Beyond Query: Interactive User Intention Understanding

Yang Yang, Jie Tang
Department of Computer Science and Technology, Tsinghua University
Tsinghua National Laboratory for Information Science and Technology (TNList)
SherlockBourne@gmail.com, jietang@tsinghua.edu.cn

***Abstract*—Users often fail to find the right keywords to precisely describe their queries in the information seeking process. Techniques such as user intention predictions and personalized recommendations are designed to help the users figure out how to formalize their queries. In this work, we aim to help users identify their search targets using a new approach called *Interactive User Intention Understanding*. In particular, we construct an automatic questioner that generates yes-or-no questions for the user. Then we infer user intention according to the corresponding answers. In order to generate "smart" questions in an optimal sequence, we propose the *IHS* algorithm based on heuristic search. We prove an error bound for the proposed algorithm on the ranking of target items given the questions and answers. We conduct experiments on three datasets and compare our result with two baseline methods. Experimental results show that *IHS* outperforms the baseline methods by 27.83% and 25.98% respectively.**

I. Introduction

With the exponential growth of information, we often fail to find the exact information that we are seeking for, even with the "powerful" search engines. Studies on two billion daily web searches show that approximately 28% of the queries are modifications of a previous query [18]. In addition, when users attempt to fill a single information need, 52% of them modified their queries [11]. Among those modifications, 35.2% totally changed the query, while 7.1% only added terms [21]. This indicates that users' needs are sometimes too vague to be stated clearly.

Psychologists and educators believe that questions-and-answers convey more precise information than mere statements [22]. This inspires us to explore an interactive question-and-answer approach to help users identify their needs. Specifically, we aim to construct an automatic questioner that generates questions for the user. We infer the user intention according to the returned answers. More formally, by assuming a hypothetical space that contains the user's target, we partition it according to the user's responses to the questions, and let it finally converge to the user's target.

This method, which we call *interactive user intention understanding*, is a novel approach and has not been studied. The major challenges include:

Question generation. Good questions should contribute to a partitioning of the hypothesis space. Also, a good question sequence should identify the target within a few rounds. Therefore, the core challenge here is how to generate an optimal sequence of questions.

User response. There are typically two types of answers that we can design: literal statements or option choices. Literal statements convey more information, at the cost of efficiency – they require more efforts for users to type in and more time for the system to process. Option choices are more user-friendly, but provide limited information. Thus another challenge is how to design proper answer types to balance the trade-off between these methods.

Interaction efficiency. The time cost of each interactive round includes: (1) time for the questioner to generate questions, (2) time for the user to response, and (3) time for updating the hypothesis space. In real applications the hypothesis space will be very large. How to design efficient algorithms to deal with large-scale data and make the interaction brief is another challenge.

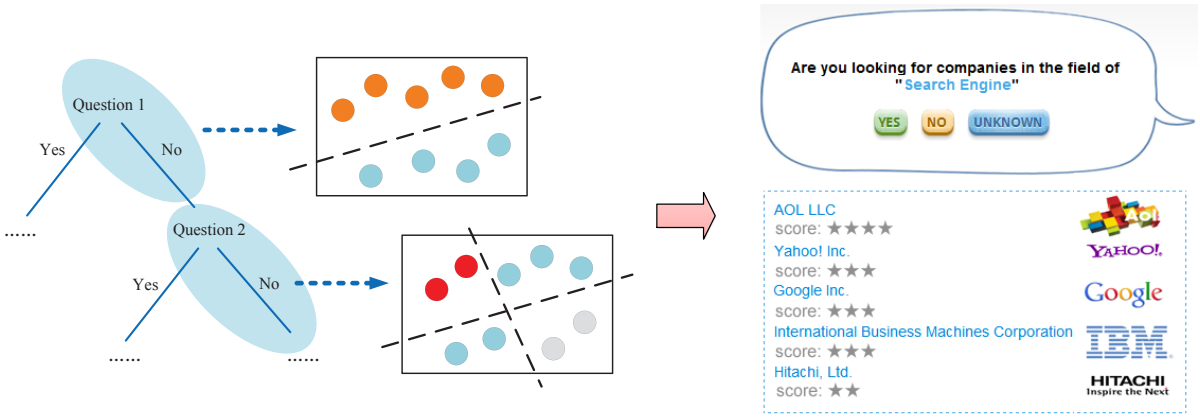
We address all three challenges in our approach, which only requires users to answer in binary forms; "yes" or "no." Figure 1(a) shows an example of the interaction process. Question 1 could be "Are you looking for companies in the field of Search Engine?" and suppose the user answers "no." Then the points standing for companies (lower part) will be assigned lower probabilities. A similar procedure is enacted for Question 2. We then obtain four regions with three different probability values: red points that are more likely to be the target and grey points that are less likely.

In this paper, we make three contributions:

First, we propose an interactive method that does not require typing. Simplified operation is especially meaningful for mobile users, considering the inefficiency of mobile typing as well as the rapid growth of the mobile Internet.

Second, we model the probability that the user may make mistakes and prove a bound of target ranking. Concretely, we realize that during the interactive process, the user may answer questions incorrectly by different reasons, for example mis-operations. We assume each user makes mistakes with some probability, and we bound the ranking of the target when we have asked some number of questions. The bound also depends on the probability of "mistaken answers".

Third, we design an Interactive Heuristic Search (IHS) method to generate sequential questions and partition the hypothesis space according to the user's answers. We compare the proposed algorithm with baseline approaches on three



(a) Ilustrative example. (b) Prototype system.

Figure 1. An example of the hypothesis space being partitioned by two questions. The image on the left stands for a decision tree made up of questions. The two boxes on the right represent two statuses of the same hypothesis space. Bright points have higher probability to be the target than pale points.

datasets. On average, IHS requires 7.47 questions less than others to identify the target. Experiments show that ignoring user response time, IHS takes only 0.16 second on average to finish an interactive round.

We have developed and deployed a web application for interactive user intention understanding based on our approach and the Patent dataset in PatentMiner[1] [24]. Figure **??** shows a screenshot of the prototype system. Users can find suitable companies for job-hunting or business analysis. We found that, in practical usage, the system needs fewer questions to identify users' intention. This might be because the system actually displays the top five companies during each questin round, enabling the user to find the target even when it does not rank first. On average, the questioner requires on average two to three questions to make an item rank first in the top five. Also, the system allows users to enter a query and search for some candidates first to limit the size of hypothesis space and reduce the number of questions required.

The paper is organized as follows. Section II reviews related literature. Section III formulates the problem. Section IV introduces our proposed algorithm. Section V gives the theoretical basis for the proposed algorithm. Section VI describes the experiments we conduct to validate the effectiveness of our methodology. Section VII concludes this work.

II. Related Work

Intention Prediction. Predicting a user's intention based on the user's query or other information is a challenging task. Bauer [2] introduced some typical methods that train agents to identify and extract interesting pieces of information from online documents. Fragoudis and Likothanassis introduced the retriever, an autonomous agent that executes user-queries and returns high quality results in [7]. This agent makes use of existing search engines and conducts self-training in order to analyze the user's preferences from semantic information.

Another approach developed by Chen [4] models and infers user actions. This approach not only considers keyword features, but also tries to form a concept hierarchy of keywords to improve performance. Other information agents like Office Assistant [9] and WebWatcher [1] use similar approaches.

Recommendation and Link Analysis. To understand users' intentions, various issues based on search engines and recommendation systems have been developed. Link analysis [17], [12], [14] is a data-analysis technique used to evaluate relationships between nodes in a network. Techniques like this will help the system identify popular items, which in turn helps users search for valuable items and information. Many recommendation engines have also been developed. Typically, users need to tell the recommendation engines their preferences on some items explicitly, since most of traditional recommendation methods rely heavily on user logs and feedbacks [13]. In [19], the authors performed an analysis on different item-based recommendation generation algorithms, including the computation of similarities between items and the way of obtaining recommendations. Meanwhile, they make comparisons between their results and the basic k-nearest neighbor approach [6], [5], which is based on collaborative filtering and is a popular recommendation system. However, recommendation engines may not produce meaningful recommendations when users cannot express their preferences accurately or there are no user logs available.

Related Machine Learning Algorithms. As we mentioned before, the core challenge of our work is the question selection problem, which is similar to that of active learning problems [20], [26]. However, we model the probability of

[1] http://pminer.org

users making mistakes in our work, which makes it different from active learning. Also, partitioning algorithms such as KD tree [16], [10] and learning algorithms such as neural networks [3] can be applied to solve the question selection problem. Another widely used algorithm is the greedy, which will be briefly introduced in Section IV as a baseline.

Systems involving the problem of finding a target item by asking yes-or-no questions have similar patterns. Usually the system asks the player a yes-or-no question, and the player answers it. Using the player's answer, the system uses some algorithm to find another question and presents it to the player. The selection of the question aims to minimize the total number of questions to be asked to identify the target item.

It is usually very useful to analyze the number of questions to be asked such that the target item is ranked in the top n. Such analysis requires considering the worst case in the procedure; in this game, it is the "worst answer" that the player could possibly give. When we consider the worst case in this game, it becomes similar to another well-known game called "Pusher-Chooser," introduced by Spencer [23], in which pusher has to find a balanced split, whereas chooser tries to take the advantage to introduce an imbalance.

III. Problem

In this section, we introduce some necessary definitions as well as the formulation of the problem. Suppose we have a set of n items as candidates for users to choose from, which are embedded in an m-dimensional tag space. Each tag can be viewed as an attribute or a feature of items. Formally, we use a tag matrix to represent the relations between items and tags:

Definition 1: **Tag Matrix.** A tag matrix $\mathbf{X}$ is an $n \times m$ matrix, where $x_{ij} = 1$ if and only if item i contains tag j.

To help the user to find the target item in her/his mind, we generate a number of Yes-or-No questions and ask the user to answer one-by-one.

Definition 2: **Yes-or-No Question.** A yes-or-no question can be regarded as a pair (q_i, l), where q_i is a tag used in the i-th question and $l \in \{0, 1\}$ stands for the user's answer. Here $l = 1$ denotes "yes" and $l = 0$ denotes "no."

As we mentioned before, we only use general questions that require user to answer "yes" or "no." The format for the i-th question is asking whether the *target item* the user is looking for contains tag q_i.

Problem&System: In this problem, we are given a tag matrix X, and our aim is to interactively understand the users' intentions in several iterations.

More precisely, we assume that there is a latent target item that best fits the users' intentions. In each round we present a yes-or-no question to the users. According to the answer provided by the user, we generate next questions to be presented. We also demonstrate a ranking list of items as recommendations to the user. Our goal is to minimize the number of questions to find the target item.

IV. Approach

A. Framework

We desribed our proposed approach in this section. The general idea is to assign each item a weight, which represents how likely the item is to be the user's target. In each question-and-answer round, we update the weights according to the user's answer and rank the items by their weights.

More formally, we use a vector $\mathbf{w^{(t)}}$ to denote items' weights in the t-th round, where $w_i^{(t)}$ denotes the weight of item i. The weights are initialized to be 1, i.e., $\mathbf{w^{(0)}} = \mathbf{1}$. In round t, for the i-th item that mismatches the answer, we decrease its weight $w_i^{(t)}$ by multiplying a *discount factor* γ ($0 \leq \gamma < 1$). For example, if the user answers that she is not interested in any items containing tag j, then all items containing tag j will receive a discounted weight. After updating, we present the ranking of items according to their weights in each round. The details of the framework can be found in Algorithm 1.

Considering users may make mistakes when they answer the questions (e.g., clicking yes-or-no buttons incorrectly), an item which mismatches the user's answer in one round may has chance to make a comeback if it matches all of the following answers. Thus, in most cases, we let the discount factor $\gamma \geq 0$ instead of defining $\gamma = 0$. Formally, we define the probability that a user makes a mistake (i.e., answer the question incorrectly) in one round as p.

Next we will introduce two algorithms for generating questions to ask the user.

B. Greedy

We first introduce a greedy algorithm as a baseline. The basic idea is, in each round, we choose the tag that can "eliminate" more candidates. In other words, we choose the tag which is able to mostly decreases $\left\|\mathbf{w}^{(t)}\right\|_1$.

Suppose we are asking the user if she is seeking for items with tag q, in which case the items are divided into two groups: one with tag q and the other without tag q. It follows that the corresponding total weight for these two groups are $\langle \mathbf{w}^{(t)}, \mathbf{X}_q \rangle$ and $\langle \mathbf{w}^{(t)}, \mathbf{1} - \mathbf{X}_q \rangle$ respectively, where $\mathbf{X}_q$ is the q-th column of the tag matrix $\mathbf{X}$. We then define a score α as follows:

$$\alpha = \min \left\{ \frac{\langle \mathbf{w}^{(t)}, \mathbf{X}_q \rangle}{\|\mathbf{w}^{(t)}\|_1}, \frac{\langle \mathbf{w}^{(t)}, \mathbf{1} - \mathbf{X}_q \rangle}{\|\mathbf{w}^{(t)}\|_1} \right\} \tag{1}$$

We want to maximize α to obtain the largest decline of total weight $\left\|\mathbf{w}^{(t)}\right\|$ when tag q is asked. We now define the expected value of each tag q at the t-th iteration as

$$\mathbb{E}(q) = \left\|\mathbf{w}^{(t)}\right\|_1^{-1} \langle \mathbf{w}^{(t)}, \mathbf{X}_q \rangle \tag{2}$$

Algorithm 1 The interactive recommendation framework.

Input: a tag matrix X, the maximum number of questions T, and the discount factor γ.
Output: an updated weight vector $\mathbf{w^{(T)}}$.

initialize weights $w_i^{(0)} = 1$ for all i;
t = 0;
repeat
  q = QuestionSelectionAlgo(X, $\mathbf{w^t}$, γ);
  Ask the user if she is seeking for items containing tag q;
  if the answer is "yes" **then**
    $w_i^{(t+1)} = w_i^{(t)} \gamma^{1-x_{iq}}$ for each item i;
  else
    $w_i^{(t+1)} = w_i^{(t)} \gamma^{x_{iq}}$ for each item i;
  end if
  $t = t + 1$;
until t = T
return $\mathbf{w^{(T)}}$;

From the property

$$\frac{1}{\|\mathbf{w}^{(t)}\|_1}(\langle \mathbf{w}^{(t)}, \mathbf{X}_q \rangle + \langle \mathbf{w}^{(t)}, 1 - \mathbf{X}_q \rangle) = 1,$$

we know that searching for q that maximizes α is actually maximizing $\min(\mathbb{E}(q), 1 - \mathbb{E}(q))$, which is also equivalent to minimizing $|\mathbb{E}(q) - 0.5|$. Thus the greedy approach is in fact searching for q such that

$$q = \arg\min_q |\mathbb{E}(q) - 0.5| \tag{3}$$

C. Interactive Heuristic Search

The limitation of Greedy is that it only considers and optimizes one-step strategy. Next we propose a heuristic algorithm that tries to optimize the question sequence.

Suppose at the beginning of each round we consider k future questions in advance. Similar to the greedy algorithm, our goal is to reduce $\|W\|_1$ as much as possible.

Let the tags used in future k questions to be $Q_k = \{q_1, q_2, \ldots, q_k\}$. Let the corresponding k answers to be $\mathbf{l} = l_1 l_2 \ldots l_k \in \{0,1\}^k$, where 1 stands for "yes" and 0 stands for "no".

After answering the t-th question, the weight of the i-th item is $w_i^{(t)}$. Then at the end of the $(t+1)$-th round, we have

$$w_i^{(t+1)} = \begin{cases} w_i^{(t)} \gamma^{1-x_{iq_t}} & \text{if the answer is "yes"} \\ w_i^{(t)} \gamma^{x_{iq_t}} & \text{if the answer is "no"} \end{cases} \tag{4}$$

where q_t is the tag contained in the t-th question.

To simplify, for the corresponding answer $l \in \{0,1\}$, we have

$$w_i^{(t+1)} = w_i^{(t)} \gamma^{l \oplus x_{iq_t}} \tag{5}$$

Therefore, after answering all k questions, the weight of the i-th item is:

$$w_i \prod_{j=1}^{k} \gamma^{l_j \oplus x_{ij}} \tag{6}$$

Hence the total decline of the weight according to answer vector $\mathbf{l}$ is

$$W[Q|\mathbf{l}] = \sum_i w_i (1 - \prod_{j=1}^{k} \gamma^{l_j \oplus x_{ij}}) \tag{7}$$

Since $\mathbf{l} \in \{0,1\}^k$, there are 2^k possible answer sequences. To construct the heuristic function, we consider the worst case, the one that produces a minimal reduction of weight. Thus the questions should be selected as:

$$Q^* = \arg\max_Q \min_{\mathbf{l} \in \{0,1\}^k} \{\sum_i w_i (1 - \prod_{j=1}^{k} \gamma^{l_j \oplus x_{ij}})\} \tag{8}$$

We then employ a heuristic search to find Q^*. Since the algorithm asks for a user's response in every round, we refer to it as *Interactive Heuristic Search*.

V. Theoretical Analysis

In this section we present some degenerate cases for the heuristic search discussed above. We first discuss the case when $k = 2$ – that is, the algorithm considers two future questions in advance in every round. We then present an even simpler case where $k = 2$ and $p = 0$ – that is, we assume that the answer always reflects the truth. We let the discount factor $\gamma = 0$; under this condition the heuristic function will degenerate to set functions.

At the end of this section we prove a bound on the ranking of target items given the questions and corresponding answers.

The Behavior of the Algorithm when k = 2. Next we consider a simple case of the proposed interactive heuristic search. Namely, the algorithm only considers $k = 2$ question in advance in each iteration. We make the deduction in a different way from the previous one, which results in a simpler heuristic function to be calculated.

Let q_1 and q_2 be the first two selected tags. l_1 and l_2 are the user's corresponding answers. According to the user's answer to q_1, we evaluate the decline of $\|\mathbf{w}\|_1$ as

$$W[q_1|l_1 = 1] = (1-\gamma)\langle \mathbf{w}, \mathbf{1} - \mathbf{X}_{q_1} \rangle \tag{9}$$
$$W[q_1|l_1 = 0] = (1-\gamma)\langle \mathbf{w}, \mathbf{X}_{q_1} \rangle \tag{10}$$

where we use $W[q|l]$ to denote the decline of the weight caused by tag q given the user's answer l.

For simplicity of the deduction and equations, we construct another weight vector $\mathbf{w}'$ and define $w_i' = w_i \gamma^{x_{iq_1}}$. We also let $W = \|\mathbf{w}\|_1$, $W' = \|\mathbf{w}'\|_1$, $W_0 = (1+\gamma)W$, and $W_a = \langle \mathbf{w}, \mathbf{X}_a \rangle$ for a tag a. Another important notation

is $W_{a\cap b} = \sum_i w_i(x_{ia} \wedge x_{ib})$. Thus we can simplify Eqs. 9 - 10 as

$$W[q_1|l_1 = 1] = (1-\gamma)(W - W_{q_1}) \quad (11)$$
$$W[q_1|l_1 = 0] = (1-\gamma)W_{q_1} \quad (12)$$

Similarly we evaluate the total decline of weight after asking q_1 and q_2 as

$$W[q_1 \,\&\, q_2|l_1 = 1,\ l_2 = 1] = (1-\gamma)(W_0 - \gamma W_{q_1} - \gamma W_{q_2} - (1-\gamma)W_{q_1\cap q_2}) \quad (13)$$
$$W[q_1 \,\&\, q_2|l_1 = 0,\ l_2 = 1] = (1-\gamma)(W + \gamma W_{q_1} - W_{q_2} + (1-\gamma)W_{q_1\cap q_2}) \quad (14)$$
$$W[q_1 \,\&\, q_2|l_1 = 1,\ l_2 = 0] = (1-\gamma)(W + \gamma W_{q_2} - W_{q_1} + (1-\gamma)W_{p_1\cap q_2}) \quad (15)$$
$$W[q_1 \,\&\, q_2|l_1 = 0,\ l_2 = 0] = (1-\gamma)(W_{q_1} + W_{q_2} - (1-\gamma)W_{q_1\cap q_2}) \quad (16)$$

Hence our selected question tags q_1 and q_2 is:

$$q_1, q_2 = \arg\max_{q_1,q_2} \min\{Eqs.13-16\} \quad (17)$$

which is a clearer and cleaner heuristic function.

The Behavior of the Algorithm when $k = 0$ and $p = 0$. Next we consider an even simpler case, where the user will always answer questions correctly ($p = 0$) and our algorithm considers two question in advance ($k = 2$). We also let $\gamma = 0$. Thus we exclude the items immediately if they mismatch the user's answers. We will see that in this case, the heuristic function degenerates into a set of set functions. The possible updates of weights are

$$W[q_1 \,\&\, q_2|l_1 = 1,\ l_2 = 1] = W - W_{q_1\cap q_2} \quad (18)$$
$$W[q_1 \,\&\, q_2|l_1 = 0,\ l_2 = 1] = W - W_{q_2} + W_{q_1\cap q_2} \quad (19)$$
$$W[q_1 \,\&\, q_2|l_1 = 1,\ l_2 = 0] = W - W_{q_1} + W_{q_1\cap q_2} \quad (20)$$
$$W[q_1 \,\&\, q_2|l_1 = 0,\ l_2 = 0] = W_{q_1} + W_{q_2} - W_{q_1\cap q_2} \quad (21)$$

The weights of all items are either 0 or 1; i.e., the weights become binary. We let S be the set of items that currently have weight 1. Let P_1 and P_2 be two sets covering the items that have tags q_1 and q_2 respectively; then W becomes the Cardinality function. I.e., we have

$$W = Card(S) \quad (22)$$
$$W_{q_1} = Card(P_1) \quad (23)$$
$$W_{q_2} = Card(P_2) \quad (24)$$
$$W_{q_1\cap q_2} = Card(P_1 \cap P_2) \quad (25)$$

And the decreased weight becomes

$$W[q_1 \,\&\, q_2|l_1 = 1,\ l_2 = 1] = Card(S) - Card(P_1 \cap P_2) \quad (26)$$
$$W[q_1 \,\&\, q_2|l_1 = 0,\ l_2 = 1] = Card(S) - Card(P_2) + Card(P_1 \cap P_2) \quad (27)$$
$$W[q_1 \,\&\, q_2|l_1 = 1,\ l_2 = 0] = Card(S) - Card(P_1) + Card(P_1 \cap P_2) \quad (28)$$
$$W[q_1 \,\&\, q_2|l_1 = 0,\ l_2 = 0] = Card(P_1) + Card(P_2) - Card(P_1 \cap P_2) \quad (29)$$

which follows that

$$q_1, q_2 = \arg\min_{q_1,q_2} \max \left\{ \begin{array}{c} Card(P_1 \cap P_2), \\ Card(P_1 - P_2), \\ Card(P_2 - P_1), \\ Card(S - P_1 \cup P_2) \end{array} \right\} \quad (30)$$

Notice that the equation above is quite intuitive: two sets P_1 and P_2 separate the hypothesis space into several parts, and we then search for the sets such that the maximum of these parts is minimized.

Ranking of the Target Item. We assume that the number of questions needed to find the target item is t. We let α be the lower bound of the ratio of the decrease over all iterations. More specifically, we let $W^{(1)}, W^{(2)}, \ldots, W^{(t)}$ be the total weights after each iteration, $l_1, l_2, \ldots, l_t \in \{0, 1\}$ be an arbitrary answer sequence. We then have

$$\alpha = \inf_{j,l_1,l_2,\ldots} \frac{\sum_i w_i^{(j)}(1 - \gamma^{l_{j+1}\oplus x_i})}{W^{(j)}} \quad (31)$$

Theorem 1: Let p be the probability that the user answers incorrectly. and γ be the discount factor. We use a sequence of random variables y_i to denote whether the user answers incorrectly in the i-th iteration. If the user answers incorrectly in the i-th iteration, $y_i = 1$; otherwise $y_i = 0$. Then

$$\Pr[y_i = 1] = p \quad (32)$$

Let $Y_t = \sum_{i=1}^{t} y_i$ and $y = \ln(\gamma)Y_t - t\ln(1-\alpha)$; the target item will surely be in the top L items with highest weights, where

$$L = \frac{n}{e^y} \quad (33)$$

the expected value of which is:

$$\mathbb{E}[L] = n[(1-\alpha)(1 + \frac{p}{\gamma}(1-\gamma))]^t \quad (34)$$

which means that the target item will surely be in the top $L = \frac{n}{e^y}$ items with highest weights, where L is a random variable. The expected value of L is $n[(1-\alpha)(1+\frac{p}{\gamma}(1-\gamma))]^t$.

However, this bound depends on α, which is hard to evaluate when we do not know the weight sequence $W^{(1)}, \ldots, W^{(t)}$.

Next we will show a bound on the expected value of L on a random tag matrix, which does not depend on α. Suppose we know the probability of each entry in the tag matrix being 1 is τ. That is, the tag matrix is generated with each entry being 1 with probability τ and 0 with probability $1-\tau$. The value of each entry is set independently. Then we have the following theorem:

Theorem 2: After answering t questions, the target item will surely be in the top L items with greatest weights, and the expected value of L has the following bound:

$$\mathbb{E}[L] \leq \frac{(2r)^t n}{2} e^s + 1 \tag{35}$$

where

$$r = ((2-4p)\tau^2 - (2-4p)\tau + 1 - p) \tag{36}$$
$$s = -t/4 + p(t-1)(1-p) \tag{37}$$

We can see that this bound on the expected value of L does not depend on α, and is thus easier to evaluate.

Please refer to Appendix for the proofs of Theorem 1&2.

VI. Experiments

A. Experimental Setup

We perform experiments on three datasets to compare the performance of Interactive Heuristic Search (IHS), Greedy, and Static Heuristic Search (SHS), the details of the former two algorithms can be found in Section IV. To show the power of interactions with users, we propose the third algorithm SHS. The question generation method of SHS is the same as IHS, but SHS only interacts with the user every two rounds. For example, at first, SHS will generate two questions q_1 and q_2. In the first round, we ask the user the question q_1. After the user has answered q_1, we then ask her the question q_2 directly. SHS then uses the obtained answers of q_1 and q_2 to generate the next two questions. Thus, unlike IHS, the generation of q_2 in SHS is independent with the answer to q_1.

We then introduce our three datasets respectively.

Random. In this synthetic dataset, the tag matrix is randomly constructed, with 1,000 items and 100 tags. Each element in the matrix is initialized to 1 with a manually defined probability τ and is initialized to 0 with the probability $1-\tau$.

Patent. In this dataset, we extract 13,091 companies and 3,770,411 patents from USPTO[2]. Companies are regarded as items and 428 categories of patents are tags. We then create the tag matrix as follows: if the company i owns at least one patent belonging to a specific category j, the corresponding element x_{ij} is 1; otherwise 0. Averagely, each item contains 17.2 tags.

ArnetMiner. This dataset consists of 36,371 conferences in computer science and 1,905,496 papers provided by ArnetMiner[3]. We then generate topic distributions, θ_v, for each conference v on 200 topics by the ACT model proposed in [25]. We treat each conference as an item and each topic as a tag. We set the entity in tag matrix $x_{vz} = 1$ if $\theta_{vz} > \beta$, where z is a topic and β is a threshold that is manually defined. In the tag matrix, each row contains 21.2 elements with value 1 on average.

For each dataset, we build a simulator (answerer) to answer questions. At first, the answerer randomly selects an item as the target. In each round, the simulator is given by a question q generated by the question selection algorithm (questioner). The answerer provides the answer to the questioner according to the value of x_{qv}. The interactive progress continues until the stop condition holds – i.e., the target ranks first (the weight of the target item is the largest) or the maximum number of questions has been reached.

B. Quantitative Analysis

In the Random dataset, we randomly pick 300 items as target one by one, and count the average number of questions used to identify each target, denoted as r. We vary the probability τ from 0.05 to 0.5, and see how the performance of different algorithms change.

In the Patent dataset, we first count the number of tags that "cover" the i-th item as

$$C(i) = \sum_{j=1}^{m} X_{ij} \tag{38}$$

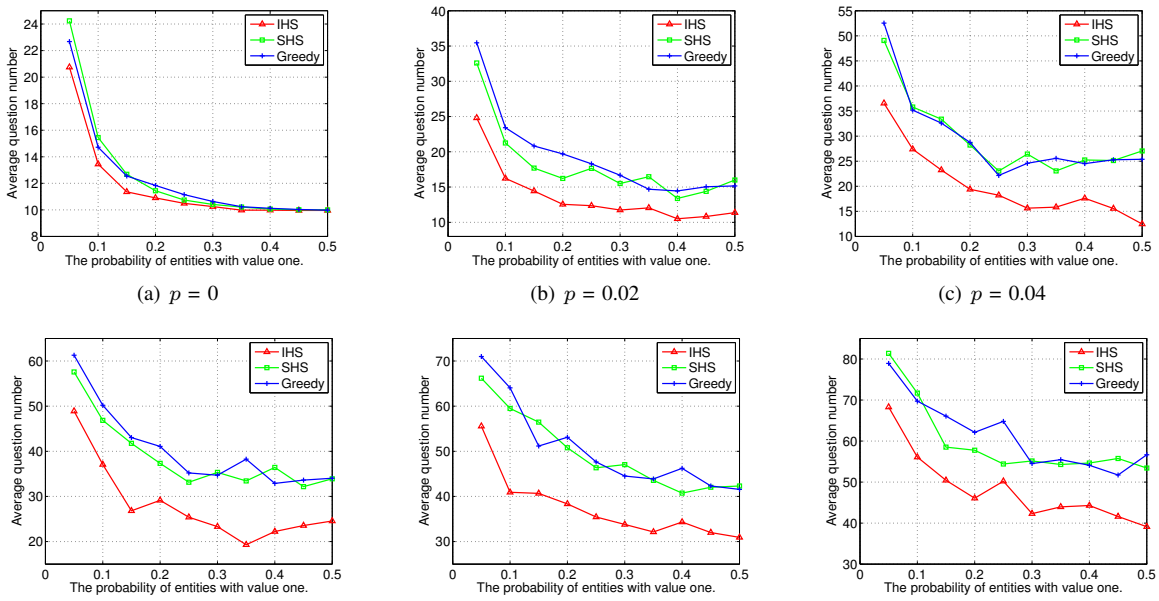
We rank all items according to $C(i)$. After that, we select the top 1,000 items with highest $C(i)$ as targets. We then select top 2,000 items and keep continue in this fashion to see how the range of target items influences the performance. Similar experiments are conducted on the ArnetMiner dataset. Besides, we also vary the p and see how user mis-operations affect the results. Figures 2 - 4 show the detailed results on three data sets respectively.

Firstly, we can see that IHS outperforms Greedy and SHS (8.07 and 8.86 fewer questions on average respectively), especially when the user answers incorrectly with higher probability; e.g., IHS uses 12.7 fewer questions than Greedy in Figure 3(f). In most cases, SHS requires more questions than Greedy (31.85 vs. 31.06 on average), which indicates that it is important to keep interacting with the user.

Secondly, from Figure 2, we can see that r generally decreases as τ increases. The reason is quite intuitive: if the tag matrix is very sparse – i.e., each item has very few tags – it is hard for the algorithm to find the tag that the item contains. Thus it costs more questions to find the target

[2]http://www.uspto.gov/, a public patent database in united states

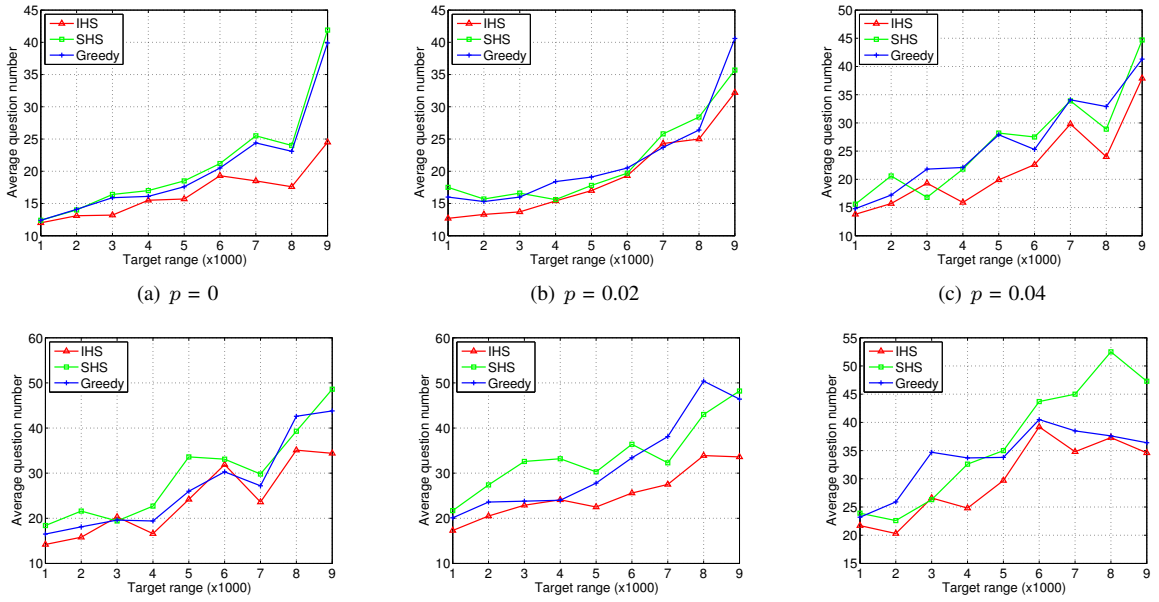
[3]http://aminer.org/



(a) $p = 0$ (b) $p = 0.02$ (c) $p = 0.04$

(d) $p = 0.06$ (e) $p = 0.08$ (f) $p = 0.1$

Figure 2. The performance of different algorithms in the Random dataset.



(a) $p = 0$ (b) $p = 0.02$ (c) $p = 0.04$

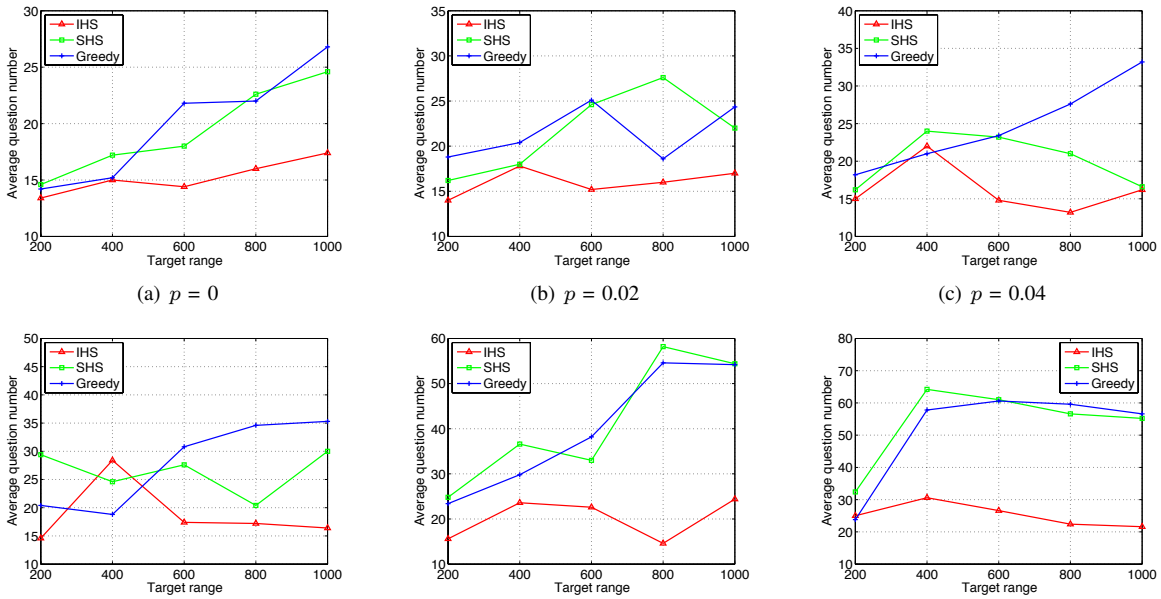
(d) $p = 0.06$ (e) $p = 0.08$ (f) $p = 0.1$

Figure 3. The performance of different algorithms in the Patent dataset.

item. In Figure 3 and Figure 4, r also decreases as the target range expands, which can be explained in the same way.

Thirdly, from all figures, we can easily find that r increases as p increases. It indicates that, if the user does not know much about the target item and keeps answering incorrectly, it is hard for the questioner to find the answer. We can also see curves in all figures tend to be monotonic and smoother with smaller p.

Another interesting fact is that, the gap between the performance of IHS and Greedy becomes larger as p increases. By a careful investigation, we find that when the user answers incorrectly, the weights of items that match the answer (we refer these item as a set S) do not change while the target item's weight decreases. To recover from this fault, the questioner must ask questions which can differentiate the



(a) $p = 0$ (b) $p = 0.02$ (c) $p = 0.04$

(d) $p = 0.06$ (e) $p = 0.08$ (f) $p = 0.1$

Figure 4. The performance of different algorithms in the ArnetMiner dataset.

Table I
EFFICIENCY STATISTIC (IN SECONDS).

Dataset	IHS	SHS	Greedy
Random	0.025	0.009	0.006
Patent	0.321	0.299	0.087
ArnetMiner	0.120	0.149	0.115

target item from items in S; otherwise the target will never rank higher than items in S. IHS performs better than Greedy since IHS considers more tags in one iteration, which makes it more likely to select a tag that can differentiate the target from the items in S. It also indicates that IHS has a stronger capability to compensate for mis-operations and has a more robust performance than Greedy.

C. Efficiency Analysis

To see how efficient our approach is, we count the average time used by Interactive Heuristic Search, Static Heuristic Search, and Greedy in each interaction round. Table I shows the results. In all datasets, IHS requires more time than Greedy and Static Heuristic Search. However, it takes less than 1 second (0.16 second on average) which can be tolerated in real applications. Static Heuristic Search costs almost half the time of Interactive Heuristic Search, as it generates a new pair of questions in every two rounds.

D. Qualitative Analysis

Now we present a case study to demonstrate the effectiveness of the proposed approach. Figure 5 shows an example generated from our experiments. It represents a number of interactive rounds where the target item is IBM. Several parameters and tags used in questions generated by different algorithms are shown in the figure, where $|W|$ denotes the sum of weights before the corresponding round starts, *rank* stands for the rank of the target item, and E denotes the expected value of the selected tag. Black arrows indicate the user's responses.

In the first round, Greedy selects a tag with expectation value close to 0.5, which is reasonable. In the next round, IHS outperforms Greedy, since all the tags cannot partition the hypothesis space well. IHS makes a relatively worse choice at first, but a better one in the next round. SHS chooses the same tag with IHS at first, but does not interact with the user after this round, thus loses helpful information. After the first two rounds, IHS reduces the weights most, and makes the target ranks top two.

VII. CONCLUSION AND FUTURE WORK

In this paper, we study a new problem, Interactive User Intention Understanding, to help users identify their target items. A heuristic-search-based algorithm is designed to generate questions. We also give a bound on the ranking of the target item after the user has been asked a number of questions. We then conduct a series of synthetic experiments on three datasets and show that our approach outperforms two baseline methods.

This is our first attempt in interactive user intention understanding, which has been largely unexplored to date. One challenge of this work is how to reduce time cost in each interaction round, which limits the total time of searching in our algorithm. An interesting and challenging idea is: can we build a decision tree corresponding to questions we will use

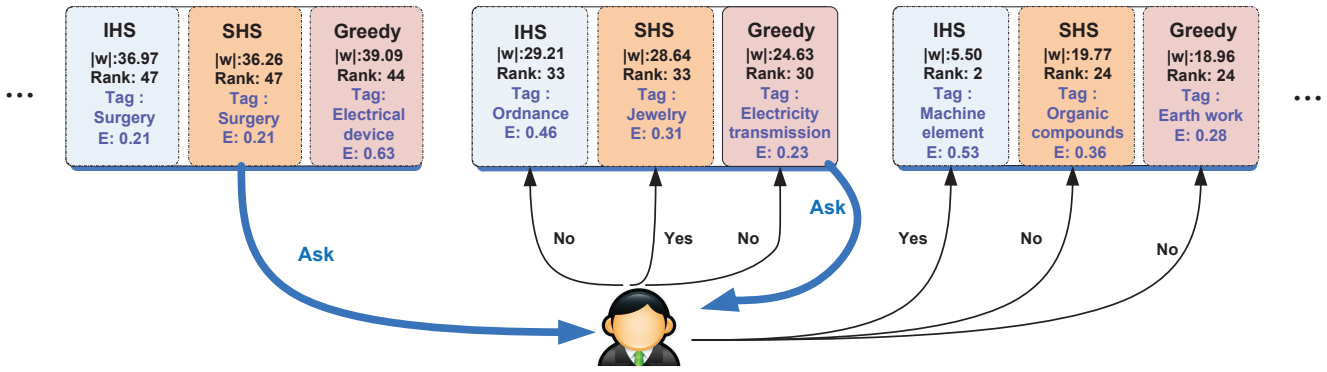


Figure 5. Case study. Portion of interactive rounds when the target company is IBM. $|W|$ denotes the sum of weights before the corresponding round starts; *rank* stands for the rank of the target item; *tag* is the selected tag used to generate questions; and E denoted the expectation value of the selected tag. Black arrows indicate the user's responses.

according to user responses? In this way it only cost $O(1)$ time to choose a question in each round, which sounds quite exciting.

Another idea is that, in the real world, the more popular an item is, the more likely it is to be the target. Thus we can first make an assumption of the distribution over all possible target items, and use the knowledge of this distribution to ask smarter questions.

Acknowledgements

The work is supported by the National High-tech R&D Program (No. 2014AA015103), National Basic Research Program of China (No. 2014CB340506, No. 2012CB316006), NSFC-ANR (No. 61261130588), Natural Science Foundation of China (No. 61222212), National Social Science Foundation of China (No. 13&ZD190), the Tsinghua University Initiative Scientific Research Program (20121088096), a research fund supported by Huawei Inc. and Beijing key lab of networked multimedia.

We thank Chengtao Li and Xun Zhen for valuable discussions and suggestions. Chengtao also contributed to the theoretical analysis of the proposed algorithm.

References

[1] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, A. Verkamo, et al. Fast discovery of association rules. *Advances in knowledge discovery and data mining*, 12:307–328, 1996.

[2] M. Bauer, D. Dengler, and G. Paul. Instructible information agents for web mining. In *IUI'00*, pages 21–28. ACM, 2000.

[3] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer-Verlag, 2006.

[4] Z. Chen, F. Lin, H. Liu, Y. Liu, W. Ma, and L. Wenyin. User intention modeling in web applications using data mining. In *WWW'02*, volume 5, pages 181–191, 2002.

[5] T. Denoeux. A k-nearest neighbor classification rule based on dempster-shafer theory. *Classic works of the Dempster-Shafer theory of belief functions*, pages 737–760, 2008.

[6] S. Dudani. The distance-weighted k-nearest-neighbor rule. *IEEE Transactions on Systems, Man and Cybernetics*, (4):325–327, 1976.

[7] D. Fragoudis and S. Likothanassis. Retriever: An agent for intelligent information recovery. In *ICIS'99*, pages 422–427, 1999.

[8] R. Graham, D. Knuth, and O. Patashnik. *Concrete mathematics: a foundation for computer science*, volume 2. 1994.

[9] E. Horvitz, J. Breese, D. Heckerman, D. Hovel, and K. Rommelse. The lumiere project: Bayesian user modeling for inferring the goals and needs of software users. In *UAI'98*, pages 256–265, 1998.

[10] W. Hunt, W. Mark, and G. Stoll. Fast kd-tree construction with an adaptive error-bounded heuristic. In *IRT'06*, pages 81–88, 2006.

[11] B. J. Jansen, A. Spink, and J. O. Pedersen. A temporal comparison of altavista web searching. *JASIST*, pages 559–570, 2005.

[12] J. M. Kleinberg. Authoritative sources in a hyperlinked environment. *J. ACM*, 46(5):604–632, Sept. 1999.

[13] D. Lemire and A. Maclachlan. Slope one predictors for online rating-based collaborative filtering. In *SDM'05*, 2005.

[14] R. Lempel and S. Moran. Salsa: the stochastic approach for link-structure analysis. *ACM Trans. Inf. Syst.*, 19(2):131–160, Apr. 2001.

[15] L. Lovász, J. Pelikán, and K. Vsztergombi. *Discrete mathematics: elementary and beyond*. Springer Verlag, 2003.

[16] A. Moore. An intoductory tutorial on kd-trees. *Extract from Andrew Moore's PhD Thesis: Effcient Memory based Learning for Robot Control*, 1991.

[17] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab, November 1999.

[18] G. Pass, A. Chowdhury, and C. Torgeson. A picture of search. In *Infoscale'06*, 2006.

[19] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In *WWW '01*, pages 285–295, 2001.

[20] B. Settles. Active learning literature survey. *Computer Sciences Technical Report 1648*, 2009.

[21] C. Silverstein, M. R. Henzinger, H. Marais, and M. Moricz. Analysis of a very large web search engine query log. *SIGIR Forum*, pages 6–12, 1999.

[22] R. Slavin. Cooperative learning. *Review of Educational*

research, 50(2):315–342, 1980.
[23] J. Spencer. Balancing games. *J. Combin. Theory Ser. B*, 1977.
[24] J. Tang, B. Wang, Y. Yang, P. Hu, Y. Zhao, X. Yan, B. Gao, M. Huang, P. Xu, W. Li, and A. K. Usadi. Patentminer: Topic-driven patent analysis and mining. In *KDD'12*, pages 1366–1374, 2012.
[25] J. Tang, J. Zhang, L. Yao, J. Li, L. Zhang, and Z. Su. Arnetminer: extraction and mining of academic social networks. In *KDD'08*, pages 990–998, 2008.
[26] S. Tong and D. Koller. Support vector machine active learning with applications to text classification. *JMLR'02*, 2:45–66, 2002.

VIII. Appendix

A. Theorem 1

Proof: Since in t iterations the user answers Y_t questions incorrectly, the weight of the target item is γ^{Y_t}. We know that at the t-th iteration, the total weight $W^{(t)}$ is no greater than $W^{(0)}(1-\alpha)^t = n[1-\alpha]^t$, because at least $\alpha W^{(i-1)}$ vanishes after answering the i-th question.

The target item will be surely ranking in top L after the t-th iteration if

$$L \times \gamma^{Y_t} \geq W^{(t)} = n(1-\alpha)^t \tag{39}$$

We could simply take

$$L = \frac{n(1-\alpha)^t}{\gamma^{Y_t}} \tag{40}$$

To get the expected value of L, we have

$$\begin{aligned}
\mathbb{E}[L] &= \mathbb{E}[n(1-\alpha)^t \gamma^{-Y_t}] && (41)\\
&= n(1-\alpha)^t \mathbb{E}[(\frac{1}{\gamma})^{Y_t}] && (42)\\
&= n(1-\alpha)^t \prod_{i=1}^{t} \mathbb{E}[\frac{1}{\gamma}^{y_i}] && (43)\\
&= n(1-\alpha)^t [1 + \frac{p}{\gamma}(1-\gamma)]^t && (44)
\end{aligned}$$

which is exactly the result we are looking for. ■

B. Theorem 2

Proof: We let the user answers $l_1, l_2, \ldots, l_t$ to these t questions where $l_i \in \{0, 1\}$. Assume that the target item is v (the v-th item). Using the definition of p, we have

$$\begin{aligned}
\Pr[l_i = x_{vi}] &= 1-p && (45)\\
\Pr[l_i = 1 - x_{vi}] &= p && (46)
\end{aligned}$$

Since each entry is set to be 0 or 1 independently, for $j \neq v$,

$$\begin{aligned}
\Pr[l_i = x_{ji}] &= p \times 2q(1-q) + \\
&\quad (1-p) \times (q^2 + (1-q)^2) = r && (47)\\
\Pr[l_i \neq x_{ji}] &= p \times (q^2 + (1-q)^2) + \\
&\quad (1-p) \times 2q(1-q) = 1-r && (48)
\end{aligned}$$

and it follows that $r > 1 - r$.

We let N_j be the number of tags that in which item j is different from $l_1, l_2, \ldots, l_t$. Then we have

$$\begin{aligned}
\Pr[N_j = w] &= \binom{t}{w}(1-r)^w r^{t-w} && (49)\\
&< \binom{t}{w} r^t && (50)
\end{aligned}$$

An item j ($j \neq v$) ranks higher than the target item v if and only if $N_j < Y_t$. The intuition for this is that item j has more tags that match the user's answers $l_1, \ldots, l_t$ than item v. Therefore we have

$$\begin{aligned}
\Pr[N_j < Y_t] &= \sum_{i=0}^{Y_t - 1} (1-r)^i r^{t-i} \binom{t}{i} && (51)\\
&< r^t \sum_{i=0}^{Y_t - 1} \binom{t}{i} && (52)\\
&< r^t \times 2^{t-1} \frac{\binom{t}{Y_t}}{\binom{t}{\frac{t}{2}}} && (53)\\
&\leq \frac{(2r)^t}{2} e^{-\frac{(t/2 - Y_t)^2}{t}} && (54)
\end{aligned}$$

where Eqs. 53-54 draw from [15] [8].

Since the rank of the target item L is exactly the number of items that rank higher than it plus 1, we have

$$\begin{aligned}
\mathbb{E}[L] &= n \times \mathbb{E}[\Pr[N_j < Y_t]] + 1 && (55)\\
&\leq \frac{(2r)^t n}{2} \mathbb{E}[e^{-\frac{(t/2 - Y_t)^2}{t}}] + 1 && (56)\\
&\leq \frac{(2r)^t n}{2} e^{\mathbb{E}[\frac{-(t/2 - Y_t)^2}{t}]} + 1 && (57)\\
&= \frac{(2r)^t n}{2} e^{-t/4 + p(t-1)(1-p)} + 1 && (58)\\
&= \frac{(2r)^t n}{2} e^s + 1 && (59)
\end{aligned}$$

where Eq. 57 draws from Jensen's Inequality.

■