with other nodes **then continue**
10: **if** (n_i, n_j).type=1
11: merge n_i as a child of n_j
12: **if** (n_i, n_j).type=2
13: merge the phrases and children of n_i as into those of n_j
14: remove n_i
15: **for each** node n_k with no parent
16: update (n_j, n_k).weight and (n_k, n_j).weight
17: create intent tree T with T.root being an empty node
18: put every node n with no parent as child of T.root
19: **output** T

5.3 Baseline Method Based on Pachinko Allocation models

The problem of generating hierarchical topic models has been studied in language modeling, and [18] presents a most popular approach called Pachinko Allocation models (PAM), which builds a hierarchy of topics from a set of documents, with each topic being a distribution of words. We adapt this approach for building intent trees, which serves as our baseline approach.

To apply topic modeling on intent phrases for a class of entities, we consider all intent phrases from queries that lead to clicks on a certain URL u to be "intent phrases of u", represented by $W(u)$. $W(u)$ is a multi-set, and the count of intent phrase w in $W(u)$ is the sum of *click*(q, u) for any query q consisted of w and an entity in the specific class. In language modeling it is assumed that keywords appearing in the same document are generated from the topic distribution this document. We make a similar assumption that each intent phrase in $W(u)$ is generated from the topic distribution of URL u. In this way we can convert each URL into a pseudo document that contains a bag of intent phrases, and can apply PAM on these pseudo documents to generate hierarchical topic models.

The output of PAM is a four-level DAG of topics, with root at the top level, super-topics at the second level, sub-topics at the third level, and intent phrases at the bottom level. Each intent phrase has a certain probability to belong to each sub-topic, and so does each sub-topic to each super-topic. To build an intent tree, we put each intent phrase as a child of the sub-topic that it belongs to with highest probability, and put each sub-topic as a child of a super-topic in the same way. A tree can be built in this way.

This tree is not an intent tree yet, because the super-topics in the output of PAM are not associated with any intent phrases. We assign an intent phrase to a super-topic if its probability of belonging to this super-topic is at least η times its total probability of belonging to any sub-topic, where $\eta > 1$. In this way an intent tree can be built using PAM, which has three levels of nodes: The root, the super-topics, and the sub-topics.

6. EXPERIMENTS

We now present the experimental evaluation of our approaches. All experiments are run on a Windows server with dual 2.66GHz Intel quad-core CPU and 32GB main memory, or on the PC cluster of Microsoft. All experiments on the Windows server are run with a single core.

6.1 Data Collection

We test the approaches on ten datasets, each containing a certain class of entities, as shown in Table 2. Nine of them are the set

Table 2: Data sources for ten classes of entities

Class of entity	*Num. Entity*	*Wikipedia categories or Web source*
car models	859	2000s_automobiles
U.S. clothing stores	103	clothing_retailers_of_the_united_states
film actors	19432	*_film_actors
musicians	21091	*_female_singers, *_male_singers, music_groups
restaurants	694	*_restaurants
universities / colleges	7191	universities_and_colleges_*
U.S. cities	246	www.mongabay.com/igapo/US.htm
U.S. presidents	57	presidents_of_the_united_states
U.S. retail companies	180	retail_companies_of_the_united_states
U.S. TV networks	276	american_television_networks

of entities from one or multiple Wikipedia categories or their descendent categories (* represent wildcard match), and one is from a web page. For the nine classes from Wikipedia, we ignore all entities with a Wikipedia disambiguation page.

We use the query click logs of Bing search (formerly as Live search) from 2008/01/01 to 2008/12/31. For each query that appears at least 6 times in the logs, we get the numbers of clicks and skips of each URL for this query. For each pair of queries with common clicked URL(s), we compute the degree of the intent of one query being included in the other, using MapReduce on a PC cluster. Based on this, we compute the relationships between the intents of each pair of intent phrases as in Section 4.2.

We compare the performances of three approaches: (1) DMST, – Method based on directed maximum spanning tree, (2) HAC, – method based on hierarchical agglomerative clustering, and (3) PAM, – method based on pachinko allocation model (baseline approach). For each class of entities, we use each approach to build an intent tree. The top 400 intent phrases that co-appear with most entities in search queries are considered for each class of entities. We ignore remaining intent phrases as they are usually obscure or do not carry generic intents.

6.2 Measurements by Mechanical Turk

Based on an intent tree, we say two intent phrases are either equivalent (if they belong to the same node), one belonging to another (if the node of one phrase is a descendent of the node of the other), or unrelated (if none of the above). In order to measure the accuracy of each approach, we test how accurate it is in judging the relationship between two queries containing the same entity and different intent phrases. We use Amazon Mechanical Turk [1] to judge the relationships between queries. For each pair of queries q_1 and q_2, a Mechanical Turk worker is asked to figure out their intents (we provide buttons for showing Google's results of each query and ask them to search on Google if they are not sure), and select one of the following four options about the intents of q_1 and q_2: (1) Unrelated, (2) q_1's intents belongs to those of q_2, (3) q_1's intents contain those of q_2, and (4) equivalent. We provide four to six examples for each type of cases, such as {tom hanks movies} and {how to contact tom hanks} (unrelated), {dayton ohio furnitures} and {furniture in ohio} (belongs), {go transit} and {go transit schedule} (contains), and {track income tax return} and {track tax return} (equivalent).

For each class of entities, we randomly select 250 query pairs, so that each pair of queries contain same entity and different intent phrases. The chance of a query being selected is proportional to the frequency of this query. To balance the number of query pairs of each type of relationship, we select one third of query pairs being unrelated, one third being equivalent, and one third in which one query belongs to another. In this step the relationship between a pair of queries is predicted by the intent tree generated either by DMST or HAC (one of them is randomly selected to make each prediction). This will not lead to biased results because the predicted relationship is not indicated to the worker in any manner.

We have 2500 Mechanical Turk questions for the 10 classes of entities, and assign each question to three workers (paying 1 cent to each). We find the chance that a worker agrees with another one is 69.8%. We only consider the questions on which at least two workers agree, and there are 2327 of such questions. The distribution of answers to these questions is as follows: 930 "unrelated", 384 "belongs", 288 "contains", and 725 "equivalent".

In order to see whether the answers by Mechanical Turk are accurate, we randomly choose 100 query pairs with Mechanical Turk judgments (ten from each class), and manually identify the relationships between each pair of queries. Unless the relationship is obvious by common sense, we figure out the intents of each query according to the most clicked URL on Bing. The average accuracy of Mechanical Turk judgments is **0.83**. The precision and recall for query pairs with each type of relationships is shown in Table 3. Query pairs with "contains" relationship is converted into those of "belongs" by switching the order. In general Mechanical Turk judgments are quite accurate. Please note although there are only three categories of results in Table 3, for each question there are four possible answers, because "belongs" relationship can happen in both directions.

Table 3: Precision/Recall of Mechanical Turk

	Precision	*Recall*	F_1
Unrelated	1.000	0.727	0.842
Belongs	0.680	0.895	0.773
Equivalent	0.944	0.919	0.931

In general the judgments by Mechanical Turk are reasonably accurate. There are some errors, such as {qvc handbags} is considered to belong to {qvc outlet}, while "qvc" is mainly an online store and these two queries have very different intents.

6.3 Query and Intent Phrase Relationship

We first check whether the query relationship defined in Def.1 in Section 4.2 is consistent with the judgments by Mechanical Turk. For each query pair q_1, q_2 with judgments by Mechanical Turk, we get the degree of intents of one query being included in those of the other, i.e., $d(q_1 \subseteq q_2)$ and $d(q_2 \subseteq q_1)$. The distributions are shown in Figure 5. We can see the distribution of relationships between unrelated query pairs and equivalent ones are mostly separated, while query pairs with one belonging to the other are less separated from other types.

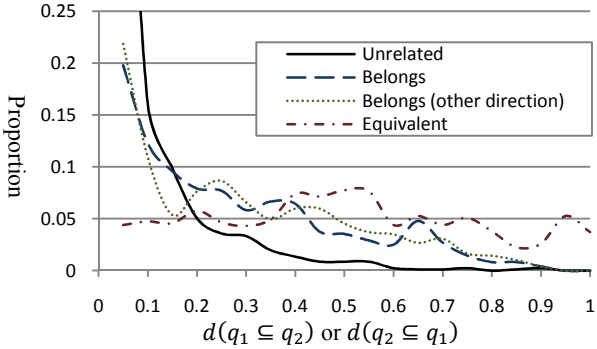


Figure 5: Distribution of relationships between queries for each type of query pairs