# Towards Rich Query Interpretation: Walking Back and Forth for Mining Query Templates

Ganesh Agarwal[†] Govind Kabra[*] Kevin Chen-Chuan Chang[†,*]
[†] Computer Science Department, University of Illinois at Urbana-Champaign
[*] Cazoodle Inc.
{gagarwa3, kcchang}@illinois.edu, {govind.kabra, kevin.chang}@cazoodle.com

## ABSTRACT

We propose to mine structured query templates from search logs, for enabling rich query interpretation that recognizes both query intents and associated attributes. We formalize the notion of template as a sequence of keywords and domain attributes, and our objective is to discover templates with high precision and recall for matching queries in a domain of interest. Our solution bootstraps from small seed input knowledge to discover relevant query templates, by harnessing the wealth of information available in search logs. We model this information in a tri-partite QueST network of **que**ries, **s**ites, and **t**emplates. We propose a probabilistic inferencing framework based on the dual metrics of precision and recall— and we show that the dual inferencing correspond respectively to the random walks in backward and forward directions. We deployed and tested our algorithm over a real-world search log of 15 million queries. The algorithm achieved accuracy of as high as 90% (on $F$-measure), with little seed knowledge and even with incomplete domain schema.

## Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Query formulation, Search process; H.2.8 [**Database Applications**]: Data mining

## General Terms

Algorithms, Experimentation

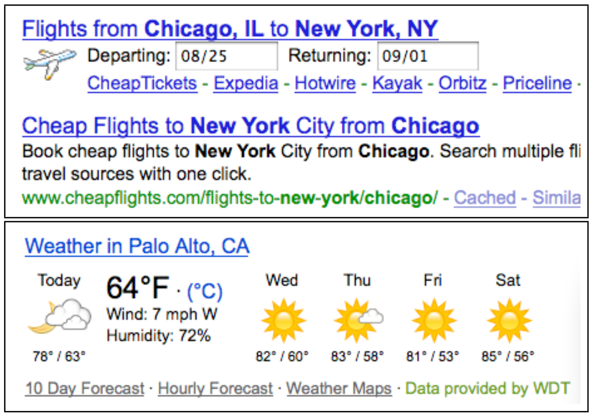
## Keywords

query templates, query intents, query attributes, search log mining

## 1. INTRODUCTION

The wide spread of the Internet and the popularity of search engines have rendered *keyword queries* as the dominating *lingua franca* of information access. Meanwhile, as the nature of the Web evolves, information search over the Web has become increasingly diverse and, thus, complicated. The proliferation of semi-structured or structured data, such as those from online databases, or the so called "deep Web," provides abundant information in various "domains" of interest. Consequently, today, users are looking for all kinds of information online—with simple keyword queries.

The prevalence of simple keyword queries coupled with the proliferation of diverse information online poses significant challenges for search systems in effective retrieval of data. As the key barrier, how to "interpret" a query for the wide variety of target domains it may aim at? This issue of *query interpretation* is central



Flights from **Chicago, IL** to **New York, NY**
Departing: 08/25 Returning: 09/01
CheapTickets - Expedia - Hotwire - Kayak - Orbitz - Priceline ·

Cheap Flights to **New York** City from **Chicago**
Book cheap flights to **New York** City from **Chicago**. Search multiple fli
travel sources with one click.
www.cheapflights.com/flights-to-**new-york**/**chicago**/ - Cached - Simila

Weather in Palo Alto, CA
Today 64°F · (°C) Wind: 7 mph W Humidity: 72% 78° / 63°
Wed 82° / 60° Thu 83° / 58° Fri 81° / 53° Sat 85° / 56°
10 Day Forecast · Hourly Forecast · Weather Maps · Data provided by WDT

**Figure 1: "chicago new york" (Google); "palo alto weather" (Bing).**

in every aspect of search: What contents to match? What vertical search services to invoke (*e.g.*, in "universal search" or in vertical search routing)? What advertisements to show? With the simplicity of keyword queries and the complexity of information they target, query interpretation has become more demanding in two ways:

*First*, as its traditional focus, query interpretation aims to recognize the *intent*, or the *domain* of interest—the diversity of the Web has mandated such recognition to be specific and fine grained. While much work has focused on classifying queries to predict their intents (*e.g.*, [1, 2]), the challenge is ever increasing. To effectively match queries to contents, vertical services, and ads, the recognition has evolved from high-level types (such as *navigation*, *transaction*, *information*), topics (*travel*, *health*) to finer-grained categories, such as *hotel*, *flight*, *job*, each of which represents certain information objects (hotels or flights to book, and jobs to apply).

*Second*, as a new issue, for each specific domain, query interpretation now also faces the challenge of recognizing intended *attributes*, or the various "aspects," that are inherently associated with a given intent. When users look for a particular domain of data objects (*e.g.*, *job*), they naturally specify the attributes (*e.g.*, *location* of jobs like "jobs in chicago"; or *type* of jobs like "accounting jobs").

This paper aims to enable *rich query interpretation* by recognizing intended domain as well as the associated attributes. Such rich interpretation is essential for search response tailored to specific intents, such as in ranking contents, invoking verticals, or matching ads. *E.g.*, Figure 1 shows $Q_1$: "chicago new york" from Google and $Q_2$: "palo alto weather" from Bing. Observe that, *first*, the responses tailor to the specific domains: While $Q_1$ does not mention "flight", Google lists flight verticals like Expedia. Similarly, Bing directs $Q_2$ to a weather vertical. Second, the responses recognize the attributes. For $Q_1$, Google recognizes "chicago" as attribute *from* and "new york" as *to*. For $Q_2$, Bing recognizes "palo alto" as the attribute *location* of weather forecast.

While rich query interpretation is essential, how can effectively recognize the intended domain as well as attributes? To date, this problem has not been systematically addressed, although current search engines have shown sporadic usage in certain scenarios (such as for *flight* and *weather* as in Figure 1).

As our *first* contribution, while the notion of query templates has been implicitly exploited (in search engines and in other mining tasks; Sec. 2), this paper is the first to survey its prevalence, formulate the new concept, and formalize its quality metrics.

We begin by surveying, as Sec. 3 will report, a large scale search log [3], with a sample of 28,000 queries, which indicated that a high frequency of queries follow structured patterns– *e.g.*, 90+% for *real estate* and *hotel*, and 80+% for *automobile* and *rental car*. Users naturally refer to different aspects of their search needs, *e.g.*, when looking for jobs, users may mention the "location" (*e.g.*, chicago, boston) or the "company" (*e.g.*, microsoft, boeing, *etc.*.) of their desired jobs. Thus, each user may talk about specific *instances* of different aspects, the overall *template* is similar. *E.g.*, in the job domain, we may see many queries sharing a similar template, *e.g.*, "jobs in boston", "jobs in seattle", "jobs in chicago" are all specific *instantiations* of a common query template—"jobs in #location", where #location represents the location aspect in the job domain.

The structured patterns as "templates" are useful in two ways:

1. It serves as a query intent *classifier*. Given a query, such as "accounting jobs in chicago", by matching to a "job" template, say $\langle$#category jobs in #location$\rangle$, we can interpret the intended domain of the query as for the job domain.
2. It is also a query *parser*, which recognizes the associated attributes of an intent. *E.g.*, matching of query with the template reveals that user is seeking job with #location="chicago" and #category="accounting." Such structured interpretation will help in better handling of queries.

As our *second* contribution, this paper proposes the problem of query template discovery, and develops a principled probabilistic inference framework as the solution. We define the metrics for distinguishing good templates—in terms of *precision* and *recall* with respect to the domain of queries interested. We then develop the QueST framework, a graphical model of inference upon the network of instantiation (between queries and templates) and click-through (between queries and sites), which will infer precision and recall in the "probabilistic" sense.

The dual frameworks, QuestP for precision and QuestR for recall-based ranking, exhibit an interesting duality: We show that, in the view of inferencing as *random walks*, they are simply walks of opposite directions—walking backward for precision, and forward for recall. The connection to random walk based inference models helps us understand the formal computational and convergence properties of our QueST algorithm. We stress that, while random walk has been widely applied, the formal relationship between forward and backward walks was unclear to-date, except for empirical performance comparison [4]. To our knowledge, our work is the first to establish the *duality* of precision and recall and its *connection* to the directions of random-walk propagation.

As our *third* contribution, we deployed and studied the performance of our solution over a large-scale search log of 15 million queries [3]. The experiments show that QueST significantly outperforms the baseline "classify then match". We tested the effectiveness of templates discovered for the application scenario of predicting query intents across seven domains of 28,000 manually labeled queries. Overall, the system is effective in finding structured query patterns that accurately predict query intents, achieving $70-90\%$ on $F$-measure, with as little as just four example sites as input, and the performance is robust to different types and sizes of inputs, as well as incomplete schema of attributes and instances.

## 2. RELATED WORK

**In terms of problem**, our work attempts to find patterns from query log, which is an instance of the general problem of knowledge discovery from query logs [5], an active area of research, with variety of applications such as implicit relevance feedback [6, 7], result caching [8], query suggestion [9, 10], query classification [1], and name-entity mining [11, 12, 13]. Ours is the first work that identifies the prevalence of query patterns, and formally addresses this new problem of query template mining.

The notion of query templates has been used only implicitly in other contexts: (1) in search engines, where templates are implicitly used for matching certain queries (*e.g.*, Fig. 1); (2) in named-entity mining, where templates are used as intermediate bridges, by using hand crafted patterns [11], or by learning a few top templates [12, 13], *e.g.*, $\langle$#movie-name trailer$\rangle$ for matching new movie names. In contrast, this paper formalizes the explicit concept of query templates, defines the metrics of precision and recall, and develops an inference framework for discovering templates of good quality.

**In terms of techniques**, we compare our solution with two-staged Classify&Match method as baseline, which first uses existing classification technique [1], and then matches templates against the classification results. Our experiments show this two-staged approach is ineffective in mining templates.

Our solution is based on interleaved inferencing upon a tripartite QueST graph of **Que**ries, **S**ites, and **T**emplates. While several related techniques use bipartite Q-S graphs [1, 14], our extension to include templates for integrated inferencing is necessary. Query logs are known to be sparse and noisy; the integrated inferencing helps in regulating propagation by accounting for all signals, as we discuss in Section 5.

We show that, interestingly, our probabilistic inference equations ultimately turn into *random walks* of the dual directions (see Sec. 6). Our first inference scheme, QuestP for estimating precision, corresponds to *backward* random walk, that minimizes harmonic energy function. As a duality, our second inference scheme, QuestR for estimating recall, corresponds to *forward* random walk, and thus is equivalent to the PageRank-family of inferencing [15].

While both the forward and backward propagation have been used before [16, 17, 18], to-date the duality relationship was unclear, except for the empirical performance comparison [4]. Our work is the first to establish the correspondence of random walks in the forward and backward directions to the recall and precision measures, respectively.

**In terms of related research areas**, since our solution needs domain schema as input, which may not be readily available, we need to incorporate automatic named-entity mining techniques for discovering new attribute classes and their instances. Several techniques exist, *e.g.*, learning from Web corpus ([19, 20, 21, 22]) and, more recently, from query logs (*e.g.*, [11, 12, 13]). Our study in Section 7.3 demonstrates that our framework can be seamlessly extended to incorporate named-entity and attribute discovery.

## 3. PROPOSAL: QUERY TEMPLATE

### 3.1 Motivating Survey: Structured Patterns

While structured patterns are useful for interpreting queries–do such patterns exist? That is, for similar search purpose, do users formulate queries of similar structures? As this question must be answered from actual query behaviors, we surveyed a search log of 15 million queries from Microsoft's MSN search engine [3].

In the survey, we checked if the sampled queries followed some patterns. As Section 3.2 defines formally, a pattern is an abstract structure that can instantiate *multiple* query instances. For a query $q1$, to see *whether* it follows some pattern, we identify *if* $q1$ mentions an instance of some attributes and *if* we could find another

| id | Query | Structured Pattern | Domain |
|---|---|---|---|
| $s_1$ | 280zx scissor doors | ⟨#model scissor doors⟩ | *automobile* |
| $s_2$ | dreams about tornadoes what does it mean | none | n/a |
| $s_3$ | fractionated stereotactic radiosurgery california | ⟨#surgery #location⟩ | *hospital* |
| $s_4$ | j william montgomery minnesota | ⟨#name #location⟩ | *people* |
| $s_5$ | locate mercedes inventory | ⟨locate #make inventory⟩ | *automobile* |
| $s_6$ | morgan milzow | ⟨#realtorcompany⟩ | *real estate* |
| $s_7$ | play nintendo online | ⟨play #videogame online⟩ | *video game* |
| $s_8$ | psychology of severus snape | ⟨psychology of #character⟩ | *literature* |
| $s_9$ | sport bike portland or | ⟨#model bike #location⟩ | *motorcycle* |
| $s_{10}$ | uhaul in colorado | ⟨#rentalcompany in #location⟩ | *car rental* |

**Figure 2: Structured patterns: *Any* queries.**

| Domain | Schema | Queries | Pattern% |
|---|---|---|---|
| *airfare* | #airline, #airport, #apcode, #location | 428 | 73.60% |
| *automobile* | #make, #model, #year, #location | 1656 | 85.87% |
| *hotel* | #hotel, #location | 1378 | 91.36% |
| *job* | #location, #category, #company | 921 | 74.38% |
| *real estate* | #location | 1471 | 91.71% |
| *car rental* | #company, #cartype, #location | 99 | 84.85% |
| *movie* | #title, #actor, #director | 723 | 37.62% |

**Figure 3: Structured patterns: 7 domains among 28K queries.**

query $q2$ that shares everything *but* the instances of the attributes. *E.g.*, consider $q$ = "280zx scissor doors" in Fig. 2. By manual checking, we recognized that "280zx" is a car model, which we denoted attribute #model. Looking in the query log, we found another query "300zx scissor doors", where "300zx" is another #model instance. Thus, these queries share pattern ⟨#model scissor doors⟩, and we name their similar interests as domain *automobile*.

**Survey 1:** *How likely does* any *random query follow some pattern with* any *attributes we could recognize?* We sampled 10 queries and, by manual checking, found that 9 of them—or a 90% "pattern ratio"— follow certain patterns (shared with at least one other query) that we could recognize. Figure 2 shows the 10 sample queries. We were able to find templates for all queries but $s2$: That is, for $s2$, we could not find another query in the log that shares a similar structure (say, "dreams about tigers what does it mean"). For each sample, we also identify the pattern and name the domain as we recognized. *E.g.*, $s4$ shares pattern ⟨#name #location⟩ (say, with another query "john tabor georgia") and $s10$ pattern ⟨#rentalcompany in #location⟩ (say, with query "hertz in chicago"). Overall, we saw that query patterns are prevalent.

**Survey 2:** *How likely do queries in a* particular *domain follow patterns with respect to some* specified *attributes?* In any domain of interest, we expect user queries would include "domain schema" (*e.g.*, #hotelname, #location). We surveyed seven domains, as shown in Fig. 3. As dataset, we used a sample of 28K queries, as an even mix of random sampling and domain biased sampling. For each query, we manually labeled if it is relevant to one of the domains. If so, we further checked if it followed some pattern. Overall, we again found that templates are prevalent. *E.g.*, as Fig. 3 summarizes, we found that 1378 of the 28K queries were in *hotel* domain, of which 1259 or 91.36% were "patterned." In 6 out of the 7 domains, we saw the pattern ratio was greater than 70%.

We note that the survey inevitably *underestimated* pattern ratios—due to our incomplete vocabularies, we were not able to recognize all attribute instances and thus will miss some query patterns. In particular, the low ratio in *movie* domain (37.62%) resulted from our incomplete movie #title and #actor names. We would miss, say, "bolt showtimes" for pattern ⟨#title showtimes⟩ if we do not include "bolt" as an instance of #title.

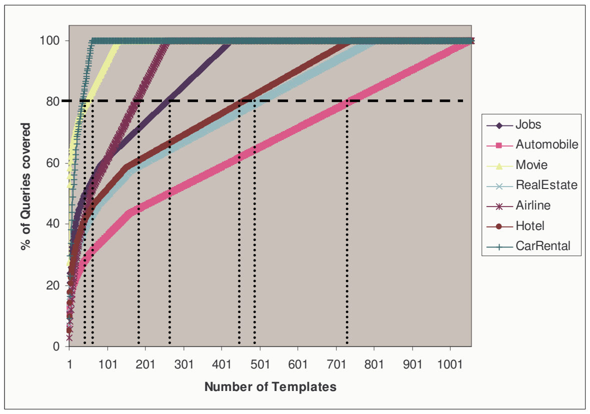


**Figure 4: Cumulative coverage of query templates.**

**Survey 3:** *How popular is each pattern? How many patterns are there?* Since we resort to structured patterns for rich query interpretation, as Section 1 motivated, we wonder how many patterns do we need to "cover" a domain of interest, say, *hotel*? We measured the *coverage* of a pattern $t$ as the ratio of queries in a domain that are instances of $t$. Figure 4 shows the cumulative distribution of the coverage ratios in each domain, with color-coded curves. We order the templates of each domain, on the $x$-axis, by their coverage of the domain queries. For example, for *automobile*, the top template is ⟨#make #model #year⟩, which covered 95 out of 1580 automobile queries (*e.g.*, "toyota corolla 2004"). For each rank position, we show on the $y$-axis the cumulative coverage over the domain queries. We observe that, for all domains, while the coverage increases rapidly in the beginning, it then slows down and overall requires a large number of patterns to cover most queries. *E.g.*, to reach 80% coverage, the *hotel* (brown-colored curve) and *real estate* (light blue) both need about 450 templates, and *job* (blue) about 250 templates. How to find these templates in a domain is thus our problem to solve.

## 3.2 Query Templates

We now formalize the notion of query templates. As our objective is to find structured query patterns in a specific domain, we begin by formally characterizing the *schema* of a domain.

**Definition 1 (Domain Schema):** The *domain schema* $\mathcal{D} = (A, I)$, for a given a domain of interest, consists of:

1. $A = \{a_1, \ldots, a_n\}$, where each $a_i$ is an *attribute*.
2. $I = \{I(a_1), \ldots, I(a_n)\}$, where $I(a_i)$ is the *vocabulary* of possible *instances* of $a_i$. ■

**Example 1 (Domain Schema):** For *job*, we may specify schema $\mathcal{D}_{job}$ = $(A, I)$: $A$ contains attributes for job listings, say, $A$ = {#location, #category, #company}. Each attribute has a vocabulary, which is conceptually a "dictionary" of instances, *e.g.*, $I$(#location) = {'new york', 'chicago', ... }, $I$(#category) = {'accounting', 'software', ... }, $I$(#company) = {'microsoft', 'boeing', ... }. ■

For simplicity, this paper assumes that complete domain schema is available. However, this is not a limitation of our framework. In practical deployment, our method can be coupled with widely studied named-entity mining techniques (*e.g.*, from Web corpus [19, 20, 21, 22], or from query logs [11, 12, 13]). Our experiments in Sec. 7.3 demonstrate that our framework can be extended to effectively incorporate existing named-entity mining techniques.

With respect to a schema, we now define templates for the domain. As motivated so far, a *query template* is an abstract query pattern with structured characteristics, which can instantiate multiple (concrete) query instances that preserve the characteristics. While our template structure is rather simple—a sequence of keywords and attributes—we note that our framework can generally handle any definitions of query templates, such as having regular expression features, as long as the computational requirement of "template generation" is met (see Section 4.1).

**Definition 2 (Query Template):** With respect to domain schema $\mathcal{D} = (A,\ I)$ and $W$ as the universe of all keywords, a *query template* $t$ is a sequence of terms $\langle v_1 \ldots v_n \rangle$, where each $v_i \in \Omega$, for the *vocabulary* $\Omega = W \cup A$, such that at least one of the terms is an attribute, *i.e.*, $\exists\ j \in [1, n]$ for which $v_j \in A$. ■

**Example 2 (Query Template):** Based on the schema $\mathcal{D}_{job}$, we can define templates like $t_j$ = ⟨jobs in #location⟩. Here, template $t_j$ specifies that the first two words must be "jobs" and "in" (in that order), followed by an instance of attribute #location. ■

A template can instantiate different queries; *i.e.*, it represents a class of queries, where the sequence and the keywords are preserved, while the place holders of attributes are replaced with their instance values. As inverse operation of *template instantiation*, we can also define the *template generation* process, *i.e.*, from a given query, we can generate the templates that instantiate that query.

**Definition 3 (Template Instantiation and Generation):** Given a query template $t = \langle v_1\ \ldots\ v_n \rangle$ and query $q$ = "$u_1\ \ldots\ u_m$", with respect to domain schema $\mathcal{D}$= ($A$, $I$), we say that $t$ *instantiates* $q$, or $q$ *generates* $t$, if $m = n$ and $\forall i$: $u_i \in I(v_i)$ when $v_i \in A$, or $u_i = v_i$ otherwise. The queries instantiated by a template $t$ is denoted by $I(t)$, and the templates generated by a query $q$ is denoted by $\mathcal{U}(q)$. Thus, we denote $t$ instantiates $q$ by $q \in I(t)$ or $t \in \mathcal{U}(q)$. ■

**Example 3 (Template Instantiation $I(t)$):** Template $t_j$ = ⟨jobs in #location⟩ can instantiate queries like $q1$: "jobs in chicago" and $q2$: "jobs in new york" or, generally, $I(t_j)$ comprises of all queries of the form "jobs in $p$", where $p$ is one or multiple words in $I$(#location).■

**Example 4 (Template Generation $\mathcal{U}(q)$):** Given a query, how to generate matching templates, *w.r.t.* our template definition (Def. 2)? Consider $q$: "accounting jobs in new york". Matching it to $\mathcal{D}_{job}$ (Example 1), we find 'accounting' $\in I$(#category) and 'new york' $\in I$(#location). We then enumerate templates by these attributes: $x1$:⟨#category jobs in chicago⟩, $x2$:⟨accounting jobs in #location⟩, and $x3$:⟨#category jobs in #location⟩. Thus, $\mathcal{U}(q) = \{x1, x2, x3\}$.■

We note that the instantiation capability of a template $t$ makes it useful for "interpreting" those matching queries in $I(t)$, by telling us the intent as well as the structure of the queries. *E.g.*, given $t_j$ for *job*, since $t_j$ instantiates $q1$, we know that $q1$ has an intent of the *job* domain (*i.e.*, finding job listings) and that "chicago" refers to the location of desired jobs.

## 4. PROBLEM: TEMPLATE DISCOVERY

### 4.1 Problem: Mining Templates

**Conceptual Template Universe:** By the definition of query template, we can conceptually characterize the universe of all *possible* templates as comprising of arbitrary combination of any keywords and any attributes in any length. However, not all these arbitrary combinations are relevant; most of these templates will not instantiate any queries in the domain of interest.

For discovering templates relevant to a given domain $D$, instead of any arbitrary patterns, we will thus generate candidate templates using only queries that can be asked in $D$. Suppose we know such "domain queries"; let $L(D)$ denote all the possible queries in $D$ from a query log $L$. For now, let's assume such domain queries $L(D)$ are clearly identified from $L$; later, our solution will need to infer the relevance of queries, bootstrapping from some initial seed knowledge. Given $L(D)$, the "universe" $T$ of template candidates is those templates that ever instantiate a query $q$ in $L(D)$ or, in other words, can be generated by at least some such $q$. Thus, the candidate template universe is $T = \{t \mid \exists q \in L(D), \textit{s.t.}\ t \in \mathcal{U}(q)\}$.

We stress that this template universe $T$ is merely conceptual. Our framework does not assume $T$ to be fully materialized. Instead, we should generate relevant templates from relevant queries—on the fly as soon as we identify such relevant queries.

**Computational Requirement of Template Generation:** Thus, to enumerate candidate templates on the fly, what we require is the capability to generate matching templates from some query $q$, or the function $\mathcal{U}(q)$. For our specific definition of query template (Def. 2), we can use $\mathcal{U}(q)$ as in Example 4. In general, our framework can support any definition of query templates, as long as there exists such a function $\mathcal{U}(q)$ to generate templates from queries.

**Quality Metrics:** As the next question, we wonder how to determine the "relevance" of a template $t$? Naturally, we measure the quality of $t$ by how well it can interpret $L(D)$.

On the one hand, $t$ should broadly capture as much of $L(D)$ as possible. We thus ask: What fraction of $L(q)$ queries are instantiations of $t$? Some templates are more "popular" than others. Our Survey 3 measured in Figure 4 such coverage, where we saw that a few templates (*e.g.*, ⟨#make #model #year⟩ for *automobile*) are more popular than others. Given $L(D)$ as the "target" set and $I(t)$ as the "matched" set, we thus propose to measure such coverage with the standard retrieval metric of *recall*:

$$R(t) = |L(D) \cap I(t)|/|L(D)| \quad (1)$$

Strictly speaking, in Eq. 1, $L(D)$ and $I(t)$ should be bags, instead of sets, with multiple occurrences of queries. Our probabilistic framework does account for frequencies of queries.

On the other hand, $t$ should precisely capture only $L(D)$ to make accurate prediction of intents. We thus ask: What fraction of $I(t)$ actually falls in $L(D)$? With the inherent ambiguity of keyword queries, a template may make wrong predictions. *E.g.*, the simple template ⟨#company⟩—say, for query "microsoft"—may mean to find *job* or something else, say, *product*, of the company. In contrast, template ⟨jobs at #company⟩ (*e.g.*, "jobs at microsoft") seems more reliable, although not perfect: Query "jobs at apple" may intend for "Steve Jobs" and not an employment. In parallel to recall, we thus propose *precision* of templates:

$$P(t) = |L(D) \cap I(t)|/|I(t)| \quad (2)$$

Ultimately, the overall quality of a query template $t$ is to be measured by some combination of precision and recall. The two metrics are "competing." A broad template, such as ⟨#company⟩, may have good recall but poor precision. A strict template like ⟨jobs at #company⟩ may have good precision but poor recall. To support different applications with varying requirements, it is imperative that we determine both $P(t)$ and $R(t)$, in order to support scoring functions *score*$(t)$ as any combinations, such as $F$-measure $F(t)$.

**Problem Definition** We can now state our problem quite simply as finding good templates by precision, recall, or $F$-measure, from a search log $L$, with respect to domain schema $\mathcal{D}$ and labeled example queries $Q_0$. The *input* consists of the following: 1) For our purpose, a *query log* gives a set of *queries* $Q$, a set of URLs or *sites* $S$, and how queries clicked-through to sites. We denote the *click-through queries* of site $s \in S$ by $\mathcal{C}(s) = \{q | q \in Q$, and $q$ clicks to $s\}$. 2) The domain schema $\mathcal{D}$ specifies attributes $A$ and instance vocabularies $I$. 3) We also need "seed knowledge" about domain relevance, *e.g.*, example queries $Q_0$ that are in the domain. The *output* returns a ranked list of templates, sorted by their usefulness. For scoring, we must be *able* to adopt different measures: precision, recall, or $F$, *depending* on application requirements.

**Problem:** ***Query Template Mining from Search Log***

**Input:** • Search log: $L$ = (queries $Q$, sites $S$, click-through $\mathcal{C}$).
• Schema $\mathcal{D}$ = ($A$, $I$) for domain $D$.
• Seed queries $Q_0$ labeled as in domain $D$.

**Output:** Ranked list of templates $t$, sorted by $score(t)$,
for $score(t) \equiv P(t)$, $R(t)$, or any combination such as $F(t)$.

**Need for Sites and Click-through:** Our problem definition above requires sites and click-through as input, so that we can exploit the search log $L$ fully to identify domain queries $L(D)$. Our solution leverages the "network" between not only queries and matching templates but also queries and matching sites to form an integrated framework, for inferring the relevance of queries, through the "regulation" of templates and sites.

**Flexibility in Seed Knowledge:** In our problem definition, we specified example queries as seed knowledge input. As we will see, our framework can be deployed with flexible seed input, *e.g.*, seed queries, seed sites, or even seed templates— or their hybrid combinations. In experiments, we show that our algorithm produces good results under all combinations of seed input.

## 4.2 Baseline Approach

To motivate our solutions, we start with a natural baseline algorithm. Since our goal is to find templates that match domain queries, if we were able to separate $L(D)$ from all queries $Q$ in the log, we can simply count in $L(D)$, for each template, how it performs in terms of precision and recall.

A natural baseline is thus Classify&Match (Fig. 5), with *two stages*. The first stage classifies log $Q$ to find domain queries $L(D)$, applying some threshold $\theta$ on the results. The second stage matches each $t$ in the template universe $T$ to this "estimated" $L(D)$, to *estimate* $P(t)$, $R(t)$, or $F(t)$ as $score(t)$.

The two-staged baseline, while intuitive, suffers two major issues inherent from the separation of the two stages. Our experiments show the poor performance of this approach, using a state of the art classifier [1] for query log as the first stage. These deficiencies motivated our development of a robust, principled probabilistic inference framework.

*First*, it lacks **probabilistic modeling**. In order to straightforwardly connect the two stages, we have to decide a threshold for the classifier to divide $Q$ as $L(D)$ and $\neg L(D)$. As we explained, queries are ambiguous, and such hard division is crude—in our experiments, we tried several choices of threshold (in the range of [0.80, 0.95] and none worked well. What we need is more robust probabilistic modeling that captures the fuzzy nature of semantic relevance and, in turn, that of recall and precision.

*Second*, it lacks **integrated inference**. Search logs, while valuable for learning query interpretation, are also known to be noisy and sparse. They are *noisy*: A click-through does not always mean semantic relevance—it might be a trial or a mistake. They are also *sparse*: Users only click on a few sites for each query. By separating template induction (stage 2) from query classification (stage 1), the baseline is critically missing semantic connections through templates during query classification, which will lead to suboptimal results. With templates integrated in the loop of inference, the "global inference" will *regulate* noises, since irrelevant queries are less likely to share templates. They will also *enrich* sparsity, by connecting queries through sharing the same templates.

```
• Input: $(Q, S, \mathcal{C})$, $\mathcal{D} = (A, I)$, $Q_0$, threshold $\theta$.
• Output: Templates $t$, ranked by $score(t)$.
1: classify $Q$ to $L(D)$ from $Q_0$, thresholded by $\theta$.
2: $T = \{t \mid t \in \mathcal{U}(q), \forall q \in L(D)\}$ //candidates.
3: for (each template $t$ in $T$) do
4:     $X = L(D) \cap I(T)$ //queries in $D \wedge$ match $t$.
5:     compute $score(t)$ //by $P(t)$, $R(t)$, or $F(t)$.
6: end for
7: return $T$ sorted by $score(t)$
```

**Figure 5: Baseline algorithm: Classify&Match.**

## 5. MODELING: QueST FRAMEWORK

We now develop our overall QueST framework for discovering templates with recall and precision as the quality metrics. As just motivated, we will first model the quality measures in the probabilistic sense, upon which we will then cast the discovery as a semi-supervised learning problem.

## 5.1 Probabilistic Modeling

As the foundation, we develop the probabilistic sense of recall and precision, for all constructs—templates, queries, and sites.

*First*, we generalize $P$ and $R$ for templates to the probabilistic sense. Eq. 1 and 2 define them in the standard set-semantics intersections, where $L(D)$ and $I(t)$ are both "crisply" defined. However, a random query $q$, with the inherent ambiguity of keywords, may be only *likely* to be in domain $D$; *e.g.*, as explained earlier, query "microsoft" might be in *job* or *product*.

Thus, we need the probabilistic notion of *semantic relevance*—a probability of how things are intended to *match*. To simplify notations, we will "generically" denote $\text{match}(A, B)$ for the event that $A$ and $B$ are semantically matching (for some $A$ and $B$ pertinent in discussion). In particular, how is a query $q$ relevant to domain $D$? *E.g.*, is "jobs in chicago" matching domain *job*? This statement is, in terms of the set notation above: How likely is $q \in L(D)$? We write $\text{match}(q, D)$ for the event that $q$ is relevant to $D$, and $p(\text{match}(q, D))$ is their probability of relevance. Similarly, we write $\text{match}(q, t)$ for that $q$ is semantically matching $t$, among the multiple templates (as discussed earlier) that $q$ can be instantiated from. We will denote $\text{match}(q, s)$ for query $q$ relevant to a site $s$ (among the multiple sites that $q$ clicks to). Together, we note that such probabilistic "matching" is necessary—not only that keyword queries are ambiguous, but click-throughs are also noisy in nature.

Now, we can view Eq. 1 is a statistical way: Since the numerator is the count of $(L(D) \cap I(t))$, it is proportional to the probability that, when we draw a random query $q$, $p(q \in L(D) \wedge q \in I(t))$ or, in our "match" notation, $p(\text{match}(q, D), \text{match}(q, t))$. Similarly, the denominator becomes $p(\text{match}(q, D))$. Substituting them into Eq. 1, we get the probabilistic recall as the conditional probability below. Similarly, we can rewrite Eq. 2 statistically.

$$R(t) = p(\text{match}(q, t)|\text{match}(q, D)) \tag{3}$$

$$P(t) = p(\text{match}(q, D)|\text{match}(q, t)) \tag{4}$$

*Second*, we extend precision and recall to each query $q$ and site $s$. By this extension, we will be able to "interrelate" $P$ and $R$ between these related events, and thus achieving inference across them.

We can model the precision and recall of site $s$ analogously to template $t$. For precision $P(s)$, given the queries matching $s$, we measure how likely they match $D$. For recall $R(s)$, given the queries matching $D$, we measure how likely they match $s$.

$$R(s) = p(\text{match}(q, s)|\text{match}(q, D)) \tag{5}$$

$$P(s) = p(\text{match}(q, D)|\text{match}(q, s)) \tag{6}$$

Finally, for query $q$, we can model its precision and recall, by capturing the semantic relevance to domain $D$. The recall of $q$ is the "fraction" of domain queries that are actually $q$, *i.e.*, the probability that $x = q$ among queries $x$ matching $D$. The precision, on the other hand, is simply the probability that $q$ matches $D$.

$$R(q) = p(x = q|\text{match}(x, D)) \tag{7}$$

$$P(q) = p(\text{match}(q, D)) \tag{8}$$

## 5.2 Inference Framework

Our framework will resort to integrated inference (unlike the partitioned phases in baseline Classify&Match) between all related constructs–queries $Q$, sites $S$, and templates $T$, by building a *tripartite* graph, QST-graph. Figure 6 shows the graph for a toy example. In the QST-graph $G = (V, E)$, the nodes are $V = Q \cup S \cup T$, and the weighted edges $E$ are as follows:

- $\forall q \in Q, \forall t \in T$: if $q \in I(t)$, or $q$ instantiates $t$, there is an edge in between, with weight $I_{qt} = 1$.
- $\forall q \in Q, \forall s \in S$: if $q \in \mathcal{C}(s)$, or $q$ clicks to $s$, there is an edge in between, with weight $C_{qs}$ as the click-through frequency.

As a semi-supervised learning problem, initially, we are given a few seed queries $Q_0$, where each query $x$ is labeled with precision $P_0(x)$. Note that, while users can label seed precision as how likely $x$ indicates the domain of interest, it is infeasible for users to provide "seed recall," which depends on *all* relevant queries. As an estimate, we initialize the seed recall as precision normalized among all the given queries, taking into account the frequency of queries $x \in Q_0$ in $Q$. Our initial condition is thus the given precisions (which are ground truth) and estimated recalls (which will be re-estimated through the inference process), as follows.

$$\forall x \in Q_0\colon P(x) = P_0(x); \quad R(x) = R_0(x) \equiv \frac{P_0(x)}{\sum_{x \in Q_0} P_0(x)} \tag{9}$$

Our goal is to estimate $R(t)$ and $P(t)$, $\forall\, t \in T$; as necessary intermediaries, the inference will also estimate $R(q)$ and $P(q)$, $\forall\, q \in Q$ as well as $R(s)$ and $P(s)$, $\forall\, s \in S$.

Overall, we develop Algorithm QueST, for template discovery. As Fig. 7 shows, it starts by generating template candidates and constructing the QST-graph. We note that the construction is only conceptual; we can "materialize" the graph during inference only as needed. On this graph, starting from $Q_0$ with given $P_0$ (and estimated $R_0$), QueST will infer $R$ and $P$ for each node. It infers recall by QuestR and precision by QuestP, as we will develop next in Sec. 6. Finally, QueST will rank templates by $P$, $R$, or the combined $F$-measure (depending on application requirements).

## 6. INFERENCE: QuestR AND QuestP

To complete the QueST framework, we develop inferencing equations that propagate the precision and recall across the QST-graph. Specifically, we will derive probability estimation of a node in terms of its neighbors, through the semantic relevance across their edges, as summarized in Fig. 8. The dual inferencing equations for recall and precision metrics can be interpreted as random walks in opposite directions on QST-graph(as illustrated in Fig. 9). To our knowledge, this paper is the first to identify this interesting duality. We will formally connect to existing learning frameworks to understand the computational and convergence properties of QuestP and QuestR algorithms.

## 6.1 Recall: Quest Forward

We now establish the inference of probabilistic recall, deriving equations $R1$, $R2$, and $R3$ as Fig. 8 summarizes. To begin with, as input, the system is given initial recall estimates $R_0(x)$ for a small

**Queries Q**
$q_1$: jobs in chicago
$q_2$: jobs in boston
$q_3$: jobs in microsoft
$q_4$: jobs in motorola
$q_5$: marketing jobs in motorola
$q_6$: 401k plans
$q_7$: illinois employment statistics

**Sites S**
$s_1$: monster.com
$s_2$: motorola.com
$s_3$: us401k.com

**Templates T**
$t_1$: jobs in #location
$t_2$: jobs in #company
$t_3$: #category jobs in #company
$t_4$: #location employment statistics

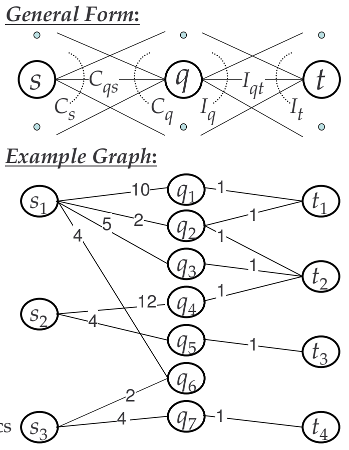


**Figure 6: Example: a toy search log and the graph.**

```
• Input: $(Q, S, \mathcal{C})$, $\mathcal{D} = (A, I)$, $Q_0$ with $P_0$ (and $R_0$; Eq. 9)
• Output: Templates $t$, ranked by $score(t)$.
1: $T = \{t \mid t \in \mathcal{U}(q), \forall q \in Q\}$ //or, materialize t on demand.
2: construct QST-Graph $G$ by $Q$, $S$, $T$, $\mathcal{C}$, $I$.
3: inference recall by QuestR on $G$ with $R_0$
4:     update $R$ till convergence by $R1$, $R2$, $R3$
5: inference precision by QuestP on $G$ with $P_0$
6:     update $P$ till convergence by $P1$, $P2$, $P3$
7: return $T$ sorted by $score(t) = P(t)$, $R(t)$, or $F(t)$
```

**Figure 7: Algorithm QueST: Mining query templates.**

number of seed queries $x \in Q_0$ (Eq. 9). Based on this seed knowledge, we now need to determine the recall measure of every node (including updating the seed nodes). We thus must "interconnect" the nodes by their dependencies– *i.e.*, how to estimate the probability $P(x_i)$, for each $x_i \in V$, given the probability values for all its neighboring nodes. As the graph is tripartite of the form $S$–$Q$–$T$ (Fig. 6), we must specify inference of $Q \rightarrow T$, $S \leftarrow Q$, as well as $Q \leftarrow T$ and $S \rightarrow Q$.

We derive the first relationship $R1$ of Figure 8 in Equation 10 below, for inferring recall of a template $R(t)$.

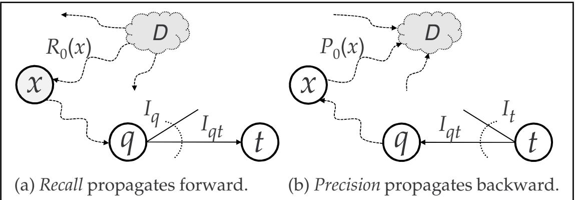
$$\begin{aligned}
R(t) &\equiv^1 p(\text{match}(q,t)|\text{match}(q,D)) \\
&=^2 \textstyle\sum_{q_i \in Q} p(\text{match}(q,t), q = q_i|\text{match}(q,D)) \\
&=^3 \textstyle\sum_{q_i \in I(t)} p(\text{match}(q,t), q = q_i|\text{match}(q,D)) \\
&=^4 \textstyle\sum_{q_i \in I(t)} p(\text{match}(q,t)|q = q_i, \text{match}(q,D)) \\
&\qquad\qquad \cdot p(q = q_i|\text{match}(q,D)) \\
&=^5 \textstyle\sum_{q_i \in I(t)} p(\text{match}(q,t)|q = q_i) \cdot p(q = q_i|\text{match}(q,D)) \\
&=^6 \textstyle\sum_{q_i \in I(t)} I_{q_i t}/I_{q_i} \cdot R(q_i)
\end{aligned} \tag{10}$$

The step 1 starts with the definition of $R(t)$ (Eq. 3). In step 2, to bring in queries, we expand it to the joint distributions with every $q_i \in Q$. Step 3 restricts $q_i$ to only those instantiated by $t$, or $q_i \in I(t)$, which are the neighbors of $t$, so that $\text{match}(q_i, t)$ can have a non-zero probability. By the Bayes' theorem, step 4 rewrites using $p(AB|C) = p(A|BC)\ p(B|C)$. In step 5, since $\text{match}(q,t)$ depends only on what $q$ is– *i.e.*, it is conditionally independent of $\text{match}(q, D)$, *given* $q = q_i$– we can remove $\text{match}(q, D)$. As the first term equals $I_{q_i t}/I_t$ (where $I_t$ is a shorthand given in Fig. 8) and the second term $R(q_i)$ (Eq. 7), step 6 completes the rewriting.

We can similarly derive the other two inference equations, which we omit due to space limit. As Fig. 8 shows, Equation $R2$ is in a form parallel to $R1$, since $S \leftarrow Q$ is symmetric to $Q \rightarrow T$ at the two

| 1) QuestR: *Quest Forward for Recall Inference* | 2) QuestP: *Quest Backward for Precision Inference* |
|---|---|
| ***R1***: $R(t) = \sum_{q \in I(t)} I_{qt}/I_q \cdot R(q)$, where $I_q = \sum_{\forall t: q \in I(t)} I_{qt}$ | ***P1***: $P(t) = \sum_{q \in I(t)} P(q) \cdot I_{qt}/I_t$, where $I_t = \sum_{\forall q: q \in I(t)} I_{qt}$ |
| ***R2***: $R(s) = \sum_{q \in C(s)} C_{qs}/C_q \cdot R(q)$, where $C_q = \sum_{\forall s: q \in C(s)} C_{qs}$ | ***P2***: $P(s) = \sum_{q \in C(s)} P(q) \cdot C_{qs}/C_s$, where $C_s = \sum_{\forall q: q \in C(s)} C_{qs}$ |
| ***R3***: $R(q) = \begin{cases} \beta_1 R_0(q) + \beta_2 \cdot \sum_{\forall t: q \in I(t)} I_{qt}/I_t \cdot R(t) \\ +(1-\beta_1-\beta_2) \cdot \sum_{\forall s: q \in C(s)} C_{qs}/C_s \cdot R(s) \end{cases}$ | ***P3***: $P(q) = \begin{cases} P_0(q) & \text{if } q \in Q_0; \\ \alpha \cdot \sum_{\forall t: q \in I(t)} P(t) \cdot I_{qt}/I_q & \\ +(1-\alpha) \cdot \sum_{\forall s: q \in C(s)} P(s) \cdot C_{qs}/C_q & \text{otherwise.} \end{cases}$ |

**Figure 8: The QueST dual inferencing framework: QuestP and QuestR.**



(a) *Recall* propagates forward. (b) *Precision* propagates backward.

**Figure 9: Inferencing recall and precision.**

sides of the tripartite graph. At the center, $R(q)$ for query $q \in Q$ depends on the initial estimate $R_0(q)$ and the recalls from the two sides, $R(t)$ and $R(s)$, thus the mixture of the dependence sources combined by parameters $\beta_1$ and $\beta_2$.

**Interpretation.** We observe that the result is simple yet interesting. Consider $R1$: The recall of $t$, $R(t)$, is the sum of the neighboring recalls, $R(q)$, each of which is distributed proportionally among its outgoing edges towards $t$—or, $t$ *receives* recall proportionally from its neighboring query nodes.

While several interpretations are possible, it is intuitive to view the inferencing as random walk. As Fig. 9 illustrates, we think of $R(t)$ as the probability of arriving at $t$, in a random walk starting from some hidden origin—the hidden domain $D$. This random walk interpretation of $R(t)$, in hindsight, is consistent with what probabilistic recall, in Eq. 3, shall capture: the proportion of queries from $D$ that will reach $t$. Equation $R1$ thus means—the probability of arriving at $t$ sums up the probabilities of arriving at its neighbors and then one last hop to $t$.

In sum, as we generalized recall from deterministic sense (Eq. 1) to a probabilistic measure (Eq. 3), upon the semantic relevance graph, we observe that the inference mechanism (as $R1$, $R2$, and $R3$ captures) lends itself naturally to the random walk interpretation in *forward* direction—from *given* seeds $x$ (*i.e.*, originating from the hidden domain) to *unknown* nodes $t$.

**Connections to Related Work and Convergence Properties:** The recall inference, as a forward walk model, connects to the PageRank [23] family of models. In particular, with "seed nodes" assignments, QuestR connects directly to topic-sensitive or personalized PageRank ([24, 15]). (We can rewrite Eq. $R1$, $R2$, and $R3$ to matrix forms that directly parallel the personalized PageRank formulation.) The connection thus allows us to understand that (subject to making the graph irreducible) iterative matrix computation would converge to the solutions.

## 6.2 Precision: Quest Backward

We next establish the inference of probabilistic precision—in which, we witness the interesting duality of recall and precision on the graph—The derivation of precision inference follows an exactly symmetric process to recall inference just discussed. Fig. 8 also summarizes the precision Equations $P1$ (for $Q \rightarrow T$), $P2$ (for $S \leftarrow Q$), and $P3$ ($Q \leftarrow T$ and $S \rightarrow Q$). We omit the symmetric process due to space limit. As the only difference, here, for seed queries $q$, as their $P_0(q)$ are specified in labeling (Eq. 9; unlike $R_0(q)$ which is only estimated), $P(q)$ will take this "ground truth" for $q \in Q_0$ and will not change throughout inferencing.

**Interpretation.** With the symmetry between precision and recall in the derivations of their inference, we also observe interesting duality in their interpretations. Consider $P1$: The precision of $t$, $P(t)$, is the sum of the neighboring precisions, $P(q)$, each of which is weighted by how $t$ can reach $q$. As Fig. 9 illustrates, we think of $P(t)$ as the probability of reaching $D$, the domain as a hidden destination, in a random walk starting from $t$. The hidden domain is specified through $P_0(x)$, *i.e.*, how seed queries $x$ can reach $D$.

This random walk interpretation of $P(t)$ is, again, consistent with what probabilistic precision, in Eq. 4, shall capture: the proportion of queries from $t$ that will reach $D$. Equation $P1$ thus means—the probability of $t$ reaching $D$ sums up the probabilities of one hop to its neighbors and then going from there.

While we identified recall inference as forward random walk, we now recognize that the inference of probabilistic precision turns out to be the opposite—backward random walk, from *unknown* nodes $t$ to *given* seeds $x$ (indicating the hidden domain as the destination).

**Connections to Related Work and Convergence Properties:** The backward random walk interpretation connects QuestP to a relatively recent semi-supervised learning framework, *harmonic energy minimization* [25], over a graph with labeled and unlabeled nodes. As the result of our formulation, we obtained inference equations $P1$, $P2$, and $P3$, which are *harmonic*—*i.e.*, the value $P(u)$ of a node $u$ is the (weighted) average of its neighbors. The harmonic update functions will lead to the assignment of $P$ values over graph $G$ in a way that will minimize the quadratic energy function in Eq. 11. That is, the closer nodes $u$ and $v$ are, *i.e.*, the larger their edge weight $w_{uv}$ is, the closer their $P$ values shall be. Thus, on the QST-graph, nodes that are close with edges (of instantiation or click-through) will have similar precision.

$$E(P) = \frac{1}{2} \sum_{u,v \in G} w_{uv}(P(u) - P(v))^2 \qquad (11)$$

As essentially an instance of the harmonic energy minimization framework [25], the solution of QuestP is unique, and can be evaluated by iterative matrix computation till convergence.

## 7. EXPERIMENTS

We report our evaluation of QueST over a large scale real query log across a range of query domains. Overall, the experiments demonstrate that QueST is accurate for finding templates for predicting query intents, outperforming the baseline of Classify&Match significantly, and is robust over a wide range of seed input types and sizes. Further, we extend QueST to handle incomplete schema, which shows the robustness for discovering not only new instances of attributes, but also new attributes.

## 7.1 Experiment Setting

**Query Log:** We used a real world query log from the MSN search engine [3] with 15 million queries and 12.25 million click-through (query, site) pairs. We preprocessed this query log: (i) As we were only interested in the clicked site, we truncated the clicked URL to its top-level domain name, *e.g.*, *monster.com*, rather than the complete URL. (ii) We remove *navigational queries* [26] (*e.g.*, "ebay"),

| **Type** | **Size:** *small* | **Size:** *medium* | **Size:** *large* |
|---|---|---|---|
| *site* | 2 sites | 4 sites | 8 sites |
| *query* | 5 queries | 20 queries | 50 queries |
| *template* | 2 templates | 5 templates | 10 templates |

**Figure 10: Seed input configurations.**

| Domain | QuestR | QuestP |
|---|---|---|
| Job | #location jobs<br>jobs in #location | #companyname #location job fairs<br>#location #industry positions |
| Airfare | cheap flights to #location<br>#airline airlines | #airline #location reservations<br>#location #airport airport |
| Automobile | #year #make #model<br>#make #model | #location #make used cars<br>#year #make #location for sale |
| Car Rental | #carrentalcompany rental car<br>#carrentalcompany rental | #carrentalcompany rental car #location<br>#cartype rent a car |
| Hotel | #hotelname #location<br>#location hotels | #hotelname #location hotels<br>#hotelname at #location |
| Movie | #movietitle<br>#actorname | #actorname new movie<br>#movietitle characters |
| Real Estate | #location real estate<br>#location homes for sale | #location area homes for sale<br>#location commercial real estate |

**Figure 11: Top 2 templates derived by QuestR and QuestP.**

whose target is a particular site (*ebay.com*). The preprocessing resulted in 2.8 million unique queries. We used 80% of these queries as the "query log" for mining templates from, and used the rest 20% for testing the discovered templates.

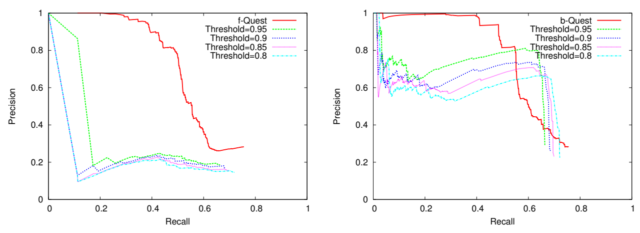
**Target Domains:** We studied performance in seven domains—*airfare*, *automobile*, *hotel*, *job*, *real estate*, *car rental*, *movie*—as our survey (Sec. 3) also studied. We report the overall performance for all the domains (Fig. 15). Due to space limitation, for fine grained studies, we report results only for *job* domain, as the observations are similar in all the domains.

**Application Scenarios:** We evaluated the usefulness of query templates, as QueST aims to discover, directly in their applications—for predicting query intents (by matching user queries). As the ground truth to test against, we sampled 28,000 unseen queries (see next)—the same set of queries that we used in our survey in Sec. 3. As Fig. 3 summarizes, among the 28K, 921 queries are for *job*, of which 74.38% (*i.e.*, 685) have patterns. We evaluate the precision and recall of the templates discovered for *job* domain using these 685 queries (which have patterns) as ground truth.

**Test Set:** Our test set included 28K unique queries—20K sampled *randomly* and 7K sampled with *domain-focus* (from queries clicking on manually enumerated sites relevant for our 7 domains). Our labelers (undergraduate students in Psychology) inspected each query, and determined the domain of the query, and if it is an instantiation of a structured pattern. To facilitate in the task of labeling, we created a labeling interface which showed results for that query from google.com. The search results helped our labelers determine the intents. The peak labeling speed was 600 queries per hour.

**Evaluation Metrics:** Our algorithms (Figure 7) as well as the baseline method (Figure 5) produce a ranked list of templates. For each template, we compute its precision, recall, and F-measure by counting the number of matches to positively labeled *vs.* negatively labeled queries in the test set of 28K queries (the application scenarios for intent prediction as just discussed).

**Seed Input Configuration:** Our QueST algorithm can take any types of seeds (site, query, template) as input, and requires only a modest size of input. We studied all the three types of input, *w.r.t.* three different sizes, as Fig. 10 summarizes. For each configuration, we manually compiled the specified number of input seeds (*e.g.*, 5 queries for the *query-small* configuration).



(a) Scoring by recall (b) Scoring by precision

**Figure 12: Comparison of QueST with Classify&Match.**

**Parameter Setting:** In all our experiments for QuestP, we set the value of $\alpha = 0.5$, thus, giving equal importance to both templates and sites in inferring the precision of queries (see P3 in Figure 8). For QuestR, we set the value of $\beta_1 = 0.1$, thus giving 10% importance to initial condition, and leaving 90% importance to the influence of templates and sites (see R3 in Figure 8). We give equal importance to sites and templates by setting value of $\beta_2 = 0.45$. It would be interesting future study to vary these parameters and evaluate relative importance of different factors.

## 7.2 Performance Results

**Illustrative Results:** As shown in Fig. 11, the top-2 templates produced by QuestR and QuestP algorithms illustrate that the two methods very well match the desired objectives of ranking templates based on recall and precision, respectively. These illustrative results are for input configuration of medium size input sites. *E.g.*, for *job*, top-2 templates produced by QuestR algorithm are ⟨#location jobs⟩ and ⟨jobs in #location⟩, indicating QuestR method ranks "popular" templates first. Likewise, for QuestP we see that the top-2 templates, *i.e.*, ⟨#companyname #location job fairs⟩ and ⟨#location #industry positions⟩, are "surely" for *job* domain.

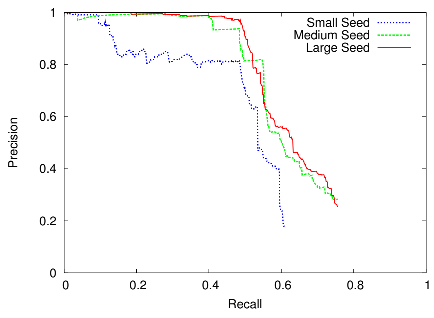
**Baseline: Classify&Match** As our first experiment, we compare the performance of our algorithms to the two-staged baseline method of Classify&Match method, described in Section 4.2. We implemented Algorithm 1 of [1] for query classification, and ran it for 20 iterations. We report results for *job* domain, with sites as seed input with medium size configuration.

As Fig. 12 (a) shows, QuestR (red color) consistently outperforms baseline method with recall based scoring (other colors), for all values of threshold.

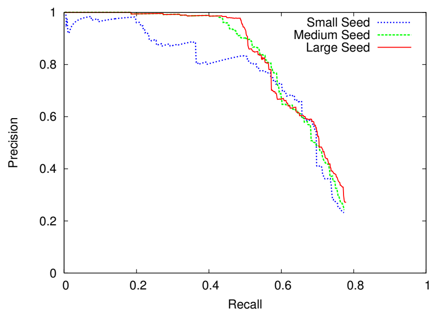
Likewise, Fig. 12 (b) shows QuestP (red color) outperforms baseline method with precision based scoring (other colors). Our QuestP method provides quite reliable ranking of templates — precision stays high for lower recall values, and gradually drops as recall increases. On the other hand, baseline method with precision based scoring ranks several non-relevant templates in top ranked positions, which results in poor precision even at low recall values.

Thus, we validate a key hypothesis of this paper, that—in order to mine templates, iterative bootstrapping must be interleaved and regulated via templates, as in our QueST network.

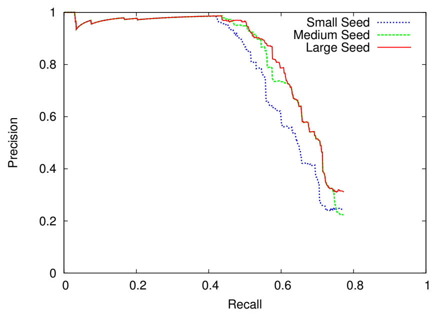
**Robustness to Seed Input Configurations:** Next, we study the performance of QueST framework under different input configurations. We report results for QuestP algorithm; the behavior for QuestR algorithm is similar. We present results only for *job* domain; similar robustness was observed in other domains as well. We can see from the P-R curve in Fig. 13 that our framework provides similar performance for all types of inputs. For variation in input size, we tend to get better results as the input size increases; however, the increase in performance is more significant from small (blue color) to medium (green color) than from medium



(a) Input sites

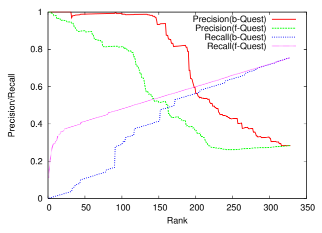


(b) Input queries

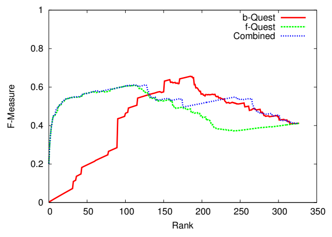


(c) Input templates

**Figure 13: Performance under different seed input configurations**



(a) QuestR vs. QuestP



(b) Combined Ranking

**Figure 14: The duality of Forward and Backward QueST**

| Domain | Input Site | Conf Intvl | Input Query | Conf Intvl | Input Template | Conf Intvl |
|---|---|---|---|---|---|---|
| Job | .82 | .82 ± .05 | .75 | .77 ± .06 | .83 | .83 ± .05 |
| Airfare | .79 | .78 ± .05 | .73 | .75 ± .05 | .77 | .78 ± .06 |
| Automobile | .81 | .80 ± .04 | .81 | .81 ± .03 | .86 | .85 ± .04 |
| CarRental | .91 | .89 ± .11 | .89 | .88 ± .10 | .91 | .89 ± .11 |
| Hotel | .76 | .77 ± .04 | .77 | .78 ± .05 | .78 | .78 ± .05 |
| Movie | .76 | .75 ± .06 | .72 | .73 ± .06 | .75 | .76 ± .06 |
| RealEstate | .76 | .76 ± .04 | .70 | .70 ± .04 | .75 | .76 ± .04 |

**Figure 15: Optimal F-measure over all domains.**

(green color) to large (red color)–indicating that medium size input already performs well.

We also tested the performance under hybrid input configuration, by providing a combination of medium size of sites, queries and templates together as input. Our framework tends to provide similar performance for this hybrid input configuration as well, with optimal F-measure of 0.87 in *job* domain.

We noted in Sec. 6 that our algorithm will converge to a unique solution. Indeed, we observe that, for all configurations of seed input, our algorithm converges in 4-5 iterations.

**Duality of Forward vs. Backward QueST:** We study the duality of recall-based (QuestR), *i.e.*, forward propagation, *vs.* precision-based (QuestP), *i.e.*, backward propagation on the QueST network. We report results for *job* domain for medium sized input.

We first compare the cumulative precision and recall at each ranked position for the two methods in Fig. 14 (a). As expected, QuestR finds out higher recall templates earlier. For example, the top 75 templates give a recall of 0.41 for QuestR (pink color) compared to 0.16 for QuestP(blue color). However, on precision metric, QuestP (red color) performs better than QuestR (green color). Thus, the two algorithms QuestR and QuestP optimize for recall and precision, respectively.

Next, we combine the two rankings to get an overall better performance. In particular, we studied $F$-measure scheme; for each template, we obtained combined score as harmonic mean of the predicted recall (by QuestR) and predicted precision (by QuestP). As shown in Fig. 14 (b), combined ranking (blue color) performs better on F-measure metric than each of the two methods (other colors). At top ranked positions, QuestR (green color) performs better; while at later ranked positions, QuestP (pink color) performs better. The performance of combined method, interestingly, follows the performance of the better of the two methods.

In summary, QuestR is better in finding popular templates whereas QuestP ranks more precise templates higher. Furthermore, better F-measure can be obtained by combining the two scores.

**Overall Performance in All Domains:** We present the optimal F-measure for QuestP algorithm for each of the 7 domains, in Fig. 15. While both the methods have their merit, we observed that QuestP achieves higher optimal F-measure compared to QuestR. We used medium size input configuration for this experiment.

We observed that for all types of input, and for all 7 domains, our method can achieve F-measure of 70-90%. Averaged over all domains, our method can achieve F-measure of 0.80, 0.81 and 0.76 using sites, templates and queries, respectively.

**Confidence Interval:** We also report 95% confidence interval for overall performance by partitioning the test set into 5 parts and evaluating each partition independently. Confidence interval is calculated as $Mean \pm (z - score * StandardError)$, where z-score =1.96 for 95% confidence interval and Standard Error is calculated as $(StandardDeviation/\ NumberofTrials)$. We can see the confidence intervals for the experiments in Fig. 15, which shows that our algorithm gives a stable performance.

## 7.3 Extensions: Incomplete Domain Schema

Our framework can be easily extended to application domains where domain schema is not fully available, *i.e.*, the list of attributes, or vocabulary instances are incomplete. In our iterative $S \rightleftharpoons Q \rightleftharpoons T$ framework, we can add another step of $T \rightarrow D$, to infer domain schema from templates.

We model this extension in two steps–by first generating candidate terms for schema extension using templates, and then grouping the candidates into homogeneous clusters. Several recent techniques on named-entity mining using search engine query logs [11, 12, 13] have shown the usefulness of templates in discovering new attribute classes as well as instances of existing attribute classes. Suppose, for example, in job listing domain, only 2 instances of #location were known, *e.g.*, $I(\#\text{location})$ = {chicago, seattle}.

*First*, we use "inferencing" query templates to obtain the list of candidate terms for inclusion in existing attributes, or for forming new attribute classes. We replace the attributes of existing template, *e.g.*, ⟨jobs in #location⟩ to formulate inferencing template, *e.g.*, ⟨jobs in $\star$⟩, where $\star$ can match any term. This inferencing template will match many more queries such as jobs in boston, jobs in microsoft, jobs in houston, jobs in yahoo, *etc.* Collecting the terms matching $\star$ position, we get the candidate list of terms for schema extension, *i.e.*, {boston, microsoft, houston, yahoo}.

*Second*, we cluster the candidate terms for schema extension based on their contextual similarity, *i.e.*, by comparing the overlap of the terms in query log that co-occur with each candidate term. In our implementation, we use Jensen-Shannon divergence (JSD), which is a smoothed and symmetric version of Kullback-Liebler divergence, as was also used in [12]. In this step, we will then be able to group the candidate terms into two separate groups: {boston, houston}, whose context is similar to existing instances of #location, and {microsoft, yahoo}, which will formulate the new attribute class of #company.

We evaluated our extension in two domains: *job* and *automobile*. For *job*, we provided only one attribute, #location, without giving #category. For *automobile*, we only specified #make and #year, without giving #model. For each of the attributes, we provided only 10 instances, instead of the full vocabulary in earlier experiments. We used QuestP algorithm for these experiments, using medium-size seed input, and sites as input type.

**Inferring Domain Schema:** For both the domains, not only our algorithm could discover all the missing attributes, it also discovered new attributes. In *job* domain, we discovered the missing attribute #category, and also a new attribute (not included in our original experiments) #company. Likewise, in Automobile domain, we discovered #model, and also a new attribute #location. We should mention that in Automobile domain, we observed four clusters of new attributes, of which one was #model, and the other three represented #country, #state and #city. Thus, the algorithm can also discover attribute classes at different levels of granularity.

**Impact on Quality of Templates:** With incomplete domain schema, the performance of basic QueST framework was 0.27 and 0.16 for Job and Automobile domain, respectively, as measured on F-measure metric. With our extension to infer domain schema, we could improve the corresponding performance to 0.75 and 0.73. While this is still lower than the performance when complete domain schema was available (0.82 and 0.81), it is significantly better than the performance with no extension.

## 8. CONCLUSION

As keyword queries have become the standard query language for searching over the Web, we believe rich query interpretation—understanding not only the intent but also the structures of keyword queries—is crucial for better search. In this paper, we formally define the concept of *query templates* and propose the problem of template discovery from search logs. We develop QueST, an inferencing framework over the graph of queries, sites, and templates to discover good templates. We define the quality measures of templates based on the dual metrics of precision and recall– and propose iterative inferencing using backward and forward random walks, respectively. We evaluated the system over a query log of 15 million queries from MSN search, and the results indicated high accuracy for query intent prediction. We look forward to extending the techniques for more complex and expressive query patterns.

## 9. REFERENCES

[1] Xiao Li, Ye-Yi Wang, and Alex Acero. Learning query intent from regularized click graphs. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, Singapore, Singapore, 2008. ACM.

[2] Steven M. Beitzel, Eric C. Jensen, Ophir Frieder, David D. Lewis, Abdur Chowdhury, and Aleksander Kolcz. Improving automatic query classification via semi-supervised learning. In *ICDE*, 2005.

[3] Microsoft Research. Microsoft live labs: Accelerating search in academic research 2006 rfp awards. *Research Grant*, 2006.

[4] Nick Craswell and Martin Szummer. Random walks on the click graph. In *SIGIR '07: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM Press, 2007.

[5] Ricardo Baeza-Yates. Applications of web query mining. *ECIR*, 2005.

[6] T. Joachims. Optimizing search engines using clickthrough data. In *KDD*, 2002.

[7] E. Agichtein, E. Brill, and S. Dumais. Improving web search ranking by incorporating user behavior information. In *SIGIR*, 2006.

[8] Charles Ling, Jianfeng Gao, Huajie Zhang, Weining Qian, and Hongjiang Zhang. Improving encarta search engine performance by mining user logs. *Journal of Pattern Recognition and Artificial Intelligence*, 2002.

[9] Ricardo Baeza-Yates, Carlos Hurtado, and Marcelo Mendoza. Query recommendation using query logs in search engines. *EDBT*, 2004.

[10] H. Cao, D. Jiang, J. Pei, Q. He, Z. Liao, E. Chen, and H. Li. Context-aware query suggestion by mining click-through and session data. In *SIGKDD*, 2008.

[11] Marius Pasca and B. Van Durme. What you seek is what you get: Extraction of class attributes from query logs. In *IJCAI*, 2007.

[12] 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.

[13] Gu Xu, Shuang-Hong Yang, and Hang Li. Named entity mining from click-through data using weakly supervised latent dirichlet allocation. In *KDD*, 2009.

[14] Ariel Fuxman, Panayiotis Tsaparas, Kannan Achan, and Rakesh Agrawal. Using the wisdom of the crowds for keyword generation. In *WWW*. ACM, 2008.

[15] Taher Haveliwala. Topic-sensitive PageRank. In *Proceedings of the Eleventh International World Wide Web Conference*, 2002.

[16] Dengyong Zhou, Olivier Bousquet, Thomas Navin Lal, Jason Weston, and Bernhard Sch Olkopf. Learning with local and global consistency. *Advances in Neural Information Processing Systems 16*, 16, 2004.

[17] M. Szummer and T. Jaakkola. Partially labeled classification with markov random walks. *Advances in Neural Information Processing Systems*, 2001.

[18] Jason K. Johnson, Dmitry M. Malioutov, and Alan S. Willsky. Walk-Sum interpretation and analysis of gaussian belief propagation. In *Advances in Neural Information Processing Systems 18*. MIT Press, 2006.

[19] Sergey Brin. Extracting patterns and relations from the web. *WebDB*, 1998.

[20] Oren Etzioni, Michael Cafarella, Doug Downey, Ana-Maria Popescu, Tal Shaked, Stephen Soderland, Daniel S. Weld, and Alexander Yates. Unsupervised named-entity extraction from the web: An experimental study. *Artificial Intelligence*, 165(1):91–134, 2005.

[21] Eugene Agichtein and Luis Gravano. Snowball: extracting relations from large plain-text collections. *ACM DL*, 2000.

[22] Mary Elaine Califf and Raymond J. Mooney. Relational learning of pattern-match rules for information extraction. *AAAI*, 1999.

[23] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project, 1998.

[24] Glen Jeh and Jennifer Widom. Scaling personalized web search. In *In Proc. of 12th WWW*, 2003.

[25] Xiaojin Zhu, Zoubin Ghahramani, and John Lafferty. Semi-supervised learning using gaussian fields and harmonic functions. *In ICML*, 2003.

[26] Uichin Lee, Zhenyu Liu, and Junghoo Cho. Automatic identification of user goals in web search. In *WWW*, pages 391–400, 2005.