Learning Non-linear Ranking Functions for Web Search using Probabilistic Model Building GP

Hiroyuki Sato*, Danushka Bollegala*, Yoshihiko Hasegawa* and Hitoshi Iba*
* The University of Tokyo, Tokyo, Japan 113-8654

***Abstract*—Ranking the set of search results according to their relevance to a user query is an important task in an Information Retrieval (IR) system such as a Web Search Engine. Learning the optimal ranking function for this task is a challenging problem because one must consider complex non-linear interactions between numerous factors such as the novelty, authority, contextual similarity, etc. of thousands of documents that contain the user query. We model this task as a *non-linear* ranking problem, for which we propose *Rank-PMBGP*, an efficient algorithm to learn an optimal non-linear ranking function using Probabilistic Model Building Genetic Programming. To our knowledge, ours is the first attempt to learn a non-linear ranking functions for IR using evolutionary algorithms. We evaluate the proposed method using the LETOR dataset, a standard benchmark dataset for training and evaluating ranking functions for IR. In our experiments, the proposed method obtains a Mean Average Precision (MAP) score of 0.291, thereby significantly outperforming a non-linear baseline approach that uses Genetic Programming.**

I. Introduction

The amount of information available on the Web continues to grow exponentially by the day. It is no longer the case that the information we seek do not exist in the Web, but the problem is to find the relevant information from a large collection of documents. Web search engines provide an efficient interface to the Web. In a typical search session, a user enters one or more keywords to a search engine, which we refer to as a *query*. The search engine then returns a ranked set of results in the descending order of their relevance to the user query. Often, there are millions of documents that match a user query and the task of ranking those documents according to their relevance is a challenging but an important one to a Web search engine [1].

Accurate ranking of search results is an important task for a Web search engine. If a search engine often ranks irrelevant results as the top hits, then users get dissatisfied with that search engine and will soon move to more competitive search engines. Adverts are a main source of income for search engines. If a search engine does not display relevant adverts to user queries, the users will not click on those adverts, resulting in reduced revenue to the search engine. Therefore, the problem of ranking search results in information retrieval systems have received much attention from both academia as well as from the industry. In particular, the Learning to Rank (LETOR) project by Microsoft Research[1], and the Yahoo! learning to rank challenge[2] are noteworthy initiatives.

[1]http://research.microsoft.com/en-us/um/beijing/projects/letor/
[2]http://learningtorankchallenge.yahoo.com

Ranking search results retrieved for a user query is a difficult problem because of several challenges. First, there are numerous factors a search engine must take into consideration when determining the rank of a search result such as the content of the document (i.e. web page) (i.e. whether the document contains the words in the query), the structure of the document (i.e. whether the query appears in the title of the document, its body or in an anchor text pointing to the document), link structure (i.e. the number of in-bound and out-bound links to the document), authority (i.e. encyclopedic resources edited by numerous authors vs. personal blogs), and novelty (i.e. how often does the content in a document is revised and the last updated time). The exact combination of those heterogenous factors that produces the best possible ranking for a set of documents is not obvious. Second, the ranking function must be simple enough to compute and scalable to be used in a Web search engine. If a search engine takes a long time to rank the retrieved set of documents, it might lead to user dissatisfaction. Third, any approach that learns a ranking function for information retrieval must be able to efficiently learn from large datasets. Search engines record the search sessions such as the query entered by the users and the search results they visit subsequently. This process enables us to collect large datasets that can be used as training data to learn ranking functions that assign higher ranks to documents that are visited by users. For example, LETOR dataset contains over 25 million rank information annotated documents for numerous queries entered by users in real-world search sessions.

We propose a method to learn a *non-linear* ranking function for information retrieval using Probabilistic Model Building Genetic Programming (PMBGP). We refer to our proposed method as *Rank-PMBGP*. PMBGP is an extension of genetic programming (GP) using probabilistic models. Although there have been several approaches proposed in prior work that can learn a *linear* ranking function, to the best of our knowledge, *Rank-PMBGP* is the first approach to learn non-linear ranking functions for information retrieval using evolutionary algorithms. The ability to learn non-linear ranking functions is particularly important for information retrieval. For example, consider the combination of the two features: the number of occurrences of the query in the document, and the authority of the document. If the number of occurrences of a query in a document is high, it indicates that the document is relevant to the query. However, sometimes spam web sites include popular queries to attract web traffic. Therefore, the

number of occurrences of a query in a document is a good indicator of relevance only when the authority of the document is high. Such conditional dependencies among factors that influence the rank of a document can be captured only by non-linear ranking functions. Consequently, non-linear ranking function learning methods have shown superior performance over methods that are limited to learning only linear ranking functions [2].

Our contributions in this paper can be summarized as follows.

- We propose a method to learn non-linear ranking functions for information retrieval using probabilistic model building genetic programming.
- We evaluate the proposed method using a standard benchmark dataset that was previously proposed for evaluating learning to rank methods for information retrieval. Our experimental results show that the proposed method significantly outperforms a baseline method that uses genetic programming to learn non-linear ranking functions. Moreover, the performance reported by the proposed method is comparable to that of the state-of-the-art learning to rank methods that use evolutionary algorithms. However, unlike prior work based on evolutionary algorithms, our method can learn non-linear combinations of features.

The remainder of this paper is organized as follows. In Section II-A, we present the learning to rank problem in the context of information retrieval. We then briefly describe the foundations upon which our proposed method is established: genetic programming (in Section II-C), and probabilistic model building genetic programming (in Section II-D). Next, in Section III, we introduce Rank-PMBGP, the proposed method for learning a non-linear ranking function for information retrieval. The benchmark dataset, LETOR, that we use to train and evaluate the proposed method is detailed in Section IV-A. In Section II-B, we introudce Mean Average Precision (MAP) and Normalized Discounted Cumulative Gain (NDCG) – the standard evaluation measures used in the information retrieval community to compare the performance of different rank learning methods. We compare the proposed method against numerous previously proposed methods for learning to rank in information retrieval in Section V. Finally, we discuss the relevant prior work in Section V and conclude this paper.

II. Background

A. Learning to rank

The problem of learning a function that can assign ranks for a set of items arises in numerous contexts. For example, in a web search scenario, we must rank the set of documents (i.e. web pages) according to their relevance to the query entered by a user. As a result of the increasing importance of web search engines as an efficient interface to the vast amounts of information available on the Web, the problem of learning to rank has received special attention in the information retrieval community. There are two main stages involved in learning to rank for information retrieval: a) learning a ranking function using a labeled dataset (i.e. *training stage*), b) applying the learnt ranking function to assign ranks to a set of documents retrieved for a user-query (i.e. *ranking stage*).

In the training stage, a ranking function learning algorithm is presented with a ranked list of documents retrieved for a particular query. To formally define the learning problem, let us denote the set of queries by $\mathcal{Q} = \{q_1, q_2, \ldots, q_{|\mathcal{Q}|}\}$, in which we use the notation, $|\mathcal{Q}|$, to represent the number of elements (i.e. cardinality) in the set $\mathcal{Q}$. Likewise, we represent the set of documents by $\mathcal{D} = \{d_1, \ldots, d_{|\mathcal{D}|}\}$. Then, the training dataset can be represented as a set of query-document pairs, $(q_i, d_j) \in \mathcal{Q} \times \mathcal{D}$, in which each query-document pair, (q_i, d_j), is assigned with a relevance judgement, $y(q_i, d_j)$, indicating the relevance of the document d_j to the query q_i. The relevance judgement $y(q_i, d_j)$ can be expressed in several ways. The simplest approach is to indicate a binary relevance $y(q_i, d_j) \in \{0, 1\}$, depending on whether the document d_j is relevant to the query q_i (i.e. $y(q_i, d_j) = 1$), or irrelevant (i.e. $y(q_i, d_j) = 0$). Alternatively, one can assign a real-valued relevance judgement that can be used to induce a total ranking among the documents retrieved for a particular query.

Web search engines record each search session in a log file called the search log to obtain relevance judgements. *Clickthorugh* [3] is popular method to easily collect a large collection of relevance judgements. In clickthrough approach, a search engine records the urls that were clicked by a user among all the urls displayed to that user for a particular query. For example, let us assume three documents d_1, d_2, and d_3 is shown to a user in that order as the list of search results for a particular query q. Moreover, let us assume that the user did not click on the first document d_1 and instead clicked on the second document d_2. This action is recorded by the search engine before it directs the user to d_2. In the clickthrough approach to obtaining relevance judgements, we assume that the document d_2 is more relevant to the query q than the document d_1. Therefore, a relevance score is assigned such that $y(q, d_2) > y(q, d_1)$. However, no relevance judgements are inferred for the documents such as d_3 that are not clicked on and appears below lowest ranked document that is clicked by a user for a particular query (i.e. d_2 in this example). Because web search engines are used by millions of users resulting in large search logs, the clickthrough approach provides a cheap solution to obtain large training datasets that can be used to train ranking function learning algorithms.

The goal in learning to rank is to learn a function $f(q, d)$ that assigns a ranking score indicating the degree of relevance of a document d for a query q. First a query-document pair (q, d) is represented by a feature vector $\boldsymbol{\phi}(q, d)$. Numerous feature have been proposed in prior work in learning to rank such as the number of occurrences of the query q in the title or the body of the document d, and PageRank [4] of d. In Section IV-A, we detail the numerous features that are used for training by our proposed method. Most prior work on learning to rank model the ranking function $f(q, d)$ as a linearly weighted combination of the features in $\boldsymbol{\phi}(q, d)$ as

follows,

$$f(q,d) = \boldsymbol{w}^\top \boldsymbol{\phi}(q,d). \quad (1)$$

Here, $\boldsymbol{w}$ is a vector representing the weight associated with a particular feature in $\boldsymbol{\phi}(q,d)$. We refer to the ranking function given by Equation 1 as a *linear* ranking function because it does not consider non-linear combinations of features in $\boldsymbol{\phi}(q,d)$. In contrast, our proposed method learns a non-linear combinations of features, thus having a greater expressiveness. Specifically, we model the problem of learning to rank as a search problem, where we must find the optimal non-linear combination of features representing a query-document pair that assigns ranking scores similar to the scores assigned in the training data.

Before we explain the search algorithm we use to find the optimal non-linear combination of features that assigns ranking scores for information retrieval, we must first devise a method to evaluate the *fitness* of a given combination of features. Let us denote the ranking function corresponding to some non-linear combination of features in $\boldsymbol{\phi}(q,d)$ by $f(q,d)$. Then we can use $f(q,d)$ to assign ranking scores to the set of documents $\mathcal{D}(q)$ retrieved for the query q. Next, we can compare the list of ranked documents produced by $f(q,d)$ against the ranks assigned to the documents in $\mathcal{D}(q)$ in the training dataset. The degree to which the two lists of ranks agree is an indicator of the fitness of the combination of features we use to define $f(q,d)$. Next, in Section II-B, we introduce the evaluation measures that are popularly used in the information retrieval community to asses the agreement between a list of ranked documents by a ranking function and that by a human annotator.

B. Evaluation Measures

To evaluate a ranking produced by an algorithm for a set of documents retrieved for a particular query, we can compare it against the ranking induced by the scores assigned by a human annotator for those documents. Precision at position n ($P@n$), Mean Average Precision (MAP), and normalized discounted cumulative gain (NDCG) are three widely used rank evaluation measures in the information retrieval community. Both those evaluation measures are in the range $[0,1]$, where a method that produces the exact ranking as in the gold standard achieves the score of 1. Next, we describe each of those evaluation measures in detail.

Precision at rank n ($P@n$) [5] measure is defined as the proportion of the relevant documents among the top n-ranked documents,

$$P@n = \frac{\text{No. of relevant docs in top } n \text{ results}}{n}. \quad (2)$$

Average precision averages the $P@n$ at over different n values to produce a single measure for a given query as follows,

$$AP = \frac{\sum_{n=1}^{N}(P@n \times rel(n))}{\text{No. of relevant docs for this query}}. \quad (3)$$

Here, N is the number of retrieved documents, and $rel(n)$ is a binary function that returns the value 1 if the n-th ranked document is relevant to the query under consideration and 0 otherwise. Mean average precision (MAP) is computed as the average of AP over all queries in the dataset.

NDCG considers the reciprocal of the logarithm of the rank assigned to relevant documents. For a ranked list of documents retrieved for a query, NDCG value at position n, $NDCG@n$, is computed as follows,

$$NDCG@n = Z_n \sum_{j=1}^{n} \frac{2^{r(j)} - 1}{\log(1+j)}. \quad (4)$$

Here, $r(j)$ is the rating of the j-th document in the ranked list, and the normalization constant Z_n is chosen such that a perfectly ranked list would obtain an NDCG@n score of 1. Specifically, it is given by,

$$Z_n = \frac{1}{\sum_{j=1}^{n} \frac{1}{\log(1+j)}}. \quad (5)$$

We use Mean Average Precision (MAP) as the fitness function because it provides a single value that we can use to determine the fitness of the ranking function $f(q,d)$. All three measures, MAP, NDCG, and P@n, are used to evaluate the performance of the final learnt ranking function.

C. Genetic Programming

Genetic Programming (GP) [6] is a widely used and successful method to optimize non-linear combinations of features represented by tree structures. First, GP randomly generates M tree structures each corresponding to some non-linear combination of features. In subsequent iterations, individual tree structures are evaluated using some fitness function and the top $M \times P_e$ individuals with the highest fitness values are retained to the next generation. Here, P_e denotes the elite rate that determines the number of individuals retained for the next generation. From those retained individuals, GP randomly selects N individuals and performs *mutation* and *cross-over* to produce offsprings. Mutation replaces a subtree in an individuals with a different subtree, whereas crossover partitions individuals into constituent subtrees and inter-change subtrees between different individuals. Over the generations, subtree structures that correspond to salient feature combinations are retained in the population, which are referred to as *building blocks*. The above-mentioned procedure is repeated until some pre-defined termination criterion is met. The pseudo code for GP in shown in Algorithm 1. Therein, P_g denotes the population (i.e. set of individuals) at the g-th generation, and S_g is an elite individual selected for reproduction at the g-th generation.

D. Probabilistic Model Building GP

Probabilistic model building GPs (PMBGP) are a variant of Estimation of Distribution Algorithms (EDA) [7], which are generative model based evolutionary computation algorithms, for optimizing tree structures. PMBGPs estimate probability distributions using individuals that have the highest fitness values. New individuals are generated by sampling from the estimated probability distribution. PMBGPs can be categorized

Algorithm 1 Genetic Programming(GP)

1: $g \leftarrow 0$
2: $P_g \leftarrow$ Initialize M individuals
3: Evaluate P_g
4: **while** terminate criterion is $False$ **do**
5: $g \leftarrow g + 1$
6: $S_g \leftarrow$ Select N $(N \leq M)$ superior individuals
7: $P_g \leftarrow$ Copy $M * P_e$ elite individuals
8: $P_g \leftarrow$ Generate $M(1 - P_e)$ individuals from S_g, using crossover or mutation
9: Evaluate P_g
10: **end while**

into two groups. The first type of PMBGPs exploit Probabilistic Context Free Grammar (PCFG) to learn subtree building blocks [8], The second type of PMBGPs use prototype trees, which extends EDAs proposed for one dimensional arrays to handle tree structures [9]. The prototype tree-based approach is essentially equivalent to EDAs. This property of prototype tree-based PMBGPs enables us to incorporate techniques devised in the field of EDAs. For example, sampling of individuals can be done using Loopy Belief Propagation (LBP) [10]. Using the notation we used in Algorithm 1, we show the pseudo code for PMBGP in Algorithm 2. Although PMBGPs have shown better performance than GPs in benchmark problems [11], comparatively to GPs, PMBGPs are yet to be applied to large-scale real-world problems such as the learning to rank for information retrieval which we study in this paper.

Algorithm 2 PMBGP

1: $g \leftarrow 0$
2: $P_g \leftarrow$ Initialize M individuals
3: Evaluate P_g
4: **while** terminate criterion is $False$ **do**
5: $g \leftarrow g + 1$
6: $S_g \leftarrow$ Select N $(N \leq M)$ superior individuals
7: $D_g \leftarrow$ Estimate distribution from S_g
8: $P_g \leftarrow$ Copy $M * P_e$ elite individuals
9: $P_g \leftarrow$ Sampling $M(1 - P_e)$ individuals from D_g
10: Evaluate P_g
11: **end while**

III. Proposed Non-linear Rank Learning Method: Rank-PMBGP

We propose Rank-PMBGP, a method to learn non-linear ranking functions for information retrieval using PMBGP. Specifically, we use Program Optimization with Linkage Estimation (POLE) [12] as the PMBGP method. POLE is a prototype tree based PMBGP method, which first translates tree structures to one-dimensional arrays and then apply EDAs to those arrays. POLE estimates mulivariate dependencies between nodes using Bayesian networks. POLE uses Expanded Parse Trees (EPT) [13] to represent the chromsomes thereby reducing the number of symbols in the tree trunk. EPT pushes terminal nodes on trunk to the leaf nodes using a special function node L. Given a list of arguments as the input, the function L returns the first argument. Therefore, in POLE, the symbols on trunk are limited to functions. This property of POLE simplifies the task of learning a Bayesian network. This is particularly important in learning to rank for information retrieval because the number of terminal symbols (features) in our task is much higher than that in benchmark problems such as MAX [14] or Royal Tree [15] for which PMBGPs have been applied so far. The types of nodes used by Rank-PMBGP are summarized in Table I. Rank-PMBGP considers non-linear combinations of features because it uses multiplication (shown by * in Table I) as a function node. We use MAP as fitness function in Rank-PMBGP.

TABLE I
NODE NAME AND MEANING OF THE PROPOSED METHOD

node name	node type	meaning
S_f	function (trunk)	the set of function nodes {+,-,*}
S_v	terminal (leaf)	the set of variable (feature) nodes
S_c	terminal (leaf)	the set of constant nodes

Rank-PMBGP consists of five steps as shown below.

Step 1: **Input**
Set parameters of PMBGP. Obtain train, validation, and test data.

Step 2: **Training by PMBGP**
Train a non-linear ranking function by PMBGP. MAP computed using train data is considered as the fitness function.

Step 3: **Validation**
Evaluate the individuals that exist in the population at the final generation by the MAP computed over the validation data.

Step 4: **Output**
Select the individual that corresponds to the maximum value of the following sum:
MAP on final generation at training +
MAP on validation data.

Step 5: **Ranking**
Rank the documents in the test dataset using the learnt non-linear ranking function.

IV. Experiments

A. Dataset

We use the LETOR (version 2.0) benchmark dataset [16] that has been widely used in prior work on learning to rank for information retrieval. The LETOR version 2.0 consists of TD2003 and TD2004 datasets, which were part of the topic distillation task of the Text REtrieval Conference (TREC) in year 2003 and 2004. TD2003 dataset contains 50 queries and TD2004 dataset contains 75 queries. The document collection contains $1,053,110$ documents together with $11,164,829$ hyperlinks and is based on a January, 2002 crawl of the .gov domain. Topic distillation aims to find a list of documents relevant to a particular topic. The TREC committee provides judgements for the topic distillation task. For each query in TD2003 and TD2004 datasets, there are about $1,000$ documents listed. Each query-document pair is given a binary

TABLE II
FEATURES IN THE LETOR TD2003 AND TD2004 DATASETS.

Category	Feature	No. of features
Content (low-level)	tf [5]	4
	idf [5]	4
	dl [5]	4
	tfidf [5]	4
Content (high-level)	BM25 [17]	4
	LMIR [18]	9
Hyperlink	PageRank [4]	1
	Topical PageRank [21]	1
	HITS [19]	2
	Topical HITS [21]	2
	HostRank [20]	1
Hybrid	Hyperlink-base relevance propagation [22]	6
	Sitemap-based relevance propagation [23]	2
Total		44

judgement indicating whether a document is relevant or non-relevant for a particular query.

A query-document pair in the LETOR dataset is represented using a 44 features as shown in Table II. The features include numerous ranking heuristics popularly used in the information retrieval community to rank a list of retrieved documents. The set of features includes low-level features such as, term frequency (tf), inverse document frequency (idf), document length (dl) combinations of low-level features such as tf*idf [5], as well as high-level features such as BM25 [17] and LMIR [18]. Hyperlink structure provides useful clues about the relevancy of a web page. Consequently, several features are computed using the hyperlink information in LETOR datasets such as PageRank [4], HITS [19], HostRank [20], topical PageRank and topical HITS [21]. Following the standard practice, all features are normalized to $[0, 1]$ range prior to training. For the TD2003 and TD2004 datasets, we define two values of ratings 0 and 1 respectively corresponding to relevant and non-relevant documents in order to compute NDCG scores. In our evaluations, we report the average values taken over all the queries in a dataset as $P@n$ and $NDCG@n$.

B. Experimental Settings

The parameters of Rank-PMBGP (POLE) is described in Table III. Individuals in POLE are intialized by GROW, where P_F is the selection rate of functions. POLE uses truncate selection, where $M * P_s$ individuals are selected and used for construction of Bayesian networks and estimation of parameters.

C. Results

First, we examine the effect of feature selection. We consider that proper limitation of the search space by feature selection is effective because the search space of non-linear functions with 44 features in TD2003 and TD2004 are too vast to optimise. Due to the time limitation, we examine the effect of feature selection in only TD2003. We show the condition with all features and selected features in Table IV and V.

TABLE III
PARAMETER VALUES FOR THE PROPOSED METHOD (RANK-PMBGP)

Prameters	Meaning	value
P_s	Selection Rate	if population size is larger than 5000 use 0.05 otherwise use 0.2
P_e	Elite Rate	if population size is larger than 5000 use 1 otherwise use 0.005
P_F	Functional Selection Rate	0.9

Feature selection is done, using the knowledge of existing method, RankDE [24]. Features whose absolute value of optimised weight in RankDE is larger than 2 are selected and showed in Table V. The proposed method and a baseline for comparison, non-linear GP, which is an extension of RankGP to non-linear, are examined. We conduct experiments 10 times and show the effect of feature selection in Fig. 1. The number of fitness evaluation is 60000 (population size is 600, and the maximum generation is 100). Fig. 1 clearly illustrates that feature selection improves MAP in both GP and PMBGP.

TABLE IV
CONDITION OF SYMBOLS 1: ALL-SYMBOLS

name	value
S_f	{+,-,*} (all function takes two arguments)
S_v	all 44 features in LETOR
S_c	{ 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0 }
The number of terminal symbols	54
depth limitation	8

TABLE V
CONDITION OF SYMBOLS 2: FEW-SYMBOLS (SELECTED)

name	value
S_f	{+,-,*} (all function takes two arguments)
S_v	11 features (id : name) 5: dl of URL 7: HITS hub 8: HostRank 9: idf of body 10: idf of anchor 11: idf of title 12: idf of URL 18: LMIR. JM of anchor 21: LMIR. DIR of extracted title 23: LMIR. ABS of title 39: Hyperlink base score propagation (weighted in-link) }
S_c	$\{0.2, 0.4, 0.6, 0.8, 1.0\}$
The number of terminal symbols	16
depth limitation	8

Second, we investigate the trade off between the learning time, the number of evaluations, and the quality of the output. We conduct experiments on three different condition of the number of evaluations: 60000 (population size=600, maximum generation=100), 250000 (population size=5000, maximum generation=50), 1000000 (population size=10000, maximum

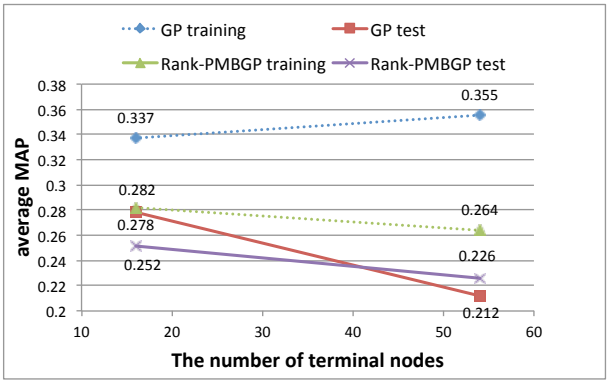


Fig. 1. The effect of feature selection in TD2003

generation=100). We visualize the results in Fig. 2. These results show the same information that PMBGP becomes better than GP as the number of evaluations increases in both TD2003 and TD2004. It is worth to note that Rank-PMBGP behaves differently on different datasets. In TD2003, MAP of Rank-PMBGP improves logarithmly according as the number of evaluations increases while that improves more sharply in TD2004.

Finally, we compare the performance of the proposed method and exisiting methods on TD2003 in Table VII. Description of each of those algorithms is the following.

- RankSVM
  This is an application of support vector machine, which is a well known binary classifier based on margin maximization, to learning to rank. RankSVM is one of the most popular pairwise approach, and the performance of RankSVM is provided as baseline of LETOR.
- RankBoost
  RankBoost combines a variety of ranking scores based on AdaBoost. The features in LETOR are employed as weak classifiers in this method. The performance of RankBoost is also provided as baseline of LETOR, as well as RankSVM.
- SwarmRank
  This method learns linear ranking functions and directly optimises MAP over given dataset, using particle swarm optimization (PSO).
- RankGP
  This is an application of GP to learning to rank. Linear ranking functions represented by tree structures evolves, optimising MAP directly.
- RankDE
  RankDE is a differential evolution (DE) based ranking algorithm. RankDE optimises wights of linear combinations of features. To our knowledge, RankDE is the best algorithm among existing ranking algorithm using evolutionary computation.
- GP (baseline)
  This is the baseline that we conducted experiments. This is an extension of RankGP for non-linear ranking functions.
- Rank-PMBGP (proposed)
  This is the proposed method using PMBGP.

GP in the present paper uses tournament selection and adopts adaptive mutation rate tuning method (AMRT) [25], as well as RankGP [26]. AMRT increases mutation rate and decreases crossover rate when population is likely to converge.

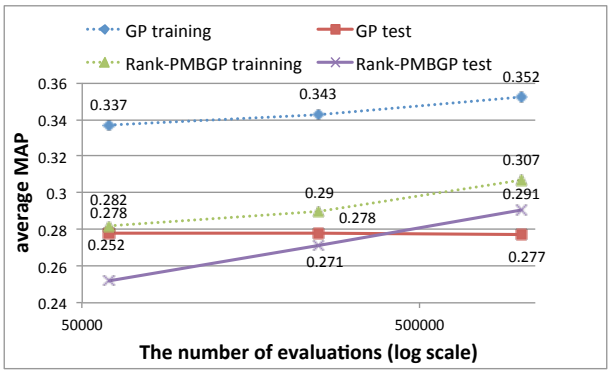
TABLE VI
PARAMETERS OF GP

Prameters	Meaning	value
P_e	Elitist reproduction Rate	only 1 inividual
P_c	crosover rate	start:0.95 change dynamically using AMRT
P_m	mutation rate	start:0.05 change dynamically using AMRT
size_t	tournament size	5
P_F	Functional Selection Rate	0.9

Although RankDE is better than Rank-PMBGP, Rank-PMBGP overwhelms RankSVM, RankBoost, SwarmRank, RankGP and baseline (GP) with all measures: MAP, P@n and NDCG. Especially, the improvements of MAP reported by Rank-PMBGP over RankSVM, RankBoost, SwarmRank and baseline (GP) are statistically significant at 5 % significant level on paired t-test.

V. RELATED WORK

Learning to rank is divided into three types:pointwise approach, pairwise approach and listwise approach. The pointwise approach [27], [28] deals with each query-document pair independently during entire training and ranking. Because the pointwise approach dismiss the relative preferences between query-document pairs for the same query, its performance is the worst in three types. In contrast, the pairwise approach [29]–[32] considers preference between two documents d_j and d_k retrieved for the same query q_i and creates pairwise constraints. The pairwise approach only considers preference between two documents and dismiss that between other documents for the same query. This fundamental problem sometimes causes awful results at test time because learning to rank for information retrieval aims to estimate the total ordering of the entire documents but not partial orderings between two documents. The listwise approach [33]–[36] gets over drawbacks mentioned about the pointwise and pairwise approaches, considering the entire set of documents retrieved for the same query. It is known that pairwise methods have better performance than pointwise methods, and listwise methods have still better performance than pairwise methods. However, it is not true that listwise is the best approach of learning to rank because there are trade off between performance and trouble to create training data. For example, pairwise methods use training data in partial order but listwise methods need training data in total order. It is more laborious to create training data in total order than in partial order. The present paper



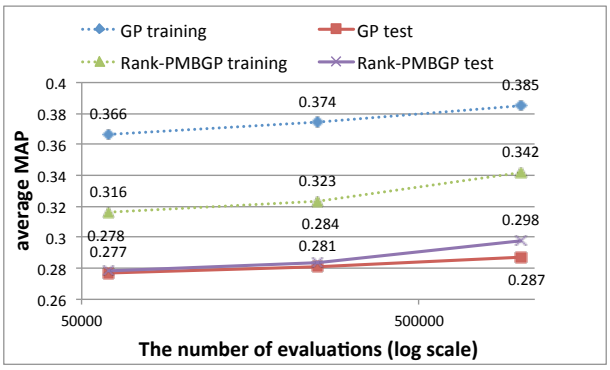


Fig. 2. the trade off between the number of evaluations and MAP (Left: TD2003, Right: TD2004)

TABLE VII
RANKING PERFORMANCE ON THE TD2003

Method	RankSVM	RankBoost	SwarmRank	RankGP	RankDE	non-linear GP baseline	Rank-PMBGP (Proposed)
MAP	0.256	0.212	0.209	0.283	0.339	0.277	0.291
P@1	0.420	0.260	0.453	0.520	0.600	0.528	0.548
P@2	0.350	0.270	0.330	0.420	0.400	0.444	0.468
P@3	0.340	0.240	0.269	0.370	0.333	0.380	0.405
P@4	0.300	0.230	0.223	0.330	0.300	0.336	0.352
P@5	0.264	0.220	0.207	0.280	0.280	0.294	0.294
P@6	0.243	0.210	0.188	0.270	0.250	0.265	0.267
P@7	0.234	0.211	0.185	0.250	0.243	0.250	0.247
P@8	0.233	0.193	0.173	0.240	0.237	0.229	0.232
P@9	0.218	0.182	0.164	0.230	0.222	0.214	0.215
P@10	0.206	0.178	0.151	0.220	0.210	0.199	0.204
NDCG@1	0.420	0.260	0.453	0.520	0.600	0.528	0.548
NDCG@2	0.370	0.280	0.343	0.450	0.445	0.463	0.486
NDCG@3	0.379	0.270	0.307	0.420	0.388	0.413	0.438
NDCG@4	0.363	0.272	0.284	0.390	0.356	0.378	0.396
NDCG@5	0.347	0.279	0.278	0.380	0.336	0.345	0.353
NDCG@6	0.341	0.280	0.271	0.370	0.310	0.321	0.329
NDCG@7	0.340	0.287	0.273	0.360	0.300	0.306	0.311
NDCG@8	0.345	0.282	0.270	0.350	0.292	0.288	0.295
NDCG@9	0.342	0.282	0.267	0.350	0.279	0.275	0.280
NDCG@10	0.341	0.285	0.263	0.350	0.267	0.261	0.269

proposes pairwise based methods, which is nicely balanced in performance and trouble to prepare training data.

Some GP based learning to rank methods have been already proposed. RankGP [26] learns linear ranking function using GP. RankGP regards linear ranking functions as individuals and use adaptive mutation training method [25]. [37] proposes non-linear ranking function optimization by GP. [37] creates non-linear ranking function better than BM25 using only simple features:term frequency(tf), inverse document frequency(idf), document length(dl) and so on. Today's learning to rank regard functions combined simple features as feature, and optimises ranking function using those combined features and simple features. We propose non-linear learning to rank by GP and PMBGP using those combined features and simple features, and therefore this paper is quite different from [37], which only uses simple features. Other evolutionary computation based methods have been also proposed. Those methods are described in Section .

One of the most successful non-linear optimisation for learning to rank is GBDT (Gradient Boosted Decision Tree) [38], [39]. GBDT is boosting using regression trees and it appears that GBDT based learning to rank tends to converge local optima more frequently than evolutionary computing based learning to rank because GBDT updates its models using local gradient. Yahoo! Learning to Rank Challenge [2] employs GBDT as a baseline. We can not compare the performance of the proposed method and that of GBDT because there is no result of GBDT on LETOR 2.0 and we do not have too much time to experiment GBDT on LETOR 2.0 The comparison between the proposed method and GBDT is a future work.

VI. CONCLUSION

This paper proposed Rank-PMBGP, which is the novel non-linear ranking function optimization using a PMBGP, POLE. It is an advantage over existing ranking algorithms for infor-

mation retrieval that Rank-PMBGP directly optimizes MAP without requiring any convex approximations. We evaluated the proposed method using the standard benchmark datasets, LETOR. In our experiments, the proposed method defeats a non-linear baseline approach using GP and competes with the state-of-the-art linear ranking methods using evolutionary algorithms.

Due to the time limitation, some tasks are left to future work. First, the performance of the proposed method seems to improve if more the number of evaluations is given. Second, we show that feature selection using knowledge of the existing method improves performances but more sophisticated feature selection might influence the performance significantly. Finally, although, in the small number of evaluations, GP competes or overwhelms PMBGP, PMBGP becomes better as the number of evaluations increases. Therefore, hybrid approach based on migration model using GP and PMBGP or switching from GP to PMBGP seems to works well. Combination of GP and PMBGP in learning to rank is future work.

REFERENCES

[1] C. D. Manning, P. Raghavan, and H. Schutze, *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[2] O. Chapelle and Y. Chang, "Yahoo! learning to rank challenge overview." *Journal of Machine Learning Research - Proceedings Track*, vol. 14, pp. 1–24, 2011. [Online]. Available: http://dblp.uni-trier.de/db/journals/jmlr/jmlrp14.html#ChapelleC11

[3] T. Joachims, "Optimizing search engines using clickthrough data," in *KDD'02*, 2002, pp. 133–142.

[4] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web," Stanford InfoLab, Tech. Rep. SIDL-WP-1999-0120, November 1999.

[5] R. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval*. Addison-Wesley, 1999.

[6] J. R. Koza, *Genetic Programming : On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

[7] M. Hauschild and M. Pelikan, "An introduction and survey of estimation of distribution algorithms," *Swarm and Evolutionary Computation*, vol. 1, no. 3, pp. 111 – 128, 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2210650211000435

[8] P. A. N. Bosman and E. D. de Jong, "Grammar transformations in an EDA for genetic programming," *GECCO 2004 Workshop Proceedings*, 2004.

[9] R. P. Sałustowicz and J. Schmidhuber, "Probabilistic incremental program evolution," *Evolutionary Computation*, vol. 5, pp. 123–141, 1997.

[10] H. Sato, Y. Hasegawa, D. Bollegala, and H. Iba, "Probabilistic model building GP with belief propagation," in *Proceedings of the 2012 IEEE Congress on Evolutionary Computation*, X. Li, Ed., Brisbane, Australia, 10-15 Jun. 2012, pp. 2089–2096.

[11] K. Kim, B. McKay, and D. Punithan, "Sampling bias in estimation of distribution algorithms for genetic programming using prototype trees," in *Proceedings of the 11th Pacific Rim international conference on Trends in artificial intelligence*, ser. PRICAI'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 100–111.

[12] Y. Hasegawa and H. Iba, "A Bayesian network approach to program generation," *IEEE Transactions on Evolutionary Computation*, vol. Vol. 12, NO. 6, pp. 750–764, 2008.

[13] M. Wineberg and F. Oppacher, "A representation scheme to perform program induction in a canonical genetic algorithm," *Proceedings of the International Conference on Evolutionary Computation. The Third Conference on Parallel Problem Solving from Nature*, pp. 292 – 301, 1994.

[14] C. Gathercole, P. Ross, and S. Bridge, "An adverse interaction between the crossover operator and a restriction on tree depth," *in Proc. 1st Annu. Conf. Genetic Programming*, pp. 291–296, 1996.

[15] W. F. Punch, "How effective are multiple populations in genetic programming," *Genetic Programming 1998: Proceedings of the Third Annual Conference: John R. Koza and Wolfgang Banzhaf and Kumar Chellapilla and Kalyanmoy Deb and Marco Dorigo and David B. Fogel and Max H. Garzon and David E. Goldberg and Hitoshi Iba and Rick Riolo, Eds*, pp. 308–313 308–313, 1998.

[16] T. Qin, T.-Y. Liu, J. Xu, W. Xiong, and H. Li, "Letor: A benchmark collection for learning to rank for information retrieval," Microsoft Research Asia, Tech. Rep., 2007.

[17] S. E. Robertson, "Overview of the okapi projects," *Journal of Documentation*, vol. 53, no. 1, pp. 3 – 7, 1997.

[18] C. Zhai and J. Lafferty, "A study of smoothing methods for language models applied to information retrieval," *ACM Transactions on Information Systems*, vol. 22, no. 2, pp. 179 – 214, 2004.

[19] J. M. Kleinberg, "Authoritative sources in a hyperlinked environment," *Journal of the ACM*, vol. 46, no. 5, pp. 604 – 632, 1999.

[20] G.-R. Xue, Q. Yang, H.-J. Zeng, Y. Yu, and Z. Chen, "Exploiting the hierarchical structure for link analysis," in *SIGIR'05*, 2005, pp. 186 – 193.

[21] L. Nie, B. D. Davison, and X. Qi, "Topical link analysis for web search," in *SIGIR'06*, 2006, pp. 91 – 98.

[22] A. Shakery and C. Zhai, "Relevance propagation for topic distillation uiuc trec 2003 web track experiments," in *TREC'03*, 2003.

[23] T. Qin, T.-Y. Liu, X.-D. Zhang, Z. Chen, and W.-Y. Ma, "A study of relevance propagation for web search," in *SIGIR'05*, 2005, pp. 408 – 415.

[24] D. Bollegala, N. Noman, and H. Iba, "Rankde: learning a ranking function for information retrieval using differential evolution," in *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, ser. GECCO '11. New York, NY, USA: ACM, 2011, pp. 1771–1778. [Online]. Available: http://doi.acm.org/10.1145/2001576.2001814

[25] J.-Y. Lin, H.-R. Ke, B.-C. Chien, and W.-P. Yang, "Designing a classifier by a layered multi-population genetic programming approach," *Pattern Recogn.*, vol. 40, no. 8, pp. 2211–2225, Aug. 2007. [Online]. Available: http://dx.doi.org/10.1016/j.patcog.2007.01.003

[26] J. Y. Yeh, J. Y. Lin, H. R. Ke, and W. P. Yang, "Learning to Rank for Information Retrieval Using Genetic Programming," in *SIGIR 2007 workshop: Learning to Rank for Information Retrieval*, T. Joachims, H. Li, T. Y. Liu, and C. Zhai, Eds., Jul. 2007. [Online]. Available: http://jenyuan.yeh.googlepages.com/jyyeh-LR4IR07.pdf

[27] R. Herbrich, T. Graepel, and K. Obermayer, "Large margin rank boundaries for ordinal regression," *Advances in Large Margin Classifiers*, pp. 115 – 132, 2000.

[28] K. Crammer and Y. Singer, "Pranking with ranking," in *NIPS'01*, 2001.

[29] R. Herbrich, T. Graepel, and K. Obermayer, "Support vector learning for ordinal regression," in *ICANN'99*, 1999, pp. 97 – 102.

[30] Y. Freund, R. Iyer, R. E. Schapire, and Y. Singer, "An efficient boosting algorithm for combining preferences," *Journal of Machine Learning Research*, vol. 4, pp. 933 – 969, 2003.

[31] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender, "Learning to rank using gradient descent," in *ICML 2005*, 2005, pp. 89–96.

[32] M.-F. Tsai, T.-Y. Liu, T. Qin, H.-H. Chen, and W.-Y. Ma, "Frank: a ranking method with fidelity loss," in *SIGIR'07*, 2007, pp. 383 – 390.

[33] T. Qin, X.-D. Zhang, M.-F. Tsai, D.-S. Wang, T.-Y. Liu, and H. Li, "Query-level loss functions for information retrieval," *Information Processing and Management*, 2007.

[34] Z. Cao, T. Qin, T.-Y. Liu, M.-F. Tsai, and H. Li, "Learning to rank: From pairwise approach to listwise approach," in *ICML 2007*, 2007, pp. 129–136.

[35] Y. Lan, T.-Y. Liu, Z. Ma, and H. Li, "Generalization analysis of listwise learning-to-rank algorithms," in *ICML 2009*, 2009, pp. 557–584.

[36] F. Xia, T.-Y. Liu, J. Wang, W. Zhang, and H. Li, "Listwise approach to learning to rank: theory and algorithm," in *ICML 2008*, 2008, pp. 1192–1199.

[37] W. Fan, M. D. Gordon, and P. Pathak, "Genetic programming-based discovery of ranking functions for effective web search," *Journal of Management Information Systems*, vol. 21, no. 4, pp. 37–56, 2005.

[38] J. Friedman, T. Hastie, and R. Tibshirani, "Additive logistic regression: a statistical view of boosting," *Annals of Statistics*, vol. 28, p. 2000, 1998.

[39] J. H. Friedman, “Stochastic gradient boosting,” *Comput. Stat. Data Anal.*, vol. 38, no. 4, pp. 367–378, Feb. 2002. [Online]. Available: http://dx.doi.org/10.1016/S0167-9473(01)00065-2