Generating Labels from Clicks

R. Agrawal A. Halverson K. Kenthapadi N. Mishra P. Tsaparas
Search Labs, Microsoft Research
{rakesha,alanhal,krisken,ninam,panats}@microsoft.com

ABSTRACT

The ranking function used by search engines to order results is learned from labeled training data. Each training point is a (query, URL) pair that is labeled by a human judge who assigns a score of Perfect, Excellent, etc., depending on how well the URL matches the query. In this paper, we study whether clicks can be used to automatically generate good labels. Intuitively, documents that are clicked (resp., skipped) in aggregate can indicate relevance (resp., lack of relevance). We give a novel way of transforming clicks into weighted, directed graphs inspired by eye-tracking studies and then devise an objective function for finding cuts in these graphs that induce a good labeling. In its full generality, the problem is NP-hard, but we show that, in the case of two labels, an optimum labeling can be found in linear time. For the more general case, we propose heuristic solutions. Experiments on real click logs show that click-based labels align with the opinion of a panel of judges, especially as the consensus of the panel grows stronger.

Categories and Subject Descriptors

H.3.3 [**Information Retrieval**]: Search; G.2.2 [**Discrete Math**]: Graph Algorithms

General Terms

Algorithms, Experimentation

Keywords

Generating Training Data, Graph Partitioning

1. INTRODUCTION

Search engines order web results via a ranking function that, given a query and a document, produces a score indicating how well the document matches the query. The ranking function is itself learned via a machine learning algorithm such as [15]. The input to the learning algorithm is typically a collection of (query, URL) pairs, labeled with relevance labels such as Perfect, Excellent, Good, Fair or Bad indicating how well the document matches the query. Obtaining labels of high quality is of critical importance: the quality of training data heavily influences the quality of the ranking function.

Currently, the labels for training data are collected using human judges. Typically, each (query, URL) pair is assigned to a single judge. However, this leads to error-prone judgments since it is very hard for a single judge to capture all the intents and nuances of a query posed by a user. To alleviate such errors, a panel of judges can be used to obtain multiple judgments for the same (query, URL) pair. The final label of the pair is then derived by aggregating the multiple judgments.

This manual way of obtaining labels is time-consuming, labor-intensive, and costly. Furthermore, as ranking models become more complex, the amount of labeled data needed to learn an accurate model increases [23]. Further, to ensure temporal relevance of labels, the labeling process must be repeated quite often. Consequently, there is a pressing need for search engines to automate the labeling process as much as possible.

It has been observed [20] that the click logs of a search engine can be used to automate the generation of training data. The click log records all queries posed to a search engine, the URLs that were shown to the user as a result of the query, and the clicks. The logs capture the preferences of the users: Clicked documents are most likely relevant to the needs and the intent of the user, while skipped (not clicked) documents are most likely not. Aggregation of the activities of many users provides a powerful signal about the quality of a (query, URL) pair. This data can thus be used for the task of automatically generating labels.

Joachims et al's seminal work [20, 22] proposes a collection of *preference rules* for interpreting click logs. These rules (e.g., clicked documents are better than preceding skipped documents) when applied to a click log produce pairwise preferences between the URLs of a query, which are then used as a training set for a learning algorithm.

Applying the ideas of Joachims et al to a search engine click log uncovers several shortcomings. First, the preference rules defined by Joachims et al assume a specific user browsing model that is overly simplified. As we outline in Section 3, the rules do not fully capture the aggregate behavior of users and also limits the generation of training data.

Second, the work tacitly assumes a relatively controlled environment with a stable search engine, i.e., one that always produces search results in the same order. It also assumes a small set of users that behave consistently. Each preference rule produces a consistent set of preferences between URLs. These assumptions are not satisfied in the environment of a real search engine: the observed behavior for a query changes dynamically over time, different orderings may be shown to different users, and the same users may exhibit different behavior depending on the time of day. When applying preference rules on click logs, many contradicting pairwise preferences must be reconciled to obtain a consistent labeling.

Third, the goal is to generate pairwise preferences which are then used directly for training. However, several learning algorithms operate on labeled training data [3, 28]. For such algorithms it is important to produce labels for the (query, URL) pairs.

Contributions: The main contribution of this paper is a method for automatically generating labels for (query, URL) pairs from click activity. The method directly address the three shortcomings mentioned above.

More concretely, we propose a new interpretation of the click log that utilizes the probability a user has seen a specific URL to infer expected click/skip behavior. This new model more accurately captures the aggregate behavior of users. We model the collection of (often contradicting) pairwise preferences as a graph and formulate the label generation problem as a novel graph partitioning problem (MAXIMUM-ORDERED-PARTITION), where the goal is to partition nodes in the graph into labeled classes, such that we maximize the number of users that agree with the labeling minus the number of users that disagree with the labeling.

The MAXIMUM-ORDERED-PARTITION problem is of independent theoretical interest. We show that in the case of finding two labels, i.e., relevant and not relevant, it surprisingly turns out that the optimum labeling can be found in linear time. On the other hand, we show that the problem of finding the optimum n labeling, where n is the number of vertices in the graph, is NP-hard. We propose heuristics for addressing the problem of label generation for multiple labels. Our methods compute a linear ordering of the nodes in the graph, and then find the optimal partition using dynamic programming.

We conduct an extensive experimental study of both the preference rules and the labeling methods. The experiments compare click inferences to the aggregate view of a panel of judges. We demonstrate that our probabilistic interpretation of the click log is more accurate than prior deterministic interpretations via this panel. Further, we show that the stronger the consensus of the panel, the more likely our click labels are to agree with the consensus. In the event that click labels disagree with a strong consensus, we show that it can often be attributed to short queries where the intent is ambiguous.

2. RELATED WORK

The idea of using clicks as a basis for training data was first proposed by Joachims [20, 22]. We explain this work in more detail in the next section. Subsequently, a series of papers proposed more elaborate models for interpreting how users interact with a search engine. For instance, Radlinski and Joachims [25] stitch together preferences from multiple sessions. Craswell et al [6] propose a cascade model that is used to predict the probability a user will click on a result. The model assumes that users view search results from top to bottom, deciding at each position whether to click, or skip and move to the next result. Dupret and Piwowarski [10] also assume that users read from top to bottom, but are less likely to click the further the distance from their last click. These ideas could be used to extend the results in this paper. We leave this as a promising direction for future work.

Training on Pairs: The idea of sorting URLs by the number of clicks and then training on the induced ordered pairs was explored in [9]. Whereas skipped documents are ignored in that work, we explicitly utilize skips in making relevance assertions. Further, no correction is made for *presentation bias*, a user's decision to click on a result based on its position as opposed to its relevance. We overcome this problem to a certain extent by appealing to eye-tracking studies (Section 3).

In our work, we construct one directed, weighted graph per query. Given such a graph, a natural question is why not directly provide the edges as pairwise training data to a machine learning algorithm such as [4, 15, 20]? A few issues arise. First, the graphs we create are incomplete in the sense that edges (u, v) and (v, w) may be present in the graph while (u, w) may not be. In this scenario, we do want a machine learning algorithm to also train on (u, w), but the edge is not present. Second, the graphs we create contain cycles. For example, in the case of an ambiguous query, some users may prefer u to v, while others may prefer v to u. Providing both pairs $u > v$ and $v > u$ to a machine learning algorithm is not consistent. Finally, we cannot create training data of the form: u and v are equally relevant. Such "equally good" training data has proved to be useful in the context of ranking search results [30]. The labeling that we generate has the power to overcome these problems. Indeed, our experiments demonstrate that our ordered graph partitioning creates more training pairs that are more likely to agree with a panel of judges than the edges of the graph.

Rank Aggregation: The techniques that we use for generating a labeling from a graph involve first ordering the vertices by relevance, then partitioning this ordering into classes and then assigning labels to these classes. To construct an ordering of the vertices, a ranking is sought that minimizes the total number of flipped preferences, i.e., for all URLs u and v, the number of users that prefer u to v, but $v > u$ in the final ranking. Such a ranking is known as a *consensus ranking*. Rank aggregation methods [1, 2, 11, 12, 13] produce a consensus ranking based on either totally or partially ordered preferences. Ailon et al [2] show that given a tournament graph, i.e., one where for every pair of vertices u and v there is either an edge from u to v or v to u, a 2-approximate consensus ranking can be found under certain assumptions. Since our graph is not a tournament, such techniques do not apply. However, we do use other methods suggested in the literature. For example, we rank vertices based on a page-rank style random walk that was previously investigated in [11, 24]. Actually, since our final goal is to produce ordered labels, the work that most closely matches our problem is the bucket ordering work described in [16]. We elaborate further on this technique in the experiments.

Limitations of Clicks: Clicks are known to possess several limitations. Some of these limitations can be addressed by existing work on clicks, while others lie at the forefront of research. For instance, spurious clicks pose a problem in that a small number of bad clicks can trigger a cascade of incorrect inferences. We address this problem by making decisions based on only a large number of users. Presentation bias is another issue with clicks that we already mentioned. Another method for coping with presentation bias includes randomly altering the order in which results are presented [26, 27]. Clicks pose further problems that are under active investigation in other research communities. For instance, fraudulent clicks need to be detected and removed [18, 19]. The bot traffic that we are able to detect is removed from our click logs, but we know more exists. Also, clicks are more an indication of the quality of the caption than the quality of the page [5, 8, 29]. We believe that continued future work on these topics will address issues with clicks.

3. INTERPRETING CLICK LOGS

The click log of a search engine consists of a sequence of query *impressions*; each impression corresponds to a user posing a query to the search engine. Each impression may have one or more clicks on the document URLs returned in the result set. The ordered list of document URLs presented to the user, and the position of each document click is stored in the click log. The goal is to use this information to derive preferences between the URLs.

In this section we describe how to interpret the click logs for obtaining pairwise preferences between URLs. We begin by discussing the work of Joachims et al in detail, and its limitations when applied to a search engine click log. We then propose our own probabilistic model for the query log interpretation.

3.1 Prior Work on Interpreting Query Click Logs

Joachims et al [20, 22] proposed a set of preference rules for interpreting the click logs. These rules generate pairwise preferences between URLs and are based on eye-tracking studies. We group the rules proposed by Joachims into two categories, rules that reinforce the existing ranking (positive rules) and rules that contradict the existing ranking (negative rules).

Rules that Reinforce the Existing Order of Search Results: This category includes the rule "Click $>$ Skip Next" which states that if a person clicks on URL A at position i and skips URL B at position $i+1$ then URL A is preferable to URL B ($A > B$). This rule is based on eye-tracking studies: a user who clicked on a document at position i is likely to have also seen the URL at position $i+1$. While this may capture the most common user's behavior, it does not capture the aggregate behavior: some users may browse more URLs below position i. This is especially true when clicking at position 1 as there are users that do read below position 2. Furthermore, this rule generates sparse data. For each click on a URL we obtain only a single pairwise preference.

Rules that Contradict the Existing Order: This category includes the rules "Click $>$ Skip Above", "Click $>$ Skip Previous", "Last Click $>$ Skip Above", "Click > Click Above". These rules rely on the eye-tracking observation that if a user clicks on a document at position i, then with some probability they have seen all the URLs preceding position i. The rule "Click $>$ Skip Above" is the most aggressive: it generates a preference for all preceding URLs that were skipped. The rules "Click $>$ Skip Previous", and "Last Click $>$ Skip Above" are more conservative in generating preferences. The former generates a preference only with the preceding URL, while the latter assumes that only the last click was a successful click. The rule "Click > Click Above" creates preferences for failed clicks.

These rules attempt to generate data that will "correct" the ranking. One limitation is that they do not fire in the case that there is a single click on the first result, a common case in the context of a real search engine.

In isolation, these rules can lead to incorrect inferences. If we only apply positive rules we cannot obtain information about incorrect decisions made by the ranking function. On the other hand, negative rules do not provide any information about the correct decisions of the ranking function.

Even in combination, these rules can lead to non-intuitive inferences. Consider combining the "Click $>$ Skip Next" rule with the "Click $>$ Skip Above" rule and suppose that 100 people click only on position 1, while 10 people click on position 3. Then the combination of these rules implies that users prefer 1 to 2, users prefer 3 to 1 and also 3 to 2. Chaining the inferences together, we have that $3 > 1 > 2$. For many queries this turns out to be an incorrect conclusion. For example, if the query is ambiguous, 1 could be the most popular intent, and 3 the second intent.

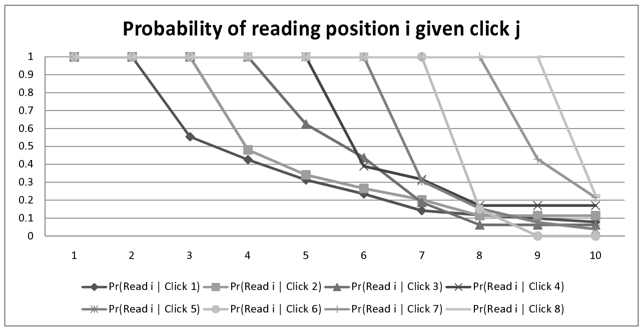

Figure 1: The probability a user reads position i given that they clicked on position j.

3.2 A Probabilistic Interpretation of the Click Log

As we outlined above, Joachims et al's rules rely on the most common behavior of the users to generate preferences. How can we capture the aggregate behavior of users instead of the most common user's behavior? We give a probabilistic method of interpreting the click log that is also informed by eye-tracking studies. In Figure 1, we show a chart adapted from the work of Cutrell and Guan [7, 8, 17] describing the probability a user reads position i given that they click on position j. In their study, hand-crafted search engine result pages were created in an information-seeking exercise, varying the position of the definitive document URL. They measured the positions viewed by the test subjects when a click occurred at a specific position in the top 10 result set. We augment this study with the rules Click > Skip Next and Click > Skip Above, so that $\Pr(\text{Read } i | \text{ Click } j) = 1$ for $1 \leq i \leq j+1$. For example, consider the $\Pr(\text{Read } i | \text{ Click } 1)$ line denoted by a diamond. The chart shows that with probability 1 position 2 is read, with probability 0.5 position 3 is read, etc. Note that the drop-off is surprisingly not steep with about 10% of users actually reading position 10. The second line $\Pr(\text{Read } i | \text{ Click } 2)$, denoted with a square, is similar to the first, but "pulled over" one position. In fact, each subsequent line seems pulled from the previous. In all of these cases, the drop-off in reading lower positions is gradual. Note that the lines $\Pr(\text{Read } i | \text{ Click } 9)$ and $\Pr(\text{Read } i | \text{ Click } 10)$ are omitted as they are equal to 1 for all i.

Using this observed behavior of the users, we generate preferences between URLs for a given query with probability proportional to what is observed in Figure 1. First, we decide on the rules that we apply. We want to capture both the positive and the negative feedback of the clicks, thus we use the "Click > Skip Next" and "Click > Skip Above" rules. Our goal is to create a per query *preference graph*: for each query, we construct a weighted directed graph where the vertices correspond to URLs and a directed edge from vertex u to v indicates the number of users who read u and v, clicked on u and skipped v. This graph forms the basis for the labels we construct.

Our method of interpreting the click log proceeds as follows.

1. Let $p_{ij} = \Pr(\text{Read } i \mid \text{Click } j)$ (from Figure 1). Suppose that for a given query in a given session a user clicks on position j. Then for all skipped URLs at position $i \neq j$, with probability p_{ij} we increase the weight of edge (URL at position j, URL at position i) by 1 and with probability $1 - p_{ij}$, we do

nothing.

2. Aggregating this probabilistic information over all users, we obtain an initial preference graph. Edges/preferences with very low weight are then removed entirely. The reason is that we do not want to create edge preferences based on spurious or inadvertent clicks.

Note that the way we generate the preferences addresses the limitations of Joachims et al's rules. We incorporate both positive and negative feedback, and we make use of the aggregate behavior of the users. Also, as a result of the probabilistic interpretation, the problem of incorrect inferences is alleviated. In the previous example, clicks on position 1 generate preferences to all positions from 2 to 10 (of varying weight).

Remark: The data in Figure 1 is limited in many regards. First, it is based on an eye-tracking study that involved a limited number of participants in a laboratory setting. Thus, the number may not reflect the true probability that a user reads a position. Furthermore, reading probability is often query dependent: navigational queries are unlikely to lead to reading many URLs after the navigational result, while informational or observational queries are likely to lead to much more reading behavior among all ten positions. Query dependent reading probabilities could be adapted from the work of [6, 10] and we leave this as a direction for future work. Finally, reading probabilities may be user dependent (targeted vs. browsing) and time dependent (weekday vs. weekend).

At this point, we could simply provide the edges of the graph as training data to a machine learning algorithm. The problem is that the edges do not tell a complete story of user preferences: Some edges that do exist should be removed as they form contradictory cycles. Also, some edges that do not exist should be added as they can be transitively inferred. In the next section, we show how to automatically generate labels from the graph with the goal of building a more complete story.

4. COMPUTING LABELS USING PAIRWISE PREFERENCES

Given the collection of user preferences for a query we now assign labels to the URLs such that the assignment is consistent with the preferences of the users. Recall that we model the collection of preferences for a query as a directed graph. The set of vertices corresponds to URLs for which we generated a pairwise preference. The set of edges captures the pairwise preferences between the URLs. The weight of an edge captures the strength of the preference. Given this graph representation, we define the labeling problem as a graph partition problem: assign the vertices to ordered classes so as to maximize the weight of the edges that agree with the classes minus the weight of the edges that disagree with the classes. The classes correspond to the relevance labels, e.g., Perfect, Excellent, Good, Fair and Bad.

4.1 Problem Statement and Complexity

We now formally define the problem. Let $G = (V, E)$ denote the preference graph, and let $\Lambda = \{\lambda_1, \dots, \lambda_K\}$ denote a set of K *ordered* labels, where $\lambda_i > \lambda_j$, if $i < j$. Let $L : V \rightarrow \Lambda$ denote a labeling of the nodes in G, with the labels in Λ, and $L(v)$ be the label of node v. The labeling function defines an ordered partition of the nodes in the graph into K disjoint classes $\{L_1, ..., L_K\}$ where $L_i = \{v : L(v) = \lambda_i\}$. We use L interchangeably to denote both the labeling and the partition. We say that the edge (u, v) is a *forward* edge in the labeling L, if $L(u) > L(v)$ and that the edge (u, v) is a *backward* edge if $L(u) < L(v)$. We define F to be the set of forward edges and B to be the set of backward edges. These are the edges that cross the classes in the partition in either forward or backward direction. Intuitively, the edges in F capture the preferences that agree with the labeling L, while the edges in B capture the preferences that disagree with the labeling. Given the graph G, our objective is to find a labeling L that maximizes the weight of edges in F and minimizes the weight of edges in B. We thus define our problem as follows.

PROBLEM 1. *(*MAXIMUM-ORDERED-PARTITION*) Given a directed graph $G = (V, E)$, and an ordered set Λ of K labels find a labeling L such that the* net agreement weight

$$A_G(L) = \sum_{(u,v)\in F} w_{uv} - \sum_{(u,v)\in B} w_{uv}$$

is maximized.

We now study the complexity of the MAXIMUM-ORDERED-PARTITION problem. We first consider the case that $K = 2$, that is, we want to partition the data points into two classes, so as to maximize the net agreement weight. The problem is reminiscent of MAX DI-CUT, the problem of finding a maximum directed cut in a graph, which is known to be NP-hard. However, surprisingly, our problem is not NP-hard. In the next theorem we show that there is a simple linear-time algorithm to compute the optimal partition: Vertices with net weighted outdegree greater than net weighted indegree are placed on one side of the partition and all other vertices are placed on the other side of the partition.

THEOREM 1. *The* MAXIMUM-ORDERED-PARTITION *problem can be solved in time $O(|E|)$ when the label set Λ contains $K = 2$ classes.*

PROOF. Let $\Lambda = \{\lambda_1, \lambda_2\}$ be the ordered classes, and let $L = \{L_1, L_2\}$ denote a labeling L. For every node $u \in V$, we also compute the difference between the outgoing and incoming edge weight for node u,

$$\Delta_u = \sum_{v\in V:(u,v)\in E} w_{uv} - \sum_{v\in V:(v,u)\in E} w_{vu} \ . \qquad (1)$$

The key observation is that the net agreement weight of labeling L can be expressed as $A_G(L) = \sum_{u\in L_1} \Delta_u$. We have that

$$\begin{aligned}\sum_{u\in L_1}\Delta_u &= \sum_{u\in L_1}\left(\sum_{v\in V:(u,v)\in E} w_{uv} - \sum_{v\in V:(v,u)\in E} w_{vu}\right)\\ &= \sum_{u\in L_1, v\in L_1:(u,v)\in E} w_{uv} - \sum_{u\in L_1, v\in L_1:(v,u)\in E} w_{vu}\\ &\quad + \sum_{u\in L_1, v\in L_2:(u,v)\in E} w_{uv} - \sum_{u\in L_1, v\in L_2:(v,u)\in E} w_{vu}\\ &= \sum_{(u,v)\in F} w_{uv} - \sum_{(v,u)\in B} w_{vu}\\ &= A_G(L)\end{aligned}$$

Note that the values Δ_u for nodes $u \in V$ depend only on the graph G, and can be computed independent of the actual labeling L. Given a set of positive and negative numbers, the subset that maximizes the sum is the set of all positive numbers. Therefore, it follows that in order to maximize $A_G(L)$, the optimal labeling L^* should place all nodes u with $\Delta_u > 0$ in class L_1, and all the nodes with $\Delta_u < 0$ in class L_2 (nodes with $\Delta_u = 0$ can be placed in either class). Computing Δ_u for all nodes can be done in time $O(|E|)$. $\square$

In the general case, when the number of classes is unlimited, the problem is NP-hard. The proof, given in the Appendix, follows even in the case that $K = n - O(1)$. We leave open the complexity of the problem for $K > 2$, and $K < n - O(1)$.

THEOREM 2. *The* MAXIMUM-ORDERED-PARTITION *problem is NP-hard when the set of labels Λ contains $K = n$ labels.*

4.2 Algorithms

We discuss heuristics for MAXIMUM-ORDERED-PARTITION next. Our algorithms proceed in two steps. First, we describe algorithms for computing a linear ordering (possibly with ties) of the nodes in the graph that tend to rank higher the nodes that attract more preferences. Then, we apply a dynamic programming algorithm to find the optimal partition L^* of this linear order into at most K classes such that $A(L)$ is maximized.

4.2.1 Obtaining a Linear Ordering

We consider three different techniques for obtaining a linear ordering (possibly with ties) which we outline below.

Δ-ORDER: We compute the value Δ_u as defined in Equation 1 for each node in the graph. We then order the nodes in decreasing order of these values. This technique is inspired by the fact that dynamic programming applied to this ordering gives an optimal solution in the case of a 2-partition (Theorem 1).

PIVOT: We adapt the Bucket Pivot Algorithm (PIVOT) proposed by Gionis et al. [16] which is a method for finding good bucket orders. A bucket order is a total order with ties, that is, we partition the set of nodes into ordered buckets such that nodes in an earlier bucket precede nodes in a later bucket but nodes within a bucket are incomparable. First we compute the transitive closure of the graph so as to capture transitive preference relationships between pairs of nodes. The algorithm proceeds recursively as follows: select a random vertex v as the pivot and divide the remaining nodes into three classes ("left", "same", "right") by comparing with v. The "left" class contains all nodes that are incoming neighbors of v but not outgoing neighbors of v (formally, the set $\{u|(u,v) \in E$ and $(v,u) \notin E\}$) and the "right" class contains all nodes that are outgoing neighbors of v but not incoming neighbors of v. The "same" class contains v and also the remaining nodes (nodes that are not neighbors of v and nodes that have both incoming edge and outgoing edge with v). The algorithm then recurses on the "left" class, outputs the "same" class as a bucket and recurses on the "right" class.

PAGERANK: PageRank is a popular algorithm for ranking web pages based on the web graph. The algorithm performs a random walk on the graph, where at each step, when at node u, with probability $1-\alpha$ the random walk follows one of the outgoing links of node u chosen uniformly at random, or with probability α it jumps to a page chosen uniformly at random from the set of all pages in the graph.

The algorithm is easy to adapt to our setting. Given the graph G, we create the *transpose* graph G^T, where the direction of all edges is inverted. Intuitively, in graph G, an edge (u,v) means that u is preferred over v. In graph G^T, the corresponding edge (v,u) means that node v gives a recommendation for node u to be ranked high. A very similar intuition governs the application of PageRank on the Web. Instead of choosing outgoing links of node u uniformly at random, we choose an outgoing link (u,v) proportional to its weight, w_{uv}.

The result of the PageRank algorithm on the graph G^T is a score for each node in the graph given by the stationary distribution of the random walk. We order the nodes of the graph in decreasing order of these scores.

4.2.2 Finding the Optimal Partition

Given the linear ordering of the nodes we now want to segment it into K classes, such that we maximize net agreement. We can find the optimal segmentation using dynamic programming. Assume that the linear ordering produced by the algorithm is $v_1, v_2, \ldots, v_n$. Generating a K-segmentation is equivalent to placing $K-1$ breakpoints in the interval $[1, ..., n-1]$. A breakpoint at position i partitions the nodes into sets $\{v_1, \ldots, v_i\}$ and $\{v_{i+1}, \ldots, v_n\}$. Let OPT denote a two-dimensional matrix, where $OPT[k,i]$ is the optimal net agreement when having k breakpoints, where the last breakpoint is at position i. Given the values of this matrix, we can find the optimal net agreement for a segmentation with K classes by computing the maximum of the $(K-1)$-th row. That is, for the optimal partition L^* we have that

$$A_G(L^*) = \max_{K-1 \leq i \leq n-1} OPT[K-1, i].$$

We will fill the entries of the OPT matrix using dynamic programming. We define another two-dimensional matrix B, where $B[j,i]$ is the added benefit to the net agreement if we insert a new breakpoint at position i to a segmentation that has the last breakpoint at position j. For $k > 1$, it is easy to see that

$$OPT[k,i] = \max_{k-1 \leq j \leq i} \{OPT[k-1,j] + B[j,i]\}. \tag{2}$$

For $k = 1$ we have that $OPT[1,i] = B[0,i]$. We can now fill the matrix OPT in a bottom up fashion using Equation 2. The cost of the algorithm is $O(Kn^2)$, since in order to fill the cell $OPT[k,i]$ for row k, and column i we need to check $i-1$ previous cells. We have $K-1$ rows and n columns, hence $O(Kn^2)$.

Computing the values of matrix B can be done by streaming through the edges of the graph, and incrementing or decrementing the entries of the table that a specific edge affects. Given an edge $e = (v_x, v_y)$, let $\ell = \min\{x,y\}$ and $r = \max\{x,y\}$. We only need to update the cells $B[j,i]$, where $0 \leq j \leq \ell - 1$, and $\ell \leq i \leq r$. In this case, the edge e falls after the last endpoint j of the existing segmentation, and i falls between the two endpoints of e. That is, the new breakpoint at i will be the first to cut the edge (v_x, v_y). If $x < y$ then the edge e is a forward edge and we increment $B[j,i]$ by w_e. If $x > y$ then edge e is a backwards edge and we decrement $B[j,i]$ by w_e. This computation is described in Algorithm 1. The cost of the algorithm is $O(n^2|E|)$, since for each edge we need to update $O(n^2)$ cells.

Finding the optimal K-segmentation comes as a byproduct of the computation of the matrix OPT, by keeping track of the decisions made by the algorithm, and tracing back from the optimal solution.

4.2.3 Label Assignment

We run the above algorithms with $K = 5$ as we want to assign URLs for a given query into five ordered classes. If the dynamic programming returns five non-empty classes, this assignment is unique. Otherwise if there are $M < 5$ non-empty classes, we provide a heuristic to steer us towards a desirable labeling. For space reasons, we omit the details as the pairwise comparison experiments in Section 5 are unaffected by this choice of labeling. The main idea is as follows. Since we are interested in a small number of labels, we enumerate over all possible assignments of

Algorithm 1 The algorithm for computing matrix B

```
Input: The edge set $E$ of the graph $G$.
Output: The added benefit matrix $B$.
1: for all $e = (v_x, v_y) \in E$ do
2:   $\ell = \min\{x, y\}; r = \max\{x, y\}$.
3:   for all $j = 0, \ldots, \ell - 1$ do
4:     for all $i = \ell, \ldots, r$ do
5:       if $x < y$ then
6:         $B[j, i] = B[j, i] + w(v_x, v_y)$
7:       else
8:         $B[j, i] = B[j, i] - w(v_x, v_y)$
9:       end if
10:    end for
11:  end for
12: end for
```

five labels to the M classes. Each assignment is scored using the inter-cluster and the intra-cluster edges. For inter-cluster edges, if users express a strong preference for one class over another, the labeling is rewarded for a class assignment that separates the classes as much as possible. For a cluster with many intra-cluster edges, the labeling is penalized for assigning the extreme classes Perfect and Bad. The justification is that we would not expect users to prefer one URL over another within a cluster of Perfect URLs (or Bad URLs). These inter and intra-cluster scores can then be appropriately combined to produce a single score. The labeling with the highest score is selected as the best labeling.

5. EXPERIMENTS

The goal of our experiments is to understand whether click-based labels align with the opinion of a panel of judges. We compare different techniques for generating the preference graph and then evaluate different labeling algorithms. Before delving into these components, we describe our experimental setup.

5.1 Experimental Setup

We obtained a collection of 2000 queries with an average of 19 URLs per query, where each (query, URL) pair was manually labeled by eleven judges for relevance. The ordinal judgments were converted into numeric values by assigning the scores 1-5 to the labels Bad-Perfect, respectively. For this collection of 2000 queries, we also obtained click activity for one month of click logs.

Performance is based on the extent to which click labels align with the Consensus Opinion as well as with the Contrasting Opinion of the panel of judges.

Consensus Opinion: For a given query, consider two URLs that are evaluated by the same eleven judges. Intuitively, we would like to show that the more the judges reach a consensus about the two URLs, the more click preferences agree. This is exactly the intuition for our Consensus charts. Pairs of URLs are grouped into those where six, seven,..., eleven out of eleven judges *agree* on two URLs, where *agree* means the two URLs are either given the same label by the group or all agree that one is better than the other. In the graph construction experiments, we compare the extent to which our *edges* align with increasing consensus opinion. In the labeling experiments, we compare the extent to which our *click-labels* align with increasing consensus.

Contrasting Opinion: A different question is whether click labels are more likely to agree the sharper the contrast in the opinion of the judges. We quantify the panel's opinion of a URL by the average score assigned to the URL. The contrast between two URLs is the difference between the average scores. Intuitively, the sharper the contrast in opinion between two URLs, the more click preferences ought to agree. Note the difference between Consensus and Contrasting opinions: Judges can reach a consensus by agreeing that two URLs are an equal match to a query, but equally good matches are not a contrast of opinion.

5.2 Evaluating the Preference Graph

In Section 3 we outlined several methods that have been previously explored in the literature, as well as a probabilistic interpretation that generates a preference between two URLs according to the expected number of times that a URL was clicked and another URL below it was read but not clicked. In this section, we consider six rules for interpreting click data and the quality of each rule for generating the preference graph. Table 1 contains the six rules, and the associated rule identifier used in subsequent tables to refer to the rule. Rules R1 to R5 are taken from [21], and R6 is the probabilistic click log interpretation developed in Section 3.

Table 1: Rules for using click data to generate a preference graph

Rule ID	Rule
R1	Click > Skip Above
R2	Last Click > Skip Above
R3	Click > Click Above
R4	Click > Skip Previous
R5	Click > Skip Next
R6	Probabilistic Click Log Rule

To evaluate the rules, we generate a graph for each of the rules R1 to R6. Edges are added to the graph when the number of user clicks generating the rule exceed a predefined threshold of 15. Due to ranking instability over time and the bidirectional nature of some of the rules, the graph is a directed graph and edges can exist in both directions between two URLs. Once the graph is complete, we enumerate the edges of the graph and compare the relative preferences to those of the judges.

We present the results of the comparison in Table 2. For the column labeled “Judges Agree”, we have values 6 through 11 that represent the number of judges agreeing on a relative preference for a pair of URLs. Their agreement may be that one URL is more relevant to the query than the other, or that the two URLs have the same relevance. We use the edges of the graph to find the click-based preference according to the rule that generated the edge. When both directional edges between the two URL vertices in the graph exist, the preference is generated according to the edge with the highest click count. Therefore, the click rules will almost always express a preference of one URL or the other. When the judges have a consensus of the two URLs being equally relevant, this will almost always show up as the click rule disagreeing with the consensus of the judges. The columns labeled “A” and ”D” in the table refer to the rule Agreement or Disagreement with the judge consensus, respectively.

The results show that R6 is the best performing rule on our dataset. The total agreement with our judges for R6 is 401 out of 809 total URL pairs, or 49.6%. The next best performing rule is R5 at 44.2%, but has only 224 total pairs in common with our judges. Note that each rule generates a different count of preference pairs. For example, rule R3 (Click > Click Above) has a very low count of pairs ($6 + 97 = 103$) due to the requirement of two clicks in the same query impression. The strength of R6 as a preference rule is

Table 2: For each rule (R1-R6), the table shows the number of edges that agree (A), disagree (D) together with the percent of edges that agree (A %) as the consensus of the judges grows stronger. The last row shows the net agreement

Judges Agree	R1 (A;D) A %	R2 (A;D) A %	R3 (A;D) A %	R4 (A;D) A %	R5 (A;D) A %	R6 (A;D) A %
6	(16;72) 18.2%	(15;69) 17.9%	(2;20) 9.1%	(5;22) 18.5%	(10;32) 23.8%	(47;121) 28.0%
7	(11;69) 13.8%	(10;64) 13.5%	(1;19) 5.0%	(5;18) 21.7%	(13;30) 30.2%	(55;95) 36.7%
8	(12;60) 16.7%	(10;55) 15.4%	(1;26) 3.7%	(4;19) 17.4%	(18;18) 50.0%	(66;70) 48.5%
9	(9;31) 22.5%	(9;27) 25.0%	(1;13) 7.1%	(4;14) 22.2%	(10;14) 41.7%	(47;43) 52.2%
10	(8;45) 15.1%	(7;42) 14.3%	(1;19) 5.0%	(3;22) 12.0%	(28;21) 57.7%	(74;46) 61.7%
11	(5;40) 11.1%	(5;36) 12.2%	(0;0) 0.0%	(3;8) 27.3%	(20;10) 66.7%	(112;33) 77.2%
Total	(61;317) 16.1%	(56;293) 16.0%	(6;97) 5.8%	(24;103) 18.9%	(99;125) 44.2%	(401;408) 49.6%

that it does not have to trade accuracy for less number of generated preference pairs. Rather, it has both higher accuracy and a higher number of preference pairs than any other rule.

Drilling into the breakdown of the number of judges in consensus, we note that the performance of R6 improves when more judges are in consensus. For example, when 11 judges are in consensus, rule R6 agrees 112 times and disagrees 33 times on the relative order of URLs. We also note that R6 has nearly 50% or better agreement when eight or more judges are in consensus. It is impressive considering that a disagreement ensues when there is consensus among judges that two URLs are equal.

We observe that the rules that contradict the existing order (R1, R2, R3, R4) perform worse compared to the rules that fully or partially reinforce the existing order (R5 and R6). This suggests that, for the queries in our dataset, the order of search results is in agreement with the judgments. Thus, the conclusions may not be completely indicative of the performance of different rules.

5.3 Evaluating Click Labels

We now compare the different algorithms for generating labels. For all the experiments we use the preference graph generated by Rule R6. We consider the three algorithms described in Section 4 for generating a linear ordering of the URLs and then find the optimal partition using dynamic programming. The algorithms will be compared in terms of Consensus and Contrasting Opinion.

The baseline algorithm assigns labels at random, denoted by RANDOM. In order to make a fair comparison, RANDOM selects labels according to the same distribution as the manually judged labels over the URLs labeled by our algorithm. This distribution is $(10\%, 16\%, 30\%, 30\%, 14\%)$ for the labels (Perfect, Excellent, Good, Fair, Bad) respectively. Given a query and two URLs, u_1 and u_2, if we guess according to this distribution then the probability that u_1 has the same label as u_2 is $0.1^2 + 0.16^2 + 0.3^2 + 0.3^2 + 0.14^2 = 0.24$. The probability that u_1 is better than u_2 is equal to the probability that u_1 is worse than u_2 and consequently is $(1 - 0.24)/2 = 0.38$. RANDOM is used to show that our results are not due to chance.

Prior to showing the charts, we remark on a few experimental details. First, clicks with low dwell time were removed. Prior work suggests that low-dwell time clicks may be spurious and not necessarily an indication of relevance [14]. We do not report results on dwell time values, but only mention that after trying several thresholds, removed clicks with dwell time of less than fifteen seconds. A second practical issue is duplicate URLs that surface in the top ten results. In such a situation, users prefer to click on the more definitive URL. In our experiments, among the duplicate URLs, we keep only the URL with the highest label.

5.3.1 Consensus Opinion

Table 3: Click Label Consensus: The chart shows the agreement of each labeling method as the consensus of the panel grows stronger. Click labels are more likely to agree the stronger the consensus

Judges Agree	RANDOM	PIVOT	Δ-ORDER	PAGERANK
6	31.0%	42.6%	34.1%	38.9%
7	29.4%	43.3%	42.8%	50.2%
8	31.7%	43.0%	42.9%	49.5%
9	37.8%	57.8%	47.9%	56.9%
10	32.9%	53.1%	60.0%	59.6%
11	33.9%	65.7%	73.3%	79.0%
Total	32.4%	49.4%	48.4%	54.0%

We begin by showing that the stronger the consensus opinion, the more likely click labels are to agree. The results are shown in Table 3. Observe that the baseline RANDOM does not improve whether the consensus consists of six or eleven out of eleven judges. On the other hand, the agreement with all three labeling algorithms progressively increases the stronger the consensus. The PAGERANK ordering has the most affinity with the consensus.

Next, we compare Click Label Consensus (Table 3) with Graph Consensus (Table 2). The first observation is that the total PAGERANK percent agreement (54%) is higher than the graph generation consensus percent agreements (50% for R6)). In addition, there are more actual agreement pairs. For the PAGERANK method, there is agreement on 570 pairs and disagreement on 486 pairs compared with R6 which agrees on 401 pairs and disagrees on 408 pairs.

When Click Labels Disagree with the Consensus: We find that the stronger the judge consensus, the more click labels agree. However, the results raise a legitimate concern: when ten or eleven out of eleven judges agree on a pair of URLs, how could click labels possibly disagree? We studied these (query, URL) pairs and the results were quite interesting: the largest difference stems from ambiguous query intent.

Most of the (query, URL) pairs (83%) where the consensus preference deviated from the click preference were short, i.e., one or two key words. This gave us an early indication that the queries could be ambiguous. Second, more than half (57%) of the (query, URL) pairs were judged to be equally good by the entire panel. In other words, these were pairs of URLs where the judges had a difficult time distinguishing between their relevance, while click-labels had a preference.

We manually grouped the queries and pairs of URLs where click labels disagree with the consensus into five categories listed in Table 4. The largest category is 'Depends on Intent' constituting 45% of the queries. One example of such a query is "vogue" where the

Table 4: Breakdown of where Judge Consensus Deviates from Click Labels

Category	**% Queries**
Depends on Intent	44.7%
Click Labels Incorrect	21.1%
Duplicate URLs	15.8%
Click Labels Correct	1.3%
Other	17.1%

two choices are the UK Vogue home page and the Australian Vogue home page. To a judge, the pages seem equally good, but the US clickers express a preference for the UK site. Another example is "dutch oven cooking" where one URL gives recipes for cooking in a dutch oven and another gives information on how to purchase dutch oven cookware. The judges claim the sites are equally good, but clickers strongly prefer the recipes. It is difficult for us to know which is correct: we did not pose the query with a specific intent and it is not likely that the panel of judges did either. The people who do know the intent are the clickers who themselves posed the query with a specific interest in mind. Arguably, we believe that the aggregate click signal is thus better able to resolve the popular preference between these URLs than a judge who never asked the query.

The second largest category of queries is where we find the judgment of the panel of judges correct (21%). Further manual analysis revealed that one source of the problem (25% of the cases where the panel of judges is correct) was due to presentation bias in search results. In other words, some URLs were assigned a higher click label simply because they were positioned higher on the page and thus received more clicks. Thus, although our technique has some implicit correction for position bias inbuilt, it may benefit from further work in removing the bias.

A third category of queries is 'duplicate URLs' (16%). While we did remove the duplicates we found, our duplicate detection method is not perfect. The remaining duplicates actually raised our awareness of another difference between clicks and judges. For example, for the query "donald trump", the two pages `trump.com` and `trump.com/main.htm` are duplicates of each other. The labels assigned by the panel of human judges are, as expected, the same for these two URLs. However, the click signal is stronger for the first, shorter URL. We observed this common trend among duplicate URLs: given two identical pages, a snippet containing a shorter URL is more likely to attract clicks than a longer URL. Thus, it may actually be desirable to train a machine learning algorithm to prefer URLs that are shorter in length.

The last two categories are where click labels are definitely correct (1%) and Other (17%). The other category includes foreign language queries, together with English queries that we could not understand.

To summarize, while click labels are not perfect in that presentation bias is an issue, the largest difference is queries with ambiguous intent. Our belief is that clickers are better able to tease out the relative relevance of URLs when queries are ambiguous.

5.3.2 *Contrasting Opinion*

In the next set of experiments, we show that the sharper the contrast of opinion between two URLs, the more likely that click labels agree. The opinion of a panel for a (query, URL) pair is just the average score of the eleven judges. The contrast of opinion between two URLs is the difference in the average score. We denote this difference by δ. If the difference is small ($< \gamma$) then we say there is no contrast. The value γ is set to 0.4 in our experiments.

Table 5: Guide to the definition of strong and weak (dis)agreements

	Clicks $u_1 > u_2$	Clicks $u_1 = u_2$	Clicks $u_1 < u_2$
Avg(u_1) $>$ Avg(u_2) $+ \gamma$	Strong A	Weak D	Strong D
$\lvert$Avg(u_1)$-$ Avg(u_2)$\rvert < \gamma$	Weak D	Weak A	Weak D

Table 6: Agreement metric for different label generation algorithms

Agree	RANDOM	PIVOT	Δ-ORDER	PAGERANK
Weak A	8.40%	15.3%	10.5%	10.3%
Strong A	25.30%	32.1%	38.5%	44.1%
Total A	33.70%	47.4%	49.0%	54.4%
Disagree				
Weak D	42.40%	42.9%	37.0%	36.6%
Strong D	23.80%	9.7%	13.9%	9.0%
Total D	66.30%	52.6%	51.0%	45.6%

To facilitate the discussion, we introduce the terminology weak and strong (dis)agreements based on the confusion matrix between click-label preferences and panel preferences, Table 5. Strong agreements are important in that they preserve the pairs of URLs used for training. Weak agreements are good but less important for training algorithms that do not train on pairs with the same label. Strong disagreements are training pairs that are flipped in the opposite direction. All other disagreements are weak.

Then we consider sharp contrasts of opinion. It is important that pronounced judge preferences be accurately reflected in click labels. Judges are typically very good at discriminating between Perfect and Bad results. It is less important to accurately capture slight judge preferences since judges often randomly decide between nearby labels such as Good and Fair – equally likely going one way or the other. Consequently, we bucket the triples (query, URL1, URL2) into those where the contrast of opinion δ is low to high and study the extent to which click labels agree.

Net Agreements: Table 6 shows the agreement metrics for the different algorithms. The PAGERANK algorithm produces the highest agreement and lowest disagreement. The PIVOT algorithm produces a small fraction of strong disagreements, but many more weak agreements and weak disagreements. This is due to the fact that it over zealously groups many URLs into the same bucket, probably due to poor pivot choices. The Δ-ORDER algorithm is better than the PIVOT algorithm, but not as good as the PAGERANK algorithm. This is mostly due to the fact that the ordering by the Δ_u values is performed via only "local" information for a node, and not incorporating information about the graph as a whole. This information is incorporated to some extent by the PAGERANK algorithm, which captures the transitivity of preferences, and thus yields an ordering with a more "global" view.

Sharper Contrasts: Next, we show that click-labels are more likely to agree with the panel of judges the sharper the contrast, i.e., the larger the value of δ. A good labeling algorithm should have higher agreement for larger values of δ since it guarantees that pairs of URLs with a marked contrast in panel preference are similarly preferred by click labels.

The results are shown in Figure 2. For RANDOM, click label agreements do not increase with sharper contrast. The PIVOT algorithm is dominated by weak disagreements, even for strong contrasts. On the other hand, for both Δ-ORDER and PAGERANK,

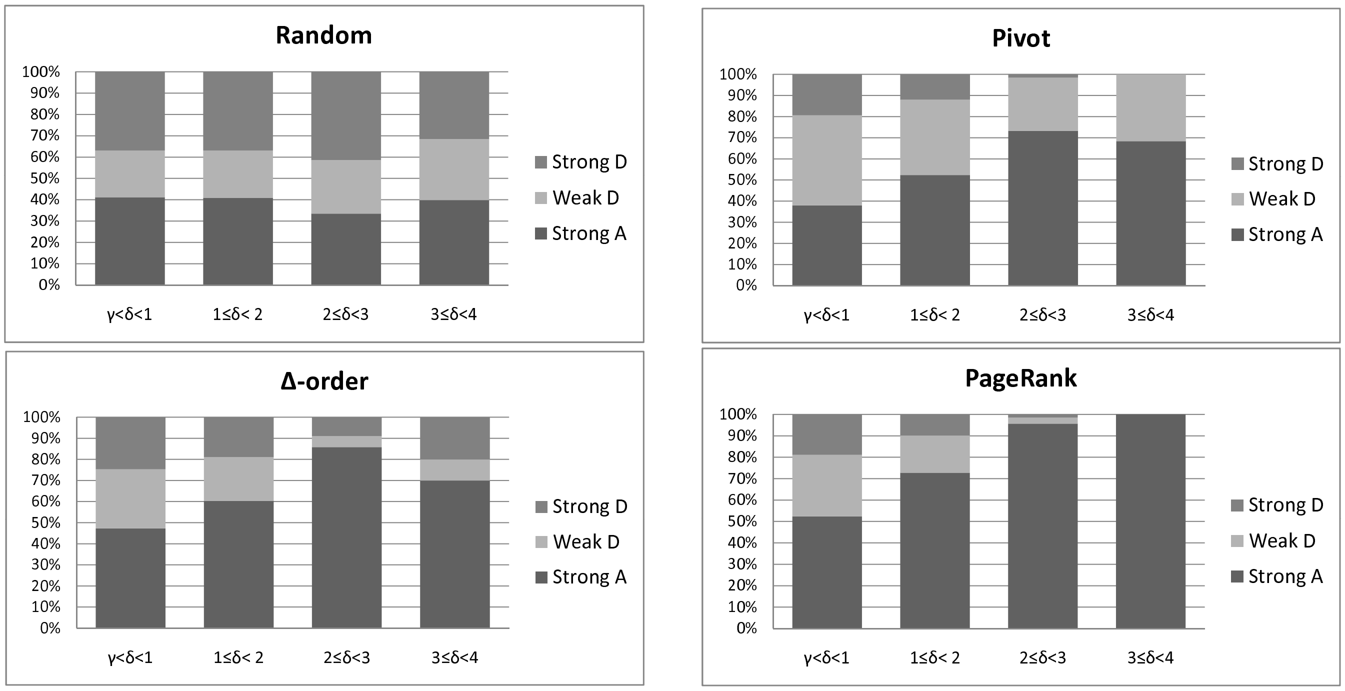

Figure 2: Contrasting Opinions: The figure shows that the sharper the contrast between the opinion of the judges, measured by larger values of δ, the more likely click labels agree.

the sharper the contrast the greater the agreement. PAGERANK actually agrees completely when the contrast is at least three, and nearly completely when the contrast is at least two. The charts demonstrate the success of click labels: the further apart the panel's average scores are, the more likely that click labels agree.

6. SUMMARY AND FUTURE WORK

We described a method for inferring a per query click-skip graph based on eye-tracking studies. Relevance labeling was modeled as an ordered graph partitioning problem, and an optimum linear-time solution was given in the two labeling case. While the optimum n labeling is NP-hard, heuristics were proposed to address the multiple label generation problem.

Experiments demonstrate that our method of inferring preferences align more with the consensus opinion of a panel of judges than prior methods. Further, our click labeling procedures also align with the consensus opinion of a panel of judges. In the event that click labels do not agree with the panel, it is often due to ambiguous queries where our belief is that clickers are better able to resolve popular intent. Finally, as the panel's view of one URL sharply contrasts with their view another URL for a query, we find that click labels are more likely to agree.

For future work, we would like to train ranking functions with click labels. Some practical problems will have to be tackled. Clicks can generate labels for many queries – potentially more queries than any ranking algorithm can use. Thus, it may be useful to find ways to score click labels by the confidence of the label. Such a scoring could be used to determine which click labels to use while training. Another question is one of evaluation: it will be important to determine when a ranking function trained on clicks performs better than a ranking function trained on human judgments. A ranking function trained on click labels will likely perform better on a click-label test set than a ranking function trained on human judgments. On the other hand, a ranking function trained on human judgments will likely perform better on a human judgment test set. We leave questions related to training ranking functions with click labels as an interesting avenue for future work.

7. ACKNOWLEDGMENTS

We are grateful to Sreenivas Gollapudi, Nazan Khan, Rina Panigrahy and John Shafer for valuable insights. We also thank Eric Brill, Chris Burges, Ken Church, Ed Cutrell, Dennis Decoste, Girish Kumar, Greg Linden, Rangan Majumder, Alex Ntoulas, Ryan Stewart, Krysta Svore, Rohit Wad and Yi-Min Wang for thoughtful feedback. Finally, we thank Gu Xu for help with URL deduplication.

8. REFERENCES

[1] Rakesh Agrawal, Ralf Rantzau, and Evimaria Terzi. Context-sensitive ranking. In *SIGMOD*, pages 383–394, 2006.

[2] Nir Ailon, Moses Charikar, and Alantha Newman. Aggregating inconsistent information: ranking and clustering. In *STOC*, pages 684–693, 2005.

[3] Christopher J. C. Burges, Robert Ragno, and Quoc Viet Le. Learning to rank with nonsmooth cost functions. In *NIPS*, pages 193–200, 2006.

[4] Christopher J. C. Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Gregory N. Hullender. Learning to rank using gradient descent. In *ICML*, volume 119, pages 89–96, 2005.

[5] Charles L. A. Clarke, Eugene Agichtein, Susan T. Dumais, and Ryen W. White. The influence of caption features on clickthrough patterns in web search. In *SIGIR*, pages 135–142, 2007.

[6] Nick Craswell and Martin Szummer. Random walks on the click graph. In *SIGIR*, pages 239–246, 2007.

[7] Edward Cutrell. Private communication. 2008.

[8] Edward Cutrell and Zhiwei Guan. What are you looking for?: an eye-tracking study of information usage in web search. In *CHI*, pages 407–416, 2007.

[9] Zhicheng Dou, Ruihua Song, Xiaojie Yuan, and Ji-Rong Wen. Are click-through data adequate for learning web search rankings? In *CIKM*, pages 73–82, 2008.

[10] Georges Dupret and Benjamin Piwowarski. A user browsing model to predict search engine click data from past observations. In *SIGIR*, pages 331–338, 2008.

[11] Cynthia Dwork, Ravi Kumar, Moni Naor, and D. Sivakumar. Rank aggregation methods for the web. In *WWW*, pages 613–622, 2001.

[12] Ronald Fagin, Ravi Kumar, Mohammad Mahdian, D. Sivakumar, and Erik Vee. Comparing and aggregating rankings with ties. In *PODS*, pages 47–58, 2004.

[13] Jianlin Feng, Qiong Fang, and Wilfred Ng. Discovering bucket orders from full rankings. In *SIGMOD*, pages 55–66, 2008.

[14] Steve Fox, Kuldeep Karnawat, Mark Mydland, Susan Dumais, and Thomas White. Evaluating implicit measures to improve web search. *ACM Trans. Inf. Syst.*, 23(2):147–168, 2005.

[15] Yoav Freund, Raj Iyer, Robert E. Schapire, and Yoram Singer. An efficient boosting algorithm for combining preferences. *JMLR*, 4:933–969, 2003.

[16] Aristides Gionis, Heikki Mannila, Kai Puolamäki, and Antti Ukkonen. Algorithms for discovering bucket orders from data. In *KDD*, pages 561–566, 2006.

[17] Zhiwei Guan and Edward Cutrell. An eye tracking study of the effect of target rank on web search. In *CHI*, pages 417–420, 2007.

[18] Nicole Immorlica, Kamal Jain, Mohammad Mahdian, and Kunal Talwar. Click fraud resistant methods for learning click-through rates. In *WINE*, pages 34–45, 2005.

[19] Bernard J. Jansen. Click fraud. *IEEE Computer*, 40(7):85–86, 2008.

[20] Thorsten Joachims. Optimizing search engines using clickthrough data. In *KDD*, pages 133–142, 2002.

[21] Thorsten Joachims, Laura A. Granka, Bing Pan, Helene Hembrooke, and Geri Gay. Accurately interpreting clickthrough data as implicit feedback. In *SIGIR*, pages 154–161, 2005.

[22] Thorsten Joachims, Laura A. Granka, Bing Pan, Helene Hembrooke, Filip Radlinski, and Geri Gay. Evaluating the accuracy of implicit feedback from clicks and query reformulations in web search. *ACM Trans. Inf. Syst*, 25(2), 2007.

[23] Michael J. Kearns and Umesh V. Vazirani. *An Introduction to Computational Learning Theory*. MIT press, Cambridge, Massachusetts, 1994.

[24] Nathan N. Liu and Qiang Yang. Eigenrank: a ranking-oriented approach to collaborative filtering. In *SIGIR*, pages 83–90, 2008.

[25] Filip Radlinski and Thorsten Joachims. Query chains: learning to rank from implicit feedback. In *KDD*, pages 239–248, 2005.

[26] Filip Radlinski and Thorsten Joachims. Minimally invasive randomization for collecting unbiased preferences from clickthrough logs. In *AAAI*, 2006.

[27] Filip Radlinski and Thorsten Joachims. Active exploration for learning rankings from clickthrough data. In *KDD*, pages 570–579, 2007.

[28] Michael J. Taylor, John Guiver, Stephen Robertson, and Tom Minka. Softrank: optimizing non-smooth rank metrics. In *WSDM*, pages 77–86, 2008.

[29] Ramakrishna Varadarajan and Vagelis Hristidis. A system for query-specific document summarization. In *CIKM*, pages 622–631, 2006.

[30] Ke Zhou, Gui-Rong Xue, Hongyuan Zha, and Yong Yu. Learning to rank with ties. In *SIGIR*, pages 275–282, 2008.

APPENDIX

A. PROOF OF THEOREM 2

We will first show that when $K = n$ there exists an optimal labeling such that each node is assigned a different label, that is, all classes defined by the labeling are singleton sets. Assume that this is not true, that is, for any optimal solution L, the number of non-empty classes generated by L is at most M for some $M < n$. Consider such an optimal labeling with exactly M non-empty classes. Then there must exist a label λ_i that is assigned to at least two nodes, and a label λ_j that is assigned to no node. Without loss of generality assume that $j = i + 1$. Let u be one of the nodes in the class L_i. Also let

$$\Delta_u(L_i) = \sum_{v \in L_i : (u,v) \in E} w_{uv} - \sum_{v \in L_i : (v,u) \in E} w_{vu}$$

denote the total weight of the outgoing edges, minus the weight of the incoming edges, for u when restricted to nodes in L_i. If $\Delta_u(L_i) \geq 0$, then we assign all the nodes $L_i \setminus \{u\}$ to label λ_{i+1}, resulting in a new labeling L'. The net agreement weight of L' differs from that of L, only with respect to the backward and forward edges introduced between class $L'_i = \{u\}$ and $L'_{i+1} = L_i \setminus \{u\}$. Therefore, we have that $A_G(L') = A_G(L) + \Delta_u(L_i) \geq A_G(L)$. Similarly, if $\Delta_u(L_i) < 0$, we create a labeling L' with classes $L'_i = L_i \setminus \{u\}$ and $L'_{i+1} = \{u\}$, with net weight $A_G(L') = A_G(L) - \Delta_u(L_i) \geq A_G(L)$. In both cases L' has net weight at least as high as that of L; therefore L' is an optimal solution with $M + 1$ classes, reaching a contradiction.

In the case that $K = n$, the problem becomes that of finding a linear ordering L of the nodes in V that maximizes $A_G(L)$. Note that in this case every edge of the graph will contribute to the net weight $A_G(L)$ either positively or negatively. That is, we have that $E = F \cup B$, and therefore,

$$\begin{aligned} A_G(L) &= \sum_{(u,v)\in F} w_{uv} - \sum_{(u,v)\in B} w_{uv} \\ &= \sum_{(u,v)\in F} w_{uv} - \left(\sum_{(u,v)\in E} w_{uv} - \sum_{(u,v)\in F} w_{uv} \right) \\ &= 2 \sum_{(u,v)\in F} w_{uv} - \sum_{(u,v)\in E} w_{uv} \,. \end{aligned}$$

The term $\sum_{(u,v)\in E} w_{uv}$ is constant, thus maximizing $A_G(L)$ is equivalent to maximizing $\sum_{(u,v)\in F} w_{uv}$. This problem is equivalent to the MAXIMUM-ACYCLIC-SUBGRAPH problem, where, given a directed graph $G = (V, E)$, the goal is to compute a subset $E' \subseteq E$ of the edges in G such that the graph $G' = (V, E')$ is an acyclic graph and the sum of edge weights in E' is maximized. As the MAXIMUM-ACYCLIC-SUBGRAPH problem is NP-hard and the edges in F are a solution to it, our problem is also NP-hard.