Session Search by Direct Policy Learning

Jiyun Luo, Xuchu Dong, Hui Yang
Department of Computer Science, Georgetown University
{jl1749,xd47}@georgetown.edu, huiyang@cs.georgetown.edu

ABSTRACT

This paper proposes a novel retrieval model for session search. Through gradient descent, the model finds optimal policies for the best search engine actions from what is observed in the user and search engine interactions. The proposed framework applies direct policy learning to session search such that it greatly reduce the model complexity than prior work. It is also a flexible design, which includes a wide range of features describing the rich interactions in session search. The framework is shown to be highly effective evaluated on the recent TREC Session Tracks. As part of the efforts to bring reinforcement learning to information retrieval, this paper makes a novel contribution in theoretical modeling for session search.

Categories and Subject Descriptors

H.3.3 [**Information Systems**]: Information Storage and Retrieval—*Information Search and Retrieval*

Keywords

Dynamic Information Retrieval; Policy Learning; Session Search

1. INTRODUCTION

Session search is a complex information retrieval (IR) task. It is the information retrieval task that aims to find relevant documents for a session of multiple queries. Session search often happens when a user searches for an information need containing multiple aspects. For instance, in the Text REtrieval Conference (TREC) 2013 Session Track (Table 1 session 2), the information need is about *purchasing scooters*, which composes of five sub information needs on *brands, store names, price information, quality* and *reliability*; the user issued 8 queries to accomplish the search task. Session search also happens when an information need is vague or exploratory in nature. For example, in Table 1 session 87, the information need is *planning a trip to the United States for a month and able to travel within 150-mile radius of the destinations.* The user issued 21 queries in the session with various search attempts, such as checking the map and searching for the distances between two cities.

ICTIR'15, September 27–30, 2015, Northampton, MA, USA.
 ISBN 978-1-4503-3833-2/15/09 ...$15.00.
DOI: http://dx.doi.org/10.1145/2808194.2809461.

Table 1: Examples in TREC Session Tracks

TREC'13 session 2, Information need:

You want to buy a scooter. So you're interested in learning more facts about scooters including: what brands of scooters are out there? What brands of scooters are reliable? Which scooters are cheap? Which stores sell scooters? which stores sell the best scooters?

query1: scooter brands	**query2:** scooter brands reliable
query3: scooter	**query4:** scooter cheap
query5: scooter review	**query6:** scooter price
query7: scooter stores	**query8:** where to buy scooters

TREC'13 session 87, Information need:

Suppose you're planning a trip to the United States. You will be there for a month and able to travel within a 150-mile radius of your destination. With that constraint, what are the best cities to consider as possible destinations?

query1: best us destinations	**query2:** distance new york boston
query3: maps.bing.com	**query8:** hartford visitors ...

query9: hartford connecticut tourism ...

query19: philadelphia nyc travel	**query20:** philadelphia nyc train
query21: philadelphia nyc bus	

TREC'14 session 1011, Information need:

... You would like to provide your friend with relevant information about: different ways to quit smoking, programs available to help quit smoking, benefits of quitting smoking, second effects ...

query 1: quit smoking	**query 2:** quit smoking hypnosis

query 3: side effects quit smoking

Session search is different from classic ad-hoc retrieval. Ad-hoc retrieval focuses on finding relevant documents for a single query. Session search aims to retrieve relevant documents for the entire session. Not surprisingly, in a session, ad-hoc retrieval, even with a relevant feedback scheme, would treat each query independently. On the contrary, session search takes the challenge to handle the dependency between queries and the dependency between documents retrieved at different iterations. Their differences also lie in that session search aims to optimize a long term reward,

which is an expectation over the overall rewards in the whole session, while ad-hoc retrieval doesn't have to do that.

In a typical search session, search happens in episodes, which we term "search iterations." Each search iteration is a complete cycle of ad-hoc retrieval. A user issues a series of queries $q_1, q_2, \ldots, q_n$, and the search engine retrieves a series of document lists, $D_1, D_2, \ldots, D_n$, for the corresponding queries. The user skims a document list and clicks on some documents, which yields clicks and other implicit feedback, such as the time spent on reading the documents (also known as *dwell time*). The clicks also form a sequence, $C_1, C_2, \ldots, C_n$, for the corresponding retrievals. Between the neighboring queries, the user performs query reformations to re-write the earlier query or search along a different path. All these signals, including queries, retrieved documents, clicks, viewed documents, dwell time, and query reformulations, intertwine together and jointly impact on the session. Given the high complexity of the task, session search demands novel retrieval algorithms which are able to capture the characteristics of search sessions and able to enhance session search effectiveness.

In a session, users often search in a *trial-and-error* fashion: repeatedly trying different search paths via writing various queries, until succeeding in finding relevant documents to satisfy the information need. The search engine receives immediate, instant feedback from the user at each search iteration and hopes to return documents that are relevant for a long term search goal. Such characteristics of session search make it fit well with the family of Reinforcement Learning (RL) algorithms [28].

Recent development in session search algorithms ([9], [22]) show that using RL for session search is a promising new direction. However, existing approaches often trade model complexity off for efficiency. Reducing the number of states and actions is often a must-do to achieve real-time interactions between the user and the search engine. For instance, only four decision-making states are modeled in [22]. In the pursuit of efficiency, an obvious drawback of the prior work is that the details of the rich user-search-engine interactions could get lost.

In this paper, we propose a novel retrieval model for session search. We aim to create a framework that is able to incorporate a variety of features to adequately represent the complex interactions in a session. Our approach is based on partially observable Markov decision processes (POMDPs). It is a direct policy learning algorithm that skips the trouble of calculating beliefs for states, but directly learns the optimal policies from observations obtained easily in the user and search engine interactions. During every search iteration, the search engine maximizes the long-term reward towards the ultimate information need, instead of finding a local maximum for the current query. The framework learns a mappings via gradient descent from observations to actions, and sequentially handles a search iteration in three phases, including a browse, a query, and a rank phase. By learning over a history of actions and observations, our algorithm is able to efficiently learn policies with less model complexity than prior work. Experiments with the TREC 2012-2014 Session Tracks demonstrate that our algorithm exhibits a statistically significant improvement over several state-of-the-art session search systems. To our knowledge, this paper is the first to study direct policy learning for session search.

The rest of the paper is organized as follows. Section 2 presents related work. Section 3 describes the optimization framework. Section 4 presents the formulation of history and the three phases. Section 5 details the proposed direct policy learning algorithm. Section 6 evaluates the approach and Section 7 concludes the paper.

2. RELATED WORK

As a challenging IR task, session search has received increasing attention from both academia and industry in a number of IR fields, such as user and task modeling [19, 20, 29], retrieval models [24], interactive search [1, 17], query log analysis [7], and evaluation [8, 10]. The TExt Retrieval Conference (TREC) has encouraged a good deal of research in session search by organizing the Session Track since 2010 [13, 15]. Efforts from industry also have promoted the broadening of research in session search through the recent WSCD (Web Search Click Data) workshops.[1]

Many session search methods are based on large query logs. [3] derived a query-flow graph, a graph representation of user behavior, from query logs. The approach detected query chains in the graph and recommended queries based on maximum weights, random walk, or the previous query. [24] employed sub-modular optimization for document ranking for a specific type of sessions – sessions showing multiple subtopics (which they termed as having "intrinsic diversity"). [21] broke the session into partitions that correspond to subtasks. Other mining approaches [3, 27] identify the importance of query change in sessions; however, they require the luxury of large query logs. In the top TREC systems, for instance, the best TREC 2012 system [13] employs an adaptive browsing model by manipulating past retrieved documents; however, it did not demonstrate how to adapt to changing user intent, which sometimes requires relevant documents and sometimes requires novel documents. In the WSCD 2014 workshop, the best submitted system took a learning-to-rank approach [23]. The features being used included document ranks in the original ranking, query statistics, clicks, missed clicks, snippet quality, user click habits, and workweek and weekend seasonality. They claimed that there were session-level features, too; in the paper, they were not clearly listed. However, this supervised learning method assumed the batch relevance feedback were available, which is not true for the online learning setting in session search.

In this paper, we propose a novel session search framework based on a partially observable Markov decision process, which belongs to the family of reinforcement learning (RL). In RL, agents take inputs from the environment and output actions. The actions in turn influence the states of the environment. An RL algorithm often performs optimization over actions and policies, based on long-term benefits for solving problems with uncertainty, and satisfying the Markov property. The environment in reinforcement learning is usually formalized as a Markov Decision Process (MDP) [28], which is able to support modeling temporal dynamics in tasks like session search. Basic RL algorithms include model-based algorithms, such as value iteration and policy iteration, and model-free methods, such as Q-learning and TD-learning [14]. Reinforcement Learning has been successfully applied to robotics and artificial intelligence in general [14] such as

[1] https://www.kaggle.com/c/yandex-personalized-web-search-challenge

natural language grounding [5]. However, RL algorithms can be computationally demanding. Therefore, for problems with a large state space or action space, generalization is often used to reduce the state-value or state-action mappings [25]. Policy gradient descent [2] is a popular direct policy search approach. It optimizes the parameters via gradient descent, and its parameterized function can be learned through Monte-Carlo sampling, by simulating the interactions between agents and the environment. In this paper, we adopt policy gradient descent as a key technique in the proposed framework.

Researchers have just started to explore RL in IR. There are only a few existing approaches that use RL to study IR problems. The related approaches include using MDP [9], POMDP [22], exploratory online learning [11] and decision theories [26], to study interactive search. Although not all of them are based on RL, they all share the idea of using states, actions, and rewards. Among them, [22] is the most similar to our work. The authors formulated session search into classic POMDPs with hidden decision making states, actions, observations and observation functions. Their formulation was complex. Our work differs from theirs in that we learn direct mapping from observations to actions, which greatly simplifies the learning process. Moreover, our model is not limited to use states to capture the dynamic changes in session search as what [22] did. Instead, our framework is flexible enough to add a wide range of features that describe the session search process with richer details.

3. FRAMEWORK

Our framework is built on top of the Partially Observable Markov Decision Process (POMDP) [14]. A POMDP is represented by a tuple, $<S, G, A, O, B, T, R>$, which indicates states, agents, actions, observations, and beliefs over the states, transitions, and rewards. The states S (actions A) are a finite set of discrete states (actions). Observations O is a finite set of discrete symbols that an agent observes about the states. In a POMDP, hidden information can be modeled as states, while visible signals can be modeled as observations or actions. Reward $r = R(s_t = s, a_t = a, s_{t+1} = s')$ is the immediate reward when transitioning from state s to s' by taking action a at time t. The agents take inputs from the environment by making observations and output actions, which in turn influence the environment. In session search, there are in fact two agents, the user and the search engine [9]. However, since they do not appear as the main actor at the same time in a session, we merge them into one agent and skips the complexity of handling two.

Here our states are search iterations. Actions include browsing documents, such as reading and clicking documents, writing queries, and retrieving documents. Every state and action pair receives an immediate reward at the state transitions. The POMDP aims to optimize the long term overall rewards for the entire process.

To solve an RL problem is to find the best policy for it. This is called "policy learning." In a POMDP, a policy π describes the general rules of which actions for each agent to perform at each time step, given a history of state and action pairs since time 0. The goal of policy learning is to find a policy that optimizes the expected discounted long-term cumulative reward – i.e., the value function

$$V_\theta(s_0) = E\big(\sum_{t=0}^{\infty} \gamma^t r(t) | s_0\big) \quad (1)$$

where γ is discounts the expected future or past rewards, since they could be inflated now. $r(t)$ is the immediate reward at time t. s_0 is the starting state.

If we denote by H the set of all possible history of search behavior sequences h and $r(t, h)$ the t^{th} reward in the history h, the value function in Eq. 1 can be written as:

$$V_\theta(s_0) = \sum_{t=0}^{\infty} \gamma^t \sum_{h \in H} P(h|\theta) r(t, h) \quad (2)$$

where $P(h|\theta)$ is the probability distribution over all possible histories given the parameter vector θ. $r(t, h)$ is the reward obtained at time step t. γ is the discount factor and t indexes the search iterations.

The best policy π^* can be induced from optimization of $V_\theta(s_0)$, which is equivalent to learning the best parameter vector θ. The value function can optimized by solving the Bellman equation if the model is known [9]. When the model is not known, there is no close form solution to it and we need to use gradient descent to find the optimal solution.

Two challenges of our framework are 1) defining the history and calculating $P(h|\theta)$, and 2) optimizing the value function via gradient descent. The former is addressed in Section 4 and the latter in Section 5.

4. HISTORY AND THREE PHASES

Given that many factors in session search, such as a set of continuous queries, documents, and clicks, it is non-trivial to clearly describe the dynamics in it. To tackle this, we propose to decompose a search history into two layers. The first is to break the history into search iterations, indexed by t. Further, an iteration is decomposed into three phases: the browse, the query, and the rank phases. The first two phases are led by the user and the last by the search engine. With such a decomposition, we are able to perform further study to model the search at a more fine-grained level.

4.1 History

We define a history to be the record of a session from search iteration 0 to the current search iteration t.[2] A *history* keeps track of the dynamic changes of states, observations, actions, and rewards during session search. It is a chain of events happening in a session. Every search iteration in the history consists of clicks C, dwell time T, query q, query changes Δq, and retrieved documents D. The history at time t can be written as:

$$h_t = [h_{t-1}, C_t, T_t, q_t, \Delta q_t, D_t]$$

where h_{t-1} is a prefix history in h_t.

Figure 1 illustrates a history gathered for TREC 2014 session 1011. The information need is s *You would like to provide your friend with relevant information about: different ways to quit smoking, programs available to help quit smoking, benefits of quitting smoking, second effects.* The session started with the first query (q_1) *quit smoking.* The history up to the end of the first iteration is h_1, which contains q_1 and a set of documents D_1 returned by the search engine. We assume a dummy starting query q_0. Therefore, the new added terms are just q_1 and the removed terms are none.

The second search iteration started when the user browsed the previously returned document set D_1. The user clicked

[2]We use 'iteration' and 'time step' interchangeably to refer to search iteration t.

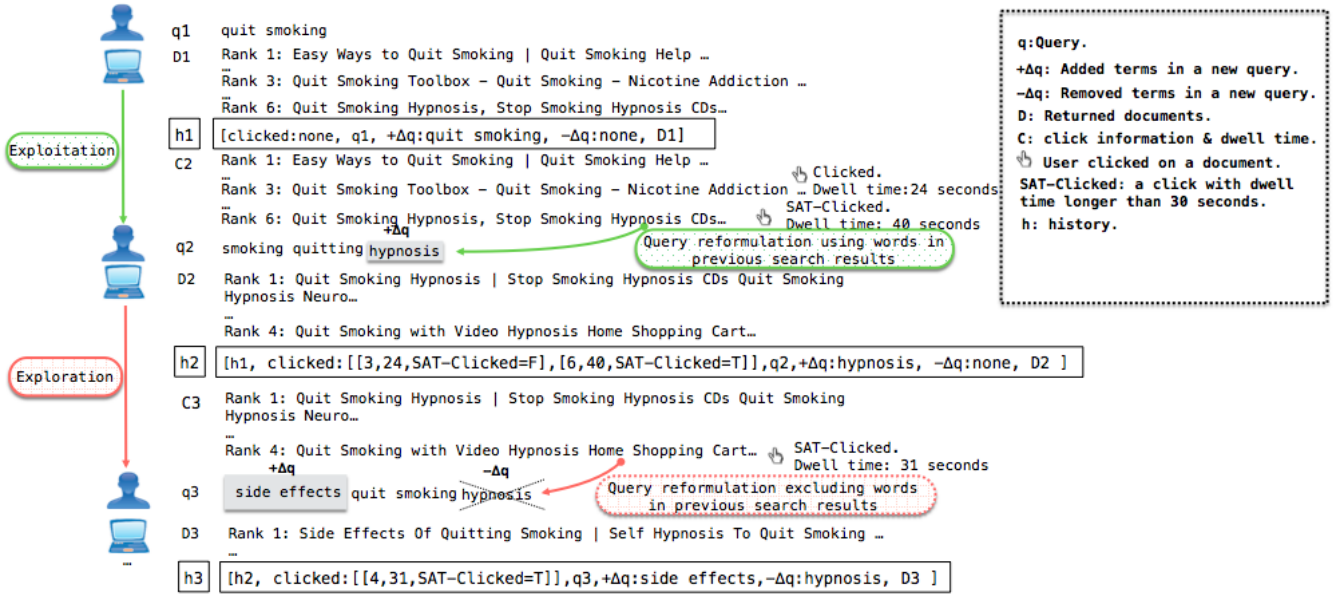

Figure 1: Example History in Session Search (TREC 2014 Session 1011).

documents ranked at positions 3 and 6. On the 3^{rd} document d_3, a dwell time of 24 seconds was spent and on the 6^{th}, 40 seconds. d_6 is considered as a satisfactory click (SAT-Click) since its dwell time is longer than 30 seconds, which is a simple rule that we use to identify SAT-Clicks [6]. The user then entered the next query, q_2 (*quit smoking hypnosis*).

The changes between the new query q_2 and the previous query q_1, termed as *query changes*, which suggest the users' judgment on document relevance and their degrees of desire to explore. Adopting the definitions in [9], we define the query changes as the syntactic editing changes between two consecutive queries.[3] They include positive query change ($+\Delta q$), i.e., the added terms in the new query, and negative query change ($-\Delta q$), i.e., the removed terms from the old query. Here we obtain $+\Delta q_2$ as *hypnosis* and $-\Delta q_2$ as none.

We also notice that the added query terms (in this case *hypnosis*) can be found in the SAT-Clicked document d_6; which suggests that the user could be inspired by reading the documents retrieved at the previous iteration and issued a new query to find more about it. Since the query reformulation used words in the previously retrieved documents, the transition from the first to the second iteration is identified as an "exploitation" – the user stays at the same subtopic. The search engine then returned documents D_2 for q_2. The history up to iteration 2 is h_2, which includes the earlier history h_1 and the new updates in this iteration.

At the third search iteration, the history h_3 also records the earlier histories, clicked documents (the 4^{th} ranked document is SAT-Clicked), the new query q_3 (*side effects quit smoking*), the query changes (added terms $+\Delta q_3$=*"side effects"* and removed terms $-\Delta q_3$=*"hypnosis"*), and the new documents being returned D_3. The positive query changes from q_3 to q_2 are not found in the SAT-Clicked document; instead, the term *"hypnosis"*, which is from the previously returned documents, is removed from q_3. Because the query reformulation excludes words from the previous search results, we consider this transition as an "exploration" – the user would like to shift to a new subtopic.

4.2 Three phases

Within a search iteration, we can see that the complexity for session search is still high – there involve queries, documents, clicks and query changes, among which many dependencies exist. However, we also notice that clicks, query, and retrieval in fact happen at different phases. Finer decomposition of the history beyond search iterations is thus possible. Based on such a decomposition, we can obtain a generative model which represents how a session is produced by both the user and the search engine. It will be greatly helpful to explore the interactions between the two agents. In this section, we present a influence diagram which describes a generative model with further decomposition of the history.

Figure 2 depicts the influence diagram of the observations, actions, and states in session search. The left most and the right most parts of the graph exhibit the states and their transitions, where the states are search iterations and indexed by t. Within an iteration, there are three phases: *browse* phase, *query* phase, and *rank* phase. They are separated by vertical lines in the figure. The upper part of the figure shows the phases that involves the user, and the lower part the search engine. At each phase, an agent performs the action and after that, an observation is made by the other agent. For instance, a_{rank} at time t (the search engine's ranking algorithm) yields o_{rank} at time $t+1$ (the document retrieved by the ranking algorithm). The observations will be used at later phases to make decisions for action selection, for instance, o_{rank} influences how a_{browse} should be selected. The observation in the current phase and earlier phases could be stored in an internal states n, which is not a real state that transitions, but a unit to memorize the observations and actions so far. Here n_1, n_2, n_3 are internal

[3]This definition of query reformulation patterns omit fine-grained classifications. However, it works effectively in session search, as reported by [9].

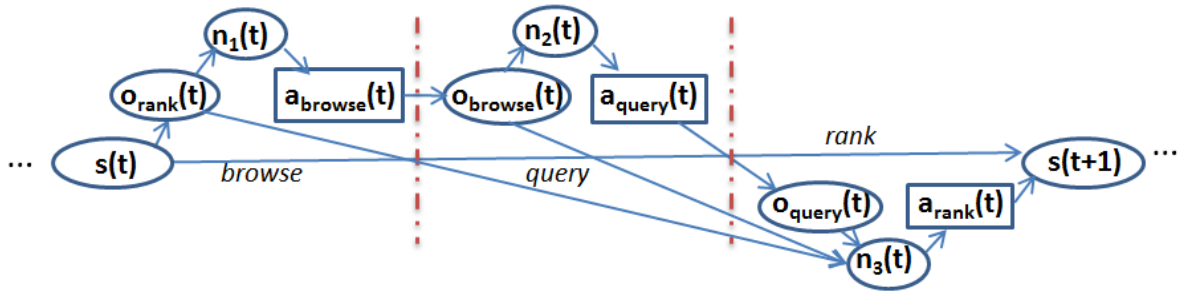

Figure 2: Influence Diagram for Session Search.

states, where 1 indicates browse, 2 query, and 3 rank.

Below are details about the three phases we propose here.

- *Phase 1: Browse.* The main actor in this phase is the user. The browse phase is after the search results are shown to the user and before the user starts to write the next query. The *browse* phase records how the user perceives and examines the (previously retrieved) search results. From a search engine's perspective, the emphasis is on getting users' feedback on these search results. Most feedback in this phase is implicit, including dwell time T and clicks C. These feedbacks and the derived feedback, such as SAT Clicks [18], are quite reliable relevance judgments from the user and hence can be used to calculate the reward function $r(t, h)$.

  Symbols that are used later in this paper and related to the browse phase are:

  - $o_{rank}(t)$ They are the observations of what the search engine's actions yield. For the previously returned documents at time $t-1$, we denote them as D_{t-1}. If the current iteration t is the only search iteration in context, we omit t from now on, for most symbols.
  - a_{browse} are the actions that the user takes to browse D_{t-1}. The actions include clicking and reading documents. The selection of the browsing actions depends on query q_t and retrieved documents D_{t-1}.
  - n_1 is an internal state at the user side. It memorizes o_{rank}.

- *Phase 2: Query.* The main actor in this phase is still the user.

  The phase happens when the user writes a new query. We assume that the query is created based on what has been seen in the browse phase, i.e., the previously retrieved documents, and the user's information need about the entire session. The new query triggers another iteration of document retrieval.

  Symbols related to the query phase are:

  - o_{browse} is the observations about how the user behaves in the browse phase. It includes the click information C_t and dwell time T_t for the previously retrieved documents. Note they are for D_{t-1} the documents retrieved at the previous iteration.
  - a_{query} is q_t, the query that the user writes at the current iteration t.
  - n_2 is another internal state at the user side. It records o_{browse} and helps the user obtain hints from documents being clicked or documents being read for a long time to write the new query q_t.

- *Phase 3: Rank.* The main actor in this phase is the search engine. The rank phase happens after the query is entered and before the search results are returned. The search engine delivers a ranked list of documents based on the current query and other earlier signals being collected. It is the search engine's ranking algorithm and the focus of this paper. The retrieved documents will be examined at the beginning – i.e., the browse phase – of the next iteration. The loop continues until the search stopped by the user. In the first iteration, there is no browse phase.

  Symbols related to the rank phase are:

  - o_{query} is what the search engine observes from what the user wrote as a query. It includes the query q_t and the query changes from q_t to q_{t-1}. The query changes contain two parts, the added terms $+\Delta q_t$ and the removed terms $-\Delta q_t$.
  - a_{rank} is the search engine action to retrieve a ranked list of documents. We define it as the action to put a document d at the top of a ranked list. It depends on a combination of query change Δq_t, previously retrieved documents D_{t-1}, and user clicks C_t and dwell time T_t for D_{t-1}.
  - n_3 is an internal state at the search engine side. It memorizes o_{rank}, o_{browse} and o_{query}, to allow the search engine generates optimal actions based on all three types of observations.

The three-phase design of the search history help produce a generative model of how the user and the search engine work together in a session. It has great potentials to study the interactions between the user and the search engine. However, in this paper, we focus on creating better search algorithms, therefore only utilizing the search engine part of the generative model.

5. SESSION SEARCH POLICY LEARNING

In this section, we describe how to learn a policy from the history, formed by observations and actions, by gradient descent.

According to the influence diagram in Figure 2, we write

the history distribution as

$$P(h|\theta) = \prod_{t=1}^{len(h)} P(o_{rank}(t), a_{browse}(t), o_{browse}(t), a_{query}(t), o_{query}(t), a_{rank}(t)|h_{t-1}, \theta) \quad (3)$$

where $len(h)$ is the length of history h and t indexes the search iterations.

By the chain rule, each factor in the history can be further factorized. In addition, by applying the Markov property and dropping the constants, $P(h|\theta)$ can be reduced into three factors:

$$\begin{aligned} P(h|\theta) \propto & \prod_{t=1}^{len(h)} P(a_{browse}(t)|o_{rank}(t), \theta_1) \\ & \times P(a_{query}(t)|o_{browse}(t), \theta_2) \\ & \times P(a_{rank}(t)|o_{browse}(t), o_{query}(t), o_{rank}(t), \theta_3) \end{aligned} \quad (4)$$

After simplifying Eq. 4 with index i to indicate the phases (1 indicates the browse phase, 2 for query, and 3 for rank):

$$P(h|\theta) \propto \prod_{t=1}^{len(h)} \prod_{i \in \{1,2,3\}} P(a^i(t)|n_i(t), \theta_i) \quad (5)$$

where θ_1, θ_2 and θ_3 are sub-vectors of θ at each phase. n_1, n_2, n_3 are internal states in the POMDP that memorize the observations and actions so far at time t. n_1 stores o_{rank}, n_2 stores o_{browse}, and n_3 stores o_{rank},o_{browse}, and o_{query}.

The actions of an agent will be selected from a range of options. The best actions are expected to be discovered through optimization. To force gradient descent to obey the simplex constraints, we assume that the actions follow the Softmax function [28]. Specifically, for an action in the i^{th} phase in a search iteration, we define the action selection distribution as

$$P(a^i|n_i, \theta_i) = \frac{e^{\theta_i \cdot \phi(a^i, n_i)}}{\sum_{a'^i} e^{\theta_i \cdot \phi(a'^i, n_i)}} \quad (6)$$

where ϕ is the feature vector and θ is the parameter vector. Section 5.2 shows feature examples used in this work.

To find the optimal action, that is to obtain the optimal model parameters for θ_1, θ_2 and θ_3, we apply gradient descent [4] on the long term reward function, i.e., the value function. If we take the gradient on the value function, we get:

$$\begin{aligned} \frac{\partial V_\theta(s_0)}{\partial \theta_k} &= \sum_{t=1}^{\infty} \gamma^t \sum_{h \in H} r(t,h) \frac{\partial P(h|\theta)}{\partial \theta_k} \\ &= \sum_{t=1}^{\infty} \gamma^t \sum_{h \in H} r(t,h) P(h|\theta) \\ &\quad \times \sum_{i=0}^{t} \frac{\partial ln[P(a_{browse}|n_1, \theta_1) P(a_{query}|n_2, \theta_2) P(a_{rank}|n_3, \theta_3)]}{\partial \theta_k} \end{aligned} \quad (7)$$

where H is the set of all the possible histories and for phase i, the gradient is:

$$\frac{\partial ln P(a^i|n_i, \theta_i)}{\partial \theta_i} = \phi(a^i, n_i) - \sum_{a'} \phi(a'^i, n_i) P(a'^i|n_i, \theta_i) \quad (8)$$

Actions at different phases in a search iteration can all be modeled in a form similar to Eq. 6. In this paper, we focus on finding the optimal actions for the search engine. In the rank phase, the search engine agent needs abundant information from the history. The history information, including o_{rank}, o_{query}, and o_{browse}, is stored in n_3. The ranking actions a_{rank} are selected based on:

$$P(a_{rank}|n_3, \theta_3) = \frac{e^{\theta_3 \cdot \phi(a_{rank}, n_3)}}{\sum_{a'_{rank}} e^{\theta_3 \cdot \phi(a'_{rank}, n_3)}} \quad (9)$$

where θ_3 is a vector of parameters relating to the policy at the rank phase.

Specifically, for the rank phase, the gradient is:

$$\begin{aligned} \frac{\partial ln P(a_{rank}|n_3, \theta_3)}{\partial \theta_3} =& \phi(a_{rank}, n_3) - \\ & \sum_{a'_{rank}} \left[\phi(a'_{rank}, n_3) \; P(a'_{rank}|n_3, \theta_3) \right] \end{aligned} \quad (10)$$

By solving the gradient, we obtain a formula to update θ_3 at each step:

$$\begin{aligned} \Delta\theta_3 = \sum_{h \in H} \sum_{t=1}^{len(h)} \gamma^t r(t,h) \times \sum_{i=1}^{t} [& \phi(a_{rank}, n_3) - \\ & \sum_{a'_{rank}} \phi(a'_{rank}, n_3) \; P(a'_{rank}|n_3, \theta_3)] \end{aligned} \quad (11)$$

where H is a set of histories.

The remaining of this section explains the gradient descent policy learning algorithm, features, document ranking function, and the reward function $r(t,h)$.

5.1 Algorithm

Algorithm 1 outlines the direct policy learning algorithm for session search.

The algorithm takes inputs from a set of histories H and a limiting threshold ϵ for gradient descent. The model parameter θ_3 is randomly initialized between 0 and 1.

The algorithm then enters a loop to examine all the histories and to learn the model parameter θ_3. Inside the loop, a history h is sampled from the set. We then examine each search iteration in h and obtain values for the observations o_{rank} and o_{browse}, which are about the previously retrieved documents D_{t-1}, and clicks and dwell time for it at time t, C_t and T_t. It also calculates the query changes Δq_t between the queries and stores it in observation o_{query}.

The search engine then calculates a document ranking score, which is $P(a_{rank}|n_3, \theta_3)$, for all the documents based on the current model parameter θ_3, and the features. A new ranking of the documents are produced based on $P(a_{rank}|n_3, \theta_3)$, which is calculated by Eq. 9. The new rankings are used to calculate the gradient updates $\Delta\theta_3$.

The loop stops when the gradient update $\Delta\theta_3$ stops changing much or we have examined all the training data.

5.2 Features

The features used in our algorithm are $\phi(a_{rank}, n_3)$. They are descriptors of observations o_{browse}, o_{query}, and o_{rank}, to show the interactions among the queries, documents, query changes, and clicks. We use 112 features in this work.

Algorithm 1 Direct Policy Learning for Session Search.

procedure DPL(H, ϵ)
  ▷ H is the training history set. ϵ is a threshold.
  $\theta_3 \leftarrow random(0, 1)$

  repeat
    Sample history h from H:
    $q_0, D_0, C_0, T_0 \leftarrow \emptyset, r(1, h) \leftarrow 0, \Delta\theta_3 \leftarrow 0$

    for $t = 1$ to $len(h)$ **do**
      $o_{rank} \leftarrow D_{t-1}$, $n_1 \leftarrow o_{rank}$

      ▷ a_{browse} is performed by the user
      $o_{browse} \leftarrow (C_t, T_t)$, $n_2 \leftarrow o_{browse}$

      $r(t, h) \leftarrow CalculateReward(D'_t, o_{browse}, h)$

      ▷ a_{query} is performed by the user
      $o_{query} \leftarrow$ GetQueryChange(q_t, q_{t-1})
      $n_3 \leftarrow (o_{rank}, o_{browse}, o_{query})$

      Sample a search engine action a_{rank}
        $\sim P(a_{rank}|n_3, \theta_3)$
      $D'_t \leftarrow DocRanking(a_{rank})$

      $\Delta\theta_3 \leftarrow UpdateGradient(r(t, h), D'_t, n_3, \theta_3)$
      $\theta_3 \leftarrow \theta_3 + \Delta\theta_3$
    end for
  until $\Delta\theta_3 < \epsilon$ or H is running out

Table 2: Example Features.

Notations
w is a term
t is the current search iteration
q_t is the current query
q_{t-1} is the previous query
$+\Delta q_t$ is the added terms in q_t, comparing to q_{t-1}
$-\Delta q_t$ is the removed terms in q_t, comparing to q_{t-1}
D_{t-1} is the previously retrieved documents/snippets
Query Features
Test if search term $w \in q_t$ and $w \in q_{t-1}$
of times that a term w occurs in $q_1, q_2, \ldots, q_t$
Document Features
the length of the content of a document
the length of the snippet of a document
Query-Document Features
Test if a search term $w \in +\Delta q_t$ and $w \in D_{t-1}$
Test if a document d contains a term $w \in -\Delta q_t$
$tfidf$ score of a document d to q_t
Click Features
Test if there are SAT-Clicks in D_{t-1}
of times a document being clicked in the current session
of seconds a document being viewed and reviewed in the current session
Query-Document-Click Features
Test if q_i leads to SAT-Clicks in D_i, where $i = 0 \ldots t-1$
Session Features
the length of the current session
position at the current session

The features can be classified into a few groups. Query features, document features, and session features are descriptors of the queries, the documents and the session, respectively. Query-document features are statistics about the interaction of the query terms and the previously retrieved documents. Click features are about the clicking information and the dwell time. The most exciting ones are the query-document-click features. They are about the interactions of all three elements. Table 2 lists some example features. It is worthy noting that out framework is able to take into account as many features as possible.

5.3 Document Ranking

It is worthy noting that $P(a_{rank}|n_3, \theta_3)$ is the scoring function for a document d. We use $a_{rank} = d$ to denote that the search engine will take an action to generate a document list with document d at the top.

$P(a_{rank}|n_3, \theta_3)$ originally presents the probability of selecting a particular action. In the context of session search, we use it to present the probability $P(a_{rank} = d|n_3, \theta_3)$ that selecting d to be put at the top of a ranked list under n_3 and θ_3 at the t^{th} search iteration. A ranked list of document is generated by sorting through the probabilities.

5.4 Calculating the Reward

The reward function $r(t, h)$ is to estimate the benefits from an action, specifically a_{rank}. A reward function can guide the search engine throughout the entire dynamic process of session search.

Since session search is a document retrieval task, it's natural that *the reward function is about document relevance*. Our reward function $r(t, h)$ uses the a weighted average of clicks and SAT Clicks as the reward. First, we define a strong SAT-Click as a click with dwell time longer than 30 seconds, and a weak SAT-Click as a click with dwell time between 10 and 30 seconds. We calculate the reward as

$$r(t, h) = \sum_{d_i \in D_t} \frac{\text{normalized } gain(d_i, t, h)}{log_2(rank_{d_i} + 1)} \quad (12)$$

where $rank_{d_i}$ is document d_i's rank position in D_t, and $gain$ is what a document contributes up to time t in history h:

$$gain(d, t, h) = a \times Count(\text{Strong SAT-Click}) + b \times Count(\text{Weak SAT-Click}) \quad (13)$$

where $Count()$ counts the occurrences of the types of clicks in h_t. The weights are normalized by the sum of such clicks in h. Experimentally we set $a = 1$ and $b = 0.5$. This reward function encourages to rank the strong SAT-Clicked documents high in a document list.

Rewards can also be directly generated from user's relevance assessments when the assessments are available [9, 22]. For instance, the reward function can be nDCG [12] with ground truth. However, in real time interactions, we will not have the ground truth before hand and therefore could not use this reward. In the experiments, we used it as an ideal reward to produce upper bound runs.

6. EXPERIMENTS

We evaluate the proposed approach on the recent TREC Session Tracks. The TREC Session Track [15, 16] has run since 2010. It is an annual evaluation for session search systems conducted at the National Institute of Standards and Technology (NIST). The task of TREC Session Tracks is to

Table 3: TREC Session Dataset Statistics.

	TREC'12	TREC'13	TREC'14
#Sessions	98	87	1021
#Queries	297	442	4226
Avg. session length	3.03	5.08	4.14
Max session length	11	21	16
#Sessions w/ length <4	75	38	466
#Sessions w/ length 4∼10	22	42	543
#Sessions w/ length >10	1	7	12

retrieve a list of 2,000 documents for the last query in a session to satisfy the information need for the entire session. For each session, queries, the top 10 retrieved documents for each query, and interaction information in the current session are provided. The participating systems are not allowed to use the search topics – i.e., the search engine does not know the information need ahead of time.

We evaluate our algorithm on the TREC Session 2012, 2013 and 2014 data. TREC 2010 and TREC 2011 are earlier years for this Track and the tasks are less relevant. Note that for TREC 2013, many relevant documents are not included by pooling from very few submitted runs. Hence, the ground truth data is not complete. The tasks in our experiments are equivalent to TREC 2012 RL4, TREC 2013 RL2, and TREC 2014 RL2.

6.1 Dataset

The document collection used in this evaluation is ClueWeb. For TREC 2012 Session, we use ClueWeb09 CatB, which contains 50 million English webpages crawled in 2009. For TREC 2013 and 2014 Session, we use ClueWeb12 CatB, which contains 50 million English webpages crawled in 2012. The spam documents and duplicated documents are removed. We use the TREC 2012 data for training when testing on 2013 and 2014 data, and use the 2014 data for training when testing on 2012 data.

The sessions and queries were provided to the participating systems in a query log. When the log was created, users were given search topics containing a complex information need. The information need usually contains multiple subtopics. For examples, Table 1 lists three examples TREC sessions. The user then wrote a set of queries and interacted with a search engine to find relevant documents for the search topic. All user interactions and clicking information, urls, snippets, and returned document identification numbers for each query, were recorded in the log.

TREC 2012 has 98 sessions with an average session length of 4 queries. TREC 2013 has 87 sessions with an average length of 5. TREC 2014 has 1021 sessions with an average length of 4. Table 3 lists the dataset statistics.

6.2 Evaluation Metrics

The main evaluation criterion in our experiments is the whole-session search accuracy, which is the official TREC evaluation metric. It evaluates the nDCG@10 [12] for the retrieval results for the last query but the ground truth was created based on document relevance to the entire session. The relevance judgments consists of six relevance grades: −2 for spams, 0 for non-relevant, 1 for relevant, 2 for highly relevant, 3 for key pages, and 4 for navigational pages (a homepage that exactly matches the search topic) [15].

Table 4: Whole-Session Search Accuracy on TREC 2012 Session († indicates a stat. significant improvement over the baseline (p<0.05, t-test, one-sided))

	nDCG@10	σ^2	MAP	σ^2
TREC median	0.2314	0.0575	0.1221	0.0152
TREC best(baseline)	0.3153	0.0844	0.1588	0.0191
lemur	0.2622	0.0800	0.1342	0.0160
winwin-short	0.2658	0.0785	0.1552	0.0195
qcm	0.3368†	0.0948	0.1536	0.0196
dpl	0.3382†	0.0932	0.1544	0.0193
winwin-long	0.3631†	0.0847	0.1651	0.0198
dpl-upper	0.3643†	0.0944	0.1609	0.0197

Table 5: Whole-Session Search Accuracy on TREC 2013 Session († indicates a stat. significant improvement over the baseline (p<0.05, t-test, one-sided))

	nDCG@10	σ^2	MAP	σ^2
TREC median	0.1504	0.0245	0.0506	0.0029
TREC best(baseline)	0.1706	0.0401	0.1008	0.0096
lemur	0.1043	0.0184	0.0398	0.0023
winwin-short	0.1253	0.0299	0.0334	0.0023
qcm	0.1555	0.0305	0.0568	0.0037
dpl	0.1604	0.0319	0.0600	0.0039
dpl-upper	0.2093†	0.0467	0.0734	0.0071
winwin-long	0.2137†	0.0396	0.0632	0.0037

6.3 Systems to Compare

We compare the following session search systems in the experiments. 1) **lemur**, which is the state-of-the-art academic search engine created by UMass and CMU. We use Lemur's multinomial language modeling and Dirichlet smoothing with the smoothing parameter μ set to 5000. The last query of each session is fed to Lemur to start the search. 2) 2) **qcm**, the query change mode, which is a state-of-the-art session search system proposed by Guan et. al. [9]. It is developed based on Markov Decision Processes and focuses on adjusting term weights based on user query reformulation patterns in a session. The term re-weighting parameters used here are theme terms $\alpha = 2.2$, added old terms $\beta = 1.8$, added new terns $\epsilon = 0.07$, and removed terms $\theta = 0.4$ (please see [9] for parameter meanings). 3) **winwin**, a session search framework proposed by Luo et. al. in [22]. This system models session search as POMDPs with four hidden decision-making states. It is like a meta-search-engine with an action space consisting many alternative retrieval algorithms, including language model, BM25, pseudo-relevance feedback, query expansion, and qcm. We implemented two version of winwin, **winwin-short** and **winwin-long**. Winwin-short uses immediate feedback, i.e. the user clicks, as a short-term feedback. Winwin-long uses an ideal reward function generated from the ground truth, which gives an upper bound for winwin. 5) **dpl**, the direct policy learning framework proposed here. 6) **dpl+upper**, the proposed framework using ground truth as the reward function (the same as in winwin-long), which gives us an upper bound performance of the algorithm.

6.4 Results

In this experiment, we conduct the official evaluation in TREC Session Tracks. We report the nDCG@10, which is

Table 6: Whole-Session Search Accuracy on TREC 2014 Session Track († indicates a stat. significant improvement over the baseline (p<0.05, t-test, one-sided))

	nDCG@10	σ^2	MAP	σ^2
TREC median	0.1876	0.0383	0.3750	0.1125
TREC best(baseline)	0.2482	0.0471	0.4270	0.1266
lemur	0.2169	0.0470	0.3860	0.1162
winwin-short	0.2241	0.0420	0.3540	0.0875
qcm	0.2422	0.0451	0.4140	0.1236
dpl	$0.2608^{\dagger}$	0.0435	0.4390	0.1182
winwin-long	$0.3076^{\dagger}$	0.0549	$0.4800^{\dagger}$	0.1158
dpl-upper	$0.3266^{\dagger}$	0.0606	0.4320	0.1208

Table 7: Efficiency on TREC 2012 Session Track.

	TREC 2012		
Approach	Wall Clock	CPU Cycle	Speed Up
winwin-long	2.5×10^4s	6.8×10^{13}	1.00
winwin-short	2.5×10^4s	6.8×10^{13}	1.00
qcm	5.6×10^3s	1.5×10^{13}	4.46
dpl-upper	3.2×10^3s	8.7×10^{12}	7.81
dpl	3.1×10^3s	8.4×10^{12}	8.06
lemur	2.6×10^3s	7.0×10^{12}	9.62

the official TREC Session evaluation metric, as well as MAP averaged over all the sessions and their standard derivations.

Tables 4, 5 and 6 report the whole-session search accuracy. Here we also report the results from the TREC median and TREC best runs. We use TREC best as a strong baseline here. They are the best performing system actually submitted to the evaluation. All the statistical significance tests are performed against this baseline.

From Tables 4, 5 and 6, we are happy to see that dpl shows the highest nDCG@10 score among the systems that use clicks as the reward function. In TREC 2012, dpl achieves a 7.3% improvement of nDCG@10 over the TREC best and in TREC 2014, a 5% improvement of nDCG@10 over the TREC best. Both improvements are statistically significant ($p < 0.05$, one-sided t-test). Moreover, dpl shows a 2.8% MAP improvement over the TREC best. More exciting, dpl is 7.6% statistically significant better than the state-of-the-art session search system qcm in nDCG@10 in TREC 2014.

Sometimes a run that has high nDCG@10 score may have a low MAP score, for instance winwin-short in TREC 2014. It is because that some models focus on achieving high precision in the top 10 retrieval documents, while their MAP over all 2000 returned documents could be low.

We observe that winwin-long and dpl-upper produce impressive results. It is not surprising because that they both use an ideal reward function nDCG@10 calculated using the ground truth data – winwin-long uses this idea reward function to pick the best retrieval algorithms from the action space, while dpl-upper uses the true document relevance grades as a feature to pick the most relevant documents from all retrieved ones. These two runs are supposed to outperform all the rest runs since it uses ground truth as the rewards, to pick an optimal action from the action sets. Note that during the process of session search, a search engine won't be able to know the ground truth data. We therefore use them to inform us an upper bound of winwin and dpl, respectively.

Table 8: Efficiency on TREC 2013 Session Track.

	TREC 2013		
Approach	Wall Clock	CPU Cycle	Speed Up
winwin-long	9.4×10^3s	2.5×10^{13}	1.00
winwin-short	8.4×10^3s	2.3×10^{13}	1.12
dpl	2.3×10^3s	6.2×10^{12}	4.09
qcm	1.5×10^3s	3.9×10^{12}	6.27
dpl-upper	1.4×10^3s	3.8×10^{12}	6.71
lemur	3.3×10^2s	9.0×10^{11}	28.48

Table 9: Efficiency on TREC 2014 Session Track.

	TREC 2014		
Approach	Wall Clock	CPU Cycle	Speed Up
winwin-short	6.0×10^3s	1.6×10^{13}	1.00
winwin-long	5.3×10^3s	1.4×10^{13}	1.13
qcm	1.2×10^3s	3.4×10^{12}	5.00
dpl	6.4×10^2s	1.7×10^{12}	9.38
dpl-upper	6.2×10^2s	1.7×10^{12}	9.68
lemur	2.0×10^2s	5.4×10^{11}	30.00

6.5 Efficiency

One motivation of this paper is to design a more efficient session search algorithm. We therefore conduct the efficiency tests for the algorithms under evaluation.

Tables 7, 8 and 9 compare the algorithms' retrieval efficiency over TREC datasets. We report the actual wall clock running time to finish the entire set of experiments in that year, cpu cycles, the speed up to (how many times faster than) winwin-long in the tables. The systems are sorted by wall clock time in decreasing order in the tables. The shorter the time, the faster the system. A hardware support of 4 CPU cores (2.70GHz), 32GB Memory and 22 TB NAS are used in this set of experiments.

Among the approaches being evaluated, winwin-short and winwin-long are the slowest. Qcm, dpl and dpl-upper are 5 $\sim$ 10 times faster than the winwin approaches. dpl and dpl-upper only take as half time as qcm on TREC 2012 and TREC 2014. They are pretty fast. Lemur appears to use the least amount of time, however it is not a session search algorithm but actually an ad-hoc approach, which only deals with the current (last) query. We can conclude that dpl is a much more efficient session search algorithm.

7. CONCLUSION

This paper presents a novel document retrieval algorithm for session search. From a new perspective based on reinforcement learning, we produce a long term optimized ranking function for documents. The direct policy learning reduces model complexity and enable us to achieve a more efficient and less complex model. In addition, we are able to flexibly incorporate a wide range of features to describe the rich interactions in session search. Due to the limitation of the setting of the TREC Session Tracks, where a real user and search engine interaction is missing, we could not actually fully apply our modeling to handle both the user side

and the search engine side, however, the modeling is general enough to allow future practice of it in experiments with real time interactions. Nonetheless, the framework is shown to be highly effective by evaluations conducted on three years of TREC Session Tracks. It statistically significantly outperforms the TREC best runs in 2012 and 2014.

Due to its trial-and-error nature, session search is an excellent application for reinforcement learning (RL). As part of the efforts to bring RL to IR, this paper makes a contribution in studying direct policy learning for session search. We hope this paper encourages more future work in dynamic search for high complexity tasks.

8. ACKNOWLEDGMENT

The research is supported by DARPA FA8750-14-2-0226, NSF IIS-1453721, and NSF CNS-1223825. Any opinions, findings, conclusions, or recommendations expressed in this paper are of the authors, and do not necessarily reflect those of the sponsor.

References

[1] M. Ageev, Q. Guo, D. Lagun, and E. Agichtein. Find it if you can: A game for modeling different types of web search success using interaction data. In *SIGIR '11*.

[2] L. Baird and A. Moore. Gradient descent for general reinforcement learning. In *Advances in neural information processing systems 11*. 1999.

[3] P. Boldi, F. Bonchi, C. Castillo, D. Donato, A. Gionis, and S. Vigna. The query-flow graph: Model and applications. In *CIKM '08*.

[4] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[5] S. R. K. Branavan, H. Chen, L. S. Zettlemoyer, and R. Barzilay. Reinforcement learning for mapping instructions to actions. In *ACL '09*.

[6] K. Collins-Thompson, P. N. Bennett, R. W. White, S. de la Chica, and D. Sontag. Personalizing web search results by reading level. In *CIKM '11*.

[7] C. Eickhoff, J. Teevan, R. White, and S. Dumais. Lessons from the journey: A query log analysis of within-session learning. In *WSDM '14*.

[8] S. Fox, K. Karnawat, M. Mydland, S. Dumais, and T. White. Evaluating implicit measures to improve web search. *ACM Trans. Inf. Syst.*, 23(2):147–168, 2005.

[9] D. Guan, S. Zhang, and H. Yang. Utilizing query change for session search. In *SIGIR '13*.

[10] A. Hassan, R. Jones, and K. L. Klinkner. Beyond dcg: User behavior as a predictor of a successful search. In *WSDM '10*.

[11] K. Hofmann, S. Whiteson, and M. de Rijke. Balancing exploration and exploitation in learning to rank online. In *ECIR'11*.

[12] K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Trans. Inf. Syst.*, 20(4):422–446, 2002.

[13] J. Jiang, D. He, and S. Han. Pitt at trec 2012 session track. In *TREC '12*.

[14] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *J Artificial Intelligence Res.*, 4:237–285, 1996.

[15] E. Kanoulas, B. Carterette, M. Hall, P. Clough, and M. Sanderson. Overview of the trec 2013 session track. In *TREC'13*.

[16] E. Kanoulas, B. Carterette, M. Hall, P. Clough, and M. Sanderson. Overview of the trec 2014 session track. In *TREC'14*.

[17] J. Y. Kim, M. Cramer, J. Teevan, and D. Lagun. Understanding how people interact with web search results that change in real-time using implicit feedback. In *CIKM '13*.

[18] Y. Kim, A. Hassan, R. W. White, and I. Zitouni. Modeling dwell time to predict click-level satisfaction. In *WSDM '14*.

[19] A. Kotov, P. N. Bennett, R. W. White, S. T. Dumais, and J. Teevan. Modeling and analysis of cross-session search tasks. In *SIGIR '11*.

[20] J. Liu and N. J. Belkin. Personalizing information retrieval for multi-session tasks: The roles of task stage and task type. In *SIGIR '10*.

[21] C. Lucchese, S. Orlando, R. Perego, F. Silvestri, and G. Tolomei. Identifying task-based sessions in search engine query logs. In *WSDM '11*.

[22] J. Luo, S. Zhang, and H. Yang. Win-win search: Dual-agent stochastic game in session search. In *SIGIR '14*.

[23] P. Masurel and K. Lefevre-Hasegawa. Dataiku's solution to yandex's personalized web search challenge. In *WSCD '14*.

[24] K. Raman, P. N. Bennett, and K. Collins-Thompson. Toward whole-session relevance: Exploring intrinsic diversity in web search. In *SIGIR '13*.

[25] J. Schmidhuber. Sequential decision making based on direct search. In R. Sun and C. L. Giles, editors, *Sequence Learning: Paradigms, Algorithms, and Applications*. Springer, 2001.

[26] X. Shen, B. Tan, and C. Zhai. Implicit user modeling for personalized search. In *CIKM '05*.

[27] Y. Song and L.-w. He. Optimal rare query suggestion with implicit user feedback. In *WWW '10*.

[28] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[29] R. W. White, I. Ruthven, J. M. Jose, and C. J. V. Rijsbergen. Evaluating implicit feedback models using searcher simulations. *ACM Trans. Inf. Syst.*, 23(3):325–361, 2005.