A probabilistic approach for mapping free-text queries to complex web forms

Kien Tjin-Kam-Jet, Dolf Trieschnigg, and Djoerd Hiemstra

University of Twente, Enschede, The Netherlands
{tjinkamj,trieschn,hiemstra}@ewi.utwente.nl

Abstract. Web applications with complex interfaces consisting of multiple input fields should understand free-text queries. We propose a probabilistic approach to map parts of a free-text query to the fields of a complex web form. Our method uses token models rather than only static dictionaries to create this mapping, offering greater flexibility and requiring less domain knowledge than existing systems. We evaluate different implementations of our mapping model and show that our system effectively maps free-text queries without using a dictionary. If a dictionary is available, the performance increases and is significantly better than a rule-based baseline.

1 Introduction

When it comes to web applications, users love the 'single text box' interface because it is extremely easy to use. However, much information on the web is stored in structured databases and can only be accessed after filling out a web form with multiple input fields [2, 6]. From a user perspective, it would be valuable to provide access to these web forms by means of a single text box interface. Also from a technical perspective like distributed IR [4], accessing and retrieving structured data by forwarding a free-text query would be valuable. This work is about automatically mapping a *free-text query*, the input from a single text box (Fig. 1), to the fields of a multi-field, complex web form (Fig. 2).

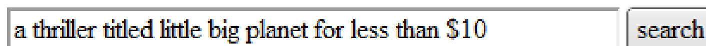

Fig. 1. A free-text interface for searching.

In contrast to previous *natural language interfaces*, which typically require grammatically correct sentences as input [3, 11, 5, 1], a *free-text interface* allows for shorter, ungrammatical input. A major limitation of current related work on free-text interfaces over structured data is that support for a complex web form containing free-text input fields *itself* is limited [7, 8, 17, 18, 20, 12, 10, 19, 14]. Typically, these approaches first scan the free-text query for dictionary entries

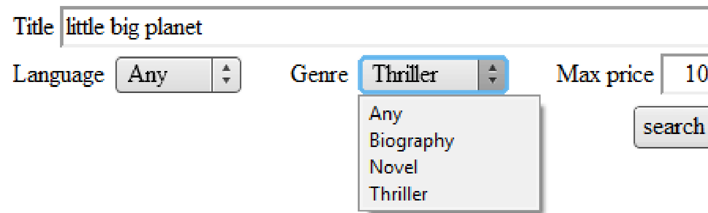

Fig. 2. A complex web form consisting of multiple input fields. It has a *free-text input field* of its own: the title field.

and then map those entries to the web form's input fields. Terms not found in the dictionary cannot be mapped to a field. Therefore, a system that does not know every dictionary entry for each field may wrongfully discard query terms (because the term should actually be mapped to a field). Especially for complex forms that have free-text input fields of their own, like the 'Title' field in Fig. 2, it is impossible to store all possible input in a dictionary. The dictionaries often serve as *the* mechanism for finding the right tokenization of a query. A different approach in web IR is query segmentation [9, 13], which uses statistics from large corpora (e.g., query logs or Wikipedia) and tries to group query terms into phrases in order to improve retrieval performance. However, this does not aim to map the detected phrases to fields. In this work, we adopt a rule-based tokenization step and a probabilistic mapping step. As such, we can effectively map and rank out-of-dictionary query terms to input fields.

Our contributions are as follows. First, we propose a model which supports the mapping of a free-text query to complex forms, including complex forms that contain free-text input fields. Second, we propose a probabilistic approach to rank the query-field mappings, which is based on a Hidden Markov Model. Third, we evaluate different implementations of our model and show that it can be effectively used in practice.

The rest of this paper is organized as follows. In Sect. 2 we describe the problem, our goal, and give an overview our approach. In Sect. 3 we outline the details of our approach. We describe our experiment in Sect. 4 and discuss our results in Sect. 5. In Sect. 6 we conclude this paper.

2 Goal and problem decomposition

Given a free-text query and a target web form with a set of input fields F, the goal is to find the best mapping from parts of the query to fields. The query is tokenized into *tokens*. A token is a contiguous part of the query that should be mapped to a field $f \in F$. F includes a special *junk* field for terms that should not be mapped to any actual field of the web form. We identify two problems. First, how to split a free-text query into tokens. Second, how to map these tokens to the intended fields. On a high level, our solution is to generate all tokenizations according to a tokenization strategy. Then generate all possible mappings for

each tokenization and rank these mappings based on their probability. In the following subsections we first give an intuitive model for formulating a free-text query, and then discuss the tokenization and ranking in more detail.

2.1 An action model for formulating free-text queries

We now define an action model which describes the actions a user takes to construct a free-text query. This model is used to obtain and rank possible interpretations of a free-text query. We define our action model as follows. Given a complex web form with input fields F, a user: *i*) decides which input field $f \in F$ to use. Then *ii*) decides what token to fill in f. Then *iii*) either decides to use an additional field, upon which the process repeats itself; or decides to use no more fields, and the process stops. Each field may be used at most once, except for the special *junk* field, which may be used any number of times.

For example, let us consider how a user formulated the free-text query shown in Fig. 1 for the complex web form shown in Fig. 2. According to our action model, the user first chose a `junk` field and then chose the token {a} for that field. He then chose the `genre` field and chose the token {thriller}. Then the `title` field with the token {titled little big planet}, and finally, the `max price` field with the token {for less than $10}. (*titled* and *for less than* serve as indicators, we discuss indicators in the next section.)

2.2 Tokenization

This section explains how to handle unknown terms in a query. Let us first introduce what can be recognized in a query: *i*) *dictionary entries*, i.e. terms found in a dictionary; *ii*) *hard separators*, i.e. a sequence of characters consisting of at least a punctuation and a white space character (the character set is denoted by the regular expression `[- \?!;,\.]`); and, *iii*) *indicators*, an indicator is a hint telling the system that an adjacent piece of text should be mapped to a specific field. Indicators reside either at the start or at the end of a token and are referred to as a prefix indicator or a postfix indicator, respectively.

In the scanning step, the system tries to recognize everything it can in the query. Any part of the query that was not recognized is called a *left-over*. The interesting part is how to treat the left-overs. We will investigate three methods:

Naive. Each left-over is considered a single token on its own.

Rigid. Each left-over is further segmented into possibly multiple tokens. The system selects one or more tokens to map to actual input fields; unselected tokens are automatically mapped to the junk field.

Tolerant. Like rigid, but now indicators can be also be placed *within* tokens created from left-overs. Only indicators at the start or end of a token are hints, indicators within tokens are no hints.

To illustrate the differences between these methods, imagine a system that only knows the prefix indicator ‘to’ and receives the query: “Boston to New York”.

We will use ~~striked~~ text to denote unselected tokens; surrounding braces { } to denote tokens; and surrounding brackets [ ] to denote indicators. The naive method will produce only one tokenization: {Boston} {[to] New York}. The rigid method produces two additional tokenizations: {Boston} {[to] New} ~~York~~, and {Boston} {[to]} ~~New~~ {York}. The tolerant method produces even more tokenizations: {Boston to New York}, {Boston to New} {York}, etc.

Tolerant tokenization is needed when an indicator is part of the actual content. For example, if one can search for book titles and the word 'titled' serves as an indicator, then only the tolerant method can yield the right tokenization for the query: "a titled maiden". In the naive and rigid methods, the indicator [titled] acts like a token separator, thereby splitting the input into two tokens.

2.3 Ranking with the Hidden Markov Model

The system uses a Hidden Markov Model (HMM) to rank the mappings assigned to a tokenization. Readers familiar with the HMM can skip to the next paragraph. A HMM is a probabilistic finite state automaton [15, 16]. It consists of a set of states, a set of transition probabilities from state to state, and a set of emission probabilities to model how each state produces some specific output. There are two special states: a start state and an end state. Except for the special states, each state emits one output symbol. The sequence of symbols can be observed, whereas the sequence of states cannot be observed, i.e. it is "hidden". Although the state sequence cannot be observed directly, a sequence of symbols gives some information about the hidden sequence of states. More specifically, beginning from the start state and finishing at the end state, the HMM generated the symbol sequence $O = o_1, o_2, \ldots, o_k$ by making $k + 1$ transitions from one state to the next. Each state emitted one symbol with some probability. In other words, we can compute the most likely sequence of states that produced the sequence O, if we knew the parameters of the HMM. Thus, the action model from Sect. 2.1 can be cast into an HMM problem: find the most likely sequence of input fields chosen by the user, given the observed sequence of tokens that the user has typed.

Building an HMM. Here we explain how to setup simple variants of the HMM parameters: the set of states, transition probabilities, and emission probabilities.

1. HMM states. Each state corresponds to an input field $f \in F$. Additionally, there is a start state and an end state.
2. Transition probabilities from one field to the next. Given a set of manually labelled free-text queries (e.g., each query is manually tokenized, and each token is manually labelled with the intended input field), we can learn the maximum likelihood estimate (MLE) $P_{MLE}(t_{ij})$ of the order of input fields f_i to f_j as follows:

$$P_{MLE}(t_{ij}) = \frac{\text{Number of times } f_i \text{ appears before } f_j}{\text{Total number of times } f_i \text{ appears}} \; .$$

3. Token emission probabilities. In our HMM, after each transition from one (previous) field to the current field, the current field will emit a token with some probability. If we have the set of tokens emitted at input field f_i, we can determine $P_{MLE}(e_{ij})$ for each token tok_j as follows:

$$P_{MLE}(e_{ij}) = \frac{\text{Number of times } tok_j \text{ is emitted at } f_i}{\text{Total number of tokens emitted at } f_i} \ .$$

Applying the HMM. Once the HMM is built, we can use it to output the most probable field sequence that could have generated a given token sequence. The learnt probabilities are smoothed beforehand, otherwise, an observation containing one token that was never seen during training will have a probability of zero. Smoothing is discussed in the next section. In any case, when applying the HMM, we must find the state sequence Q which maximizes the probability of the observation $O = o_1, o_2, \ldots, o_k$, given the model β:

$$Q \ = \ \underset{Q' \, \in \, \mathcal{Q}}{\arg\max} \ P(O|Q', \beta) \ P(Q'|\beta) \ ,$$

where $\mathcal{Q}$ denotes all possible state (field) sequences of length $k+1$ that could have generated the observation (tokens) O.

3 Token models and Smoothing

Each input field has its own dictionary of accepted tokens. As we explained in the introduction, a dictionary may not be exhaustive and a query could contain *unknown* tokens, i.e. tokens not listed in the dictionary. We need methods that could assign adequate probabilities to such unknown tokens. The MLE assigns a probability of zero to an unknown token, hence, it is not adequate for our purposes. One way to obtain non-zero probabilities for unknown tokens is to apply Laplace smoothing:

$$P_{Laplace}(tok) = \frac{\text{Number of times } tok \text{ appears in data} + \alpha}{\text{Total tokens in data} + \alpha * \text{total different tokens in data}} \ ,$$

where α is usually set to one. However, this method assigns the same probability to each and every unknown token: even two unknown tokens that look completely different from each other get the same probability. This is also not adequate for our purposes. Therefore, we will investigate several token models that could differentiate between unknown tokens.

3.1 Tokens models for discriminating unknown tokens

We now introduce three models based on n-grams and one based on a Poisson distribution. An assumption we must make when using n-grams, is that the probability of the next item depends only on the previous $n-1$ items. These models will be used to build the token emission models for each field.

Word *n*-grams approximate the emission probability for some token based on the word sequence in that token. Words are separated by (an optional punctuation mark (;,.!?) followed by) a white space.

Character *n*-grams approximate the probability that a field emitted a particular token, based on the sequence of characters of that token.

Word-length *n*-grams approximate the probability that a field emitted a particular token, based on the sequence of word-lengths in that token. As a variation, we can also consider the length *differences* between the words in a token, which can be measured as either absolute or relative differences. We can also decide to categorize the lengths or differences into, for example: zero, small, medium, and large lengths or differences. One benefit of using (categorized) length differences instead of plain lengths, is that our training data becomes less sparse.

Poisson models With n-gram models, to obtain the probability of a complete token, we must multiply the probabilities of each item in the token. Consequently, longer tokens usually have much smaller probabilities. For this reason, we adopt a Poisson model to assign probabilities based on the token's average word length, conditioned on the number of words in that token. We first bin all tokens: tokens consisting of one word go in the fist bin, tokens consisting of two words in the second, and so on. For each bin, we calculate the average word length. Unlike n-gram models which usually assign higher probabilities to smaller tokens, this model assigns higher probabilities if the token's average word length is closer to the average word length of the bin the token belongs to. For example, if the expected word length for 3-word tokens is 5 characters, and our unknown token (consisting of 3 words) has an average word length of 6.7 characters, then the Poisson probability of that token is $P_{Pois}(6.7, 5) = 0.11$.

3.2 Smoothing by using linear interpolation

With larger n, n-grams suffer more from data sparseness (i.e. when many items from the vocabulary are not present in the training data). One solution is to mix an n-gram with n-grams that have smaller n. For example, by mixing a trigram ($n = 3$), with a bigram ($n = 2$) and a unigram ($n = 1$) like:

$$P'_3(w_n|w_{n-2}, w_{n-1}) = \gamma_1 P_1(w_n) + \gamma_2 P_2(w_n|w_{n-1}) + \gamma_3 P_3(w_n|w_{n-2}, w_{n-1}) ,$$

where $0 \leq \gamma_i \leq 1$, and $\sum_i \gamma_i = 1$. Used this way, linear interpolation acts as a smoothing mechanism for n-grams, this is often referred to as *deleted interpolation*. Linear interpolation is also useful for combining different kinds of (smoothed) token models like:

$$P(tok) = \lambda_1 P'_{word}(tok) + \lambda_2 P'_{char}(tok) + \lambda_3 P'_{len}(tok) + \lambda_4 P_{Pois}(tok) ,$$

which in this case, assigns a probability to a token based on (smoothed) word, character and length n-gram models, and a Poisson model.

In case we know some of the tokens for a field, for example, we could have an incomplete dictionary, then we could artificially boost the score of the known token. We could incorporate this in the linear equation as:

$$P(tok) = \lambda_1 P'_{word}(tok) + \lambda_2 P'_{char}(tok) + \lambda_3 P'_{len}(tok) + \lambda_4 P_{Pois}(tok) + \lambda_5 P_{dict}(tok) \ ,$$

where $0 \leq \lambda_i \leq 1$, $\sum_i \lambda_i = 1$, and $P_{dict}(tok) = \frac{1}{|dict|}$ if $tok \in dict$, 0 otherwise.

4 Experiment

We evaluated our system in a travel planning scenario. The system had to find the best query interpretation without it knowing the actual Dutch train station names. Our token models were trained on non-Dutch train station names and should compensate for the lack of Dutch station names. We show that, using the system reported in [19], near-perfect mapping from free-text queries to fields is possible in this scenario if a list of Dutch station names is available (MRR 0.95). The research questions are: *i*) Does HMM ranking improve this further? And *ii*) what performance can be obtained if there is no list of station names?

4.1 Training data for building the HMM

The training data for the **token emission models** of the *from*, *to*, and *via* input fields, was a list of station names crawled from Wikipedia; it contained stations from Belgium, France, Indonesia, the UK, and Germany. The training data for the **token emission models** of the *junk* field was a list of words crawled from web blogs containing the words `ik`, `trein`, and `van` (in English: I, train, and from, respectively). We scraped the pages from the web blogs and used simple heuristics to create sentences: we split the text on certain HTML tags (like <br> and <p>), on a question or exclamation mark, or on a dot followed by white space. Only those sentences containing at least two station names were tokenized with the naive method from Sect. 2.2 and the left-overs constituted the training data for the token emission model of the *junk* field.

The training data for the **field transition model** was manually created, and based on the field sequences reported in [19]. Those sequences did not contain any 'junk' fields, so we created and added variations containing junk fields. For example, if the study reported that the transition sequence"*from-to*" appeared x times, then we added these sequences to our training data: "*junk-from-to*" $\frac{1}{2}x$, "*from-junk-to*" $\frac{1}{4}x$, and "*from-to-junk*" $\frac{1}{4}x$. This reflects our belief that it is more likely for "junk" text to appear at the start of a query.

4.2 Validation and test data

We used a list of Dutch train station names from Wikipedia to create queries; we randomly selected 50% of the stations for creating a validation set and the other 50% for creating a test set. For each set, we used a script to randomly

generate 50 train station name pairs (from and to) and 50 train station name triples (from, to, and via) for a total of 100 different *information needs*. As each information need can be *phrased* differently, we also generated 12 different query formulations based on query templates extracted from the query logs from [19]. As an example, say that we have the information need (“`Amsterdam`”, “`Hoek van holland haven`”), and two query templates “van `from` naar `to` op `date` om `time`”, and “naar `to` `from` rond `time`”. We substituted the station names in the corresponding slots of each template, and we simply replaced the time and date with 13:00 and 1-1-2012, respectively resulting in the *queries*: “van `Amsterdam` naar `Hoek van holland haven` op `1-1-2012` om `13:00`”, and “naar `Hoek van holland haven` `Amsterdam` rond `13:00`”.

4.3 Method – systems without station names

The same transition model (to score the sequence of field names) was used throughout all experiments. For each tokenization method, using the validation data, we applied a simple parameter sweep to find the best linear interpolation weights for the emission models (to predict the score of the tokens – see the final equation at the end of Sect. 3.2). Each λ could (initially) take a value of 0, 0.01, 0.1, or 1, which was then normalized. For example, if we combined the word and length models, we could have $0.01P'_{word} + 0.01P'_{len}$ which was normalized to $\frac{1}{2}P'_{word} + \frac{1}{2}P'_{len}$ (the character, Poisson, and dictionary models are ignored by setting their λs to zero). As another example, $0.01P'_{word} + 0.1P'_{len}$ would be normalized to $\frac{1}{11}P'_{word} + \frac{10}{11}P'_{len}$, and so on. We selected the systems with the highest MRR (mean reciprocal rank) in the validation data. Then we compared them with two baseline systems using the test data.

Note that the dictionary did not contain any station names. It was used to assign a constant score to the dates and times, otherwise the emission probability of those fields would be zero. Furthermore, we fixed the dictionary's weight at 0.1 so that all other components could have a lower (0.01), equal (0.1) or higher (1.0) contribution to the total token score.

4.4 Upper bound – systems with station names

We compared our validated systems C, with two other “baseline” systems A and B which can be considered as upper bounds. System A is the system reported in [19]: it knows all station names, ignores unknown parts of the query, and uses rules to rank the query interpretations. System B is almost like A, it knows all station names, ignores unknown parts of the input, but uses probabilistic ranking (i.e. the transition probabilities discussed in Sect. 2.3). The systems in C do not know the station names and must apply the tokenization methods and emission models which were introduced in Sect. 2.2 and Sect. 3, respectively. The same transition probabilities of system B are also used in C.

5 Results and Discussion

5.1 Validation results

Table 1 shows the emission models that yielded the highest MRR results per tokenization. It also shows the components of the emission model, their corresponding weights are shown in Table 2. The reported average MRR is averaged over all parameter combinations (i.e. non-zero weights for each component) of the given emission model. The total average MRR per tokenization is averaged over all parameter combinations of all emission models (not just the stated components). From these results we can see that, for example for the tolerant tokenization, it is better to use an emission model consisting of a character and a dictionary component rather than any other combination of components, since on average, these two components yield a much higher MRR. We can also see that, from naive to rigid to tolerant, the difference between the maximum and the average MRR grows. This difference is consistent across all our validation results. This can be explained by the fact that each tokenization method induces a search space. A smaller search space contains less erroneous answers, but on the other hand, it may often fail to include the right answer in the first place.

Table 1. Validation results: best emission models per tokenization method. The letters stand for word (W), character (C), length (L), Poisson (P), and dictionary (D). E.g., the emission model L+P+D consists of a length n-gram, a Poisson model and a dictionary.

	Tokenization		
	Naive	Rigid	Tolerant
Emission model	L+P+D	W+C+L+D	C+D
Maximum MRR	0.481	0.703	0.725
Average MRR	0.477	0.678	0.666
Total avg. MRR	0.455	0.584	0.447

Table 2. Weights of the corresponding components shown in Table 1.

	Normalized weights				
	Word	Character	Length	Poisson	Dictionary
Naive	0.000	0.000	0.048	0.476	0.476
Rigid	0.474	0.474	0.005	0.000	0.047
Tolerant	0.000	0.500	0.000	0.000	0.500

5.2 Test results

The test results are shown in Table 3. The baselines (upper bound A and upper bound B) are equipped with a dictionary containing all Dutch train station

names. Since they ignore unknown words, they have almost perfect performance; the only thing they do not know for sure is which station name should be mapped to which field. In any case, the high MRRs indicate that the ranking heuristics and the transition probabilities are well suited to the task at hand. By using the same transition probabilities and discarding the dictionary, we see the impact of tokenization and token modeling. The tolerant and rigid tokenization methods

Table 3. Test results.

	System	**MRR**
With dictionary	Upper bound A (heuristic)	0.953
	Upper bound B (probabilistic)	0.996
Without dictionary	System C-Naive	0.533
	System C-Rigid	0.738
	System C-Tolerant	0.732

perform much better than the naive method. This can be seen in both the validation and test runs. There is almost no difference between the rigid and the tolerant methods. A closer inspection of the data showed that the validation data contained train station names that contained indicators, while no names in the test data contained indicators. This explains why the tolerant method was better than rigid method according to the validation runs, but not according to the test results. This finding reveals two things. First, it confirms what we mentioned earlier, that the added value of the tolerant method over the rigid method is apparent when the tokens contain indicators. Second, even when tokens contain no indicators, the benefits of the rigid method over the tolerant method are small in terms of retrieval performance.

System B is significantly better than system A ($p <= 0.01$). Systems A and B are both significantly better than (all variants of) system C. Finally, both the rigid and tolerant methods are significantly better than the naive method.

5.3 Discussion

Generalizability. Why would someone want to use train station names from other countries as training data to model Dutch station names? Normally, they would not. We used a large list of train station names that did not contain the actual Dutch names to avoid overfitting. We may reasonably assume that if the training data looks more similar to the actual testing data, it will have a beneficial effect on retrieval performance. More importantly, the station and junk training data had sufficiently distinct characteristics that allowed the system to distinguish train stations from junk tokens. We believe that our approach is generic and that it would work well in other domains. Ideally we would train our models from query log data. Again, to prevent overfitting, we used junk tokens that were extracted from blogs instead of query logs.

Flexibility. The fact that baseline B performs even better than baseline A shows that *our approach can alleviate developers from spending effort in designing ranking heuristics* because it is capable of learning a suitable ranking function. Also, once the system is up and running it could continuously adapt its ranking function given the stream of query log data.

6 Conclusion and future work

We introduced and examined three tokenization methods and several token models. We proposed a probabilistic approach based on a Hidden Markov Model to rank mappings of tokens, i.e. contiguous parts of the query, to the fields of a complex web form. In contrast to previous work, our approach is able to map unknown (i.e., out-of-dictionary) tokens to fields; thereby offering greater flexibility and requiring less domain knowledge than existing systems. We can conclude that our probabilistic ranking improves over the rule-based baseline when our system is equipped with a dictionary. Also, even without a dictionary, we can still correctly interpret many queries; however, we *must* process the unknown tokens in a clever way, as the results show that the naive tokenization method is far inferior the other methods. In future work we will test our approach in more domains to verify that our approach is indeed generic and that it can be applied to more than just travel planner websites.

Acknowledgment

This research was supported by the Netherlands Organization for Scientific Research, NWO, grants 639.022.809 and 612.066.513.

References

[1] Androutsopoulos, I., Ritchie, G.D., Thanisch, P.: Natural language interfaces to databases – an introduction. Natural Language Engineering 1(01), 29–81 (1995)

[2] Bergman, M.K.: The deep web: Surfacing hidden value. Journal of Electronic Publishing 7(1) (August 2001)

[3] Burton, R.R.: Semantic grammar: An engineering technique for constructing natural language understanding systems. Tech. rep., Bolt, Beranek and Newman, Inc., Cambridge, MA. (Dec 1976)

[4] Callan, J.: Distributed information retrieval. In: Advances in Information Retrieval. pp. 127–150. Kluwer Academic Publishers (2000)

[5] Carbonell, J.G., Boggs, W.M., Mauldin, M.L., Anick, P.G.: The XCALIBUR project: a natural language interface to expert systems. In: IJCAI'83. pp. 653–656. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1983)

[6] Chang, K.C.C., He, B., Li, C., Patel, M., Zhang, Z.: Structured databases on the web: observations and implications. SIGMOD Rec. 33(3), 61–70 (2004)
[7] Dar, S., Entin, G., Geva, S., Palmon, E.: Dtl's dataspot: Database exploration using plain language. In: Proceedings of the 24rd International Conference on Very Large Data Bases. pp. 645–649. VLDB '98, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1998)
[8] Demidova, E., Fankhauser, P., Zhou, X., Nejdl, W.: Divq: diversification for keyword search over structured databases. In: SIGIR '10. pp. 331–338. ACM, New York, NY, USA (2010)
[9] Hagen, M., Potthast, M., Stein, B., Braeutigam, C.: Query segmentation revisited. In: Proceedings of the 20th international conference on World wide web. pp. 97–106. WWW '11, ACM, New York, NY, USA (2011)
[10] Hassan, M., Alhajj, R., Ridley, M.J., Barker, K.: Simplified access to structured databases by adapting keyword search and database selection. In: Proceedings of the 2004 ACM symposium on Applied computing. pp. 674–678. SAC '04, ACM, New York, NY, USA (2004)
[11] Hendrix, G.G., Sacerdoti, E.D., Sagalowicz, D., Slocum, J.: Developing a natural language interface to complex data. ACM TODS 3(2), 105–147 (1978)
[12] Kandogan, E., Krishnamurthy, R., Raghavan, S., Vaithyanathan, S., Zhu, H.: Avatar semantic search: a database approach to information retrieval. In: SIGMOD'06. pp. 790–792. ACM, New York, NY, USA (2006)
[13] Li, Y., Hsu, B.J.P., Zhai, C., Wang, K.: Unsupervised query segmentation using clickthrough for information retrieval. In: Proceedings of the 34th international ACM SIGIR conference on Research and development in Information. pp. 285–294. SIGIR '11, ACM, New York, NY, USA (2011)
[14] Meng, F.: A natural language interface for information retrieval from forms on the world wide web. In: ICIS. pp. 540–545. Association for Information Systems, Atlanta, GA, USA (1999)
[15] Rabiner, L.R.: A tutorial on hidden markov models and selected applications in speech recognition. Proceedings of the IEEE 77(2), 257–286 (1989)
[16] Rabiner, L.R., Juang, B.H.: Fundamentals of speech recognition (1993)
[17] Sarkas, N., Paparizos, S., Tsaparas, P.: Structured annotations of web queries. In: Proceedings of the 2010 international conference on Management of data. pp. 771–782. SIGMOD '10, ACM, New York, NY, USA (2010)
[18] Tata, S., Lohman, G.M.: Sqak: doing more with keywords. In: SIGMOD'08. pp. 889–902. ACM, New York, NY, USA (2008)
[19] Tjin-Kam-Jet, K., Trieschnigg, D., Hiemstra, D.: Free-text search over complex web forms. In: Multidisciplinary Information Retrieval. Lecture Notes in Computer Science, vol. 6653, p. 14. Springer Berlin / Heidelberg (2011)
[20] Zhou, Q., Wang, C., Xiong, M., Wang, H., Yu, Y.: Spark: adapting keyword query to semantic search. In: ISWC'07/ASWC'07. pp. 694–707. Springer-Verlag, Berlin, Heidelberg (2007)