# Fast Neural Ranking on Bipartite Graph Indices

Shulong Tan
Cognitive Computing Lab
Baidu Research
shulongtan@baidu.com

Weijie Zhao
Cognitive Computing Lab
Baidu Research
weijiezhao@baidu.com

Ping Li
Cognitive Computing Lab
Baidu Research
liping11@baidu.com

## ABSTRACT

Neural network based ranking has been widely adopted owing to its powerful capacity in modeling complex relationships (e.g., users and items, questions and answers). Online neural network ranking, i.e., the so called **fast neural ranking**, is considered a challenging task because neural network measures are in general non-convex and asymmetric. Traditional approximate near neighbor (ANN) search which typically focuses on metric ranking measures, is not applicable to these complex measures. To tackle this challenge, in this paper, we propose to construct **BipartitE Graph INdices (BEGIN)** for fast neural ranking. BEGIN contains two types of nodes: base/searching objects and sampled queries. The edges connecting these types of nodes are constructed via the neural network ranking measure. The proposed algorithm is a natural extension from traditional search on graph methods and is more suitable for fast neural ranking. Experiments demonstrate the effectiveness and efficiency of the proposed method.

**PVLDB Reference Format:**
Shulong Tan, Weijie Zhao, and Ping Li. Fast Neural Ranking on Bipartite Graph Indices. PVLDB, 15(4): 794 - 803, 2022.
doi:10.14778/3503585.3503589

## 1 INTRODUCTION

In recent years, neural network based ranking models play more and more vital roles in information retrieval, recommendation, question answering, i.e., [2, 4, 5, 9, 15, 16, 18, 47]. Figure 1 presents a typical example of neural network ranking measure – given a query and a base/searching object, the neural network style measure returns a ranking score. Neural networks are flexible in modeling complex relationships among different types of objects, such as queries and documents, users and items, or questions and answers. They are however often too time-consuming to be deployed for online ranking applications. Thus, neural network based ranking models are usually used in offline ranking or re-ranking on pre-produced small subsets [2, 4, 5]. Directly deploying neural network based measures for online ranking services (on large base sets) requires highly efficient searching indices, like **approximate near neighbor (ANN)** search methods [12, 20, 25, 33, 40, 50]. Fast and approximate ranking by neural network measures—so called **fast neural ranking** [35]—is considered challenging since these ranking measures are complex, usually non-convex and asymmetric. Traditional ANN methods are designed for simple ranking measures,

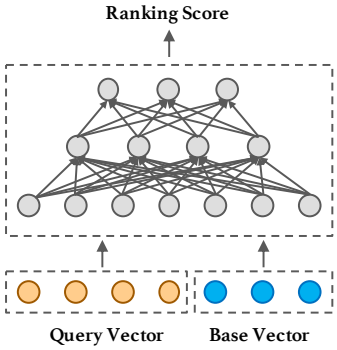


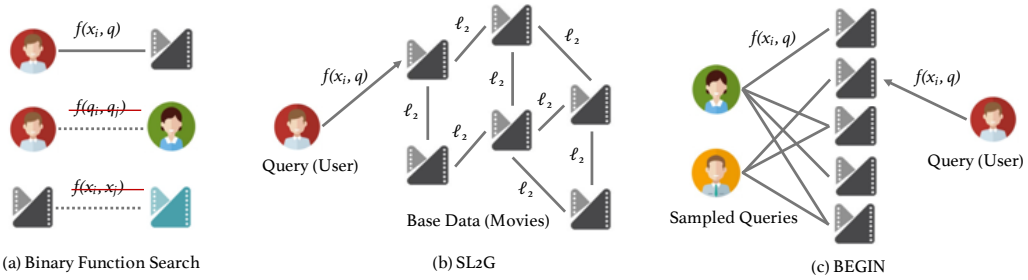

**Figure 1: An example of neural network ranking measures.**

such as $\ell_2$-distance, cosine similarity, inner product, or generalized min-max (GMM) kernel [28, 29], etc. It is oten not straightforward to extend ANN search methods for fast neural ranking scenarios.

Recently, [42] extends the definition of traditional ANN search to a generic setting, **Optimal Binary Function Search (OBFS)**: Let $X$ and $Y$ be subsets of Euclidean spaces, given a dataset $S = \{x_1, \ldots, x_n\} \subset X$ and a continuous binary function, $f : X \times Y \to \mathbb{R}$, given $q \in Y$, OBFS aims to find: $\arg\max_{x_i \in S} f(x_i, q)$. In this definition, ranking measures are considered as binary functions. There are no strong assumptions for the search function: metric or non-metric, linear or non-linear, convex or non-convex, symmetric or asymmetric. Beyond the definition, they also provide a solution for fast OBFS, called "Search on L2 Graph" (SL2G). They extend traditional graph based fast vector searching algorithms by constructing graph indices in $\ell_2$-distance but searching according to the focused binary function. Although SL2G works well for some binary functions, as shown in the paper, we find that it is problematic to approximate relationships among base data (e.g., item vectors) by $\ell_2$-distance. As defined in OBFS, we only have the ranking measure definition between base vectors (from $X$) and queries (from $Y$). There are usually no definitions for distances between base vectors (i.e., $(x_i, x_j)$), nor for distances between query vectors (i.e., $(q_i, q_j)$). It is not proper to model relationships among base data by $\ell_2$-distance. Figure 2 (a) and (b) present the challenges of OBFS and the SL2G solution. Besides, we design a baseline in the experiment, Sample-Ave (Section 4.2), which estimates zbase data distances by the average of $f$ value difference with a query sample set. It statistically estimates the relationships of base vectors. However, its performance is similar to SL2G and far from optimal. This is another evidence that shows estimated base data distances (i.e., $(x_i, x_j)$) cannot solve the OBFS problem well.

To solve fast OBFS in a better way, in this paper, we propose **BipartitE Graph INdices (BEGIN)**. Two types of nodes in the bipartite graph are sampled queries (i.e., $q$) and base data (i.e., $x$). The edges, which connect these two kinds of nodes, are constructed


Proceedings of the VLDB Endowment, Vol. 15, No. 4 ISSN 2150-8097.
doi:10.14778/3503585.3503589




(a) Binary Function Search (b) SL2G (c) BEGIN

**Figure 2: An example of the OBFS problem and solutions: we have collections of users and movies, and a binary function $f(x_i, q)$ learned on historical user-movie preference pairs. Given a user $q$ and a movie $x_i$, the binary function $f(x_i, q)$ predicts the ranking score of this pair—how the user may like the movie. (a) Challenges in adapting ANN algorithms to the fast OBFS problem. The binary function is defined on movie-user pairs—no user-user nor movie-movie distance/similarity is defined. However, traditional ANN methods, e.g., proximity graph, require the distance between base data vectors to construct an index. (b) SL2G exploits $\ell_2$-graph to approximate the binary function search space. (c) Our proposed solution–BEGIN–builds a bipartite graph that leverages users to bridge relations among movies. This allows us to apply graph-based search algorithms on fast OBFS without knowing the distance between users $f(q_i, q_j)$ or the relationship between movies $f(x_i, x_j)$.**

based on the ranking measure (i.e., $f(x, q)$). Intuitively, the ranking tasks, such as recommendation, naturally pose bipartite graphs. The two types of nodes correspond to users and items, respectively. The edges are constructed based on the interactions between users and items. Figure 2(c) shows an example of BEGIN. The graph construction procedure does not require to estimate distances between base vectors (as in SL2G) or between queries. Traditional graph indices (containing uni-type nodes), such as HNSW [33] and NSG [10] are designed for ranking according to metric measures. BEGIN is a natural extension of previous algorithms but for ranking by generic binary functions. To obtain better performance on the trade-off between searching efficiency and effectiveness, we propose a two-hop edge selection criterion and a fast search algorithm. In the experiments, we evaluate BEGIN on both synthetic and learned neural network measures. Our contributions are summarized as below:

- We introduce BEGIN–a novel graph indexing and searching algorithm–that bridges the gap between generic binary function search and common metric ranking on graph-based indices.
- We present a two-hop edge selection criterion to ensure that restricted number of edges in BEGIN are chosen effectively.
- We propose a fast searching algorithm that heuristically prunes out "bad" candidates, to optimize the query execution on the bipartite graph index.
- We investigate 4 methods to generate query samples when the known queries are not enough.
- We experimentally evaluate BEGIN on 2 synthetic measures and 3 recommendation neural networks, over 3 real-world recommendation datasets. Experiments show that BEGIN outperforms SL2G in most cases, under various evaluation measures.

## 2 RELATED WORK

In this section, we will first connect the generic ranking problem–Optimal Binary Function Search (OBFS)–with traditional approximate nearest neighbor (ANN) search and introduce some applications of OBFS. Then, in the methodology level, we will explain why previous fast search algorithms–designed for ANN search–are not applicable to the generic OBFS problem.

### 2.1 Fast Top-$K$ Search

Fast top-$K$ search has wide applications in modern information systems, such as top-$K$ recommender systems for e-commerce and link prediction for social networks. For online services, the search efficiency is as important as search effectiveness. Fast search algorithms try to construct some kind of index structures beforehand to speed up the online searching. Traditional fast search problems restrict the searching measures as metric measures (e.g., cosine similarity or $\ell_2$ distance) or some simple non-metric measures, such as inner product widely exploited in recommender systems. Fast search via metric measures is usually referred as approximate near neighbor (ANN) search [12, 20, 25, 33, 40, 50].

Optimal Binary Function Search (OBFS) is a generic definition for fast top-$K$ search. Theoretically, an OBFS task can choose any binary function as the search measure, linear or non-linear, metric or non-metric, convex or non-convex, symmetric and asymmetric. Traditional ANN search is a special instance of OBFS. Neural network based ranking measures [2, 8, 15, 32] are also examples of searching binary functions. Setting a user vector and an item vector as inputs, one can design any neural network structures as ranking measures. Parameters of the network are learned on training data but not fixed beforehand. These kinds of neural network based searching functions are usually non-convex, which are not studied by traditional ANN search work. These are lots of real cases of neural networks based searching measures, such as Multi-Layer Perceptron (MLP) and BERT-style ones [8, 18, 39, 44], which has wide applications in recommendation, ads ranking and retrieve based question answering [2, 4, 5, 9, 15, 16, 18, 47]. In this paper, we focus on the generic fast OBFS problem, especially how to speed up the searching process under neural network based measures, or called fast neural ranking.

### 2.2 Existing Fast Search Methods

There have been many algorithms for fast near neighbor search, such as hashing based methods [3, 12, 20, 30, 38, 40], quantization based methods [11, 24, 25, 30, 45], graph based indices [17, 22, 23, 33, 41, 46, 49, 50] and tree based methods [1, 6, 7]. For all those methods, each given query is compared with a subset instead of the

whole dataset or comparing by shorter codes, significantly reducing the time complexity while hopefully retaining high search recalls.

Most of these methodologies are designed for traditional ANN search. For example, one particular LSH algorithm is usually designed for one specific measure, such as "Gaussian random projections" for cosine similarity [3, 13, 30], "random Fourier features" for radial basis function kernel similarity [28, 31, 37], and "consistent weighted sampling" for min-max kernel similarity [21, 29, 34], etc.

They are infeasible to be extended for neural network based measures. Search on graph methods often claim that there are no constraints on searching measures (actually must be symmetric) [17, 33], most existing search on graph methods, however, mainly focus on searching by metric distances, with a few exceptions, e.g., [33, 36]. Although graph based indices for traditional ANN search are proved great superiority in the trade-off between search efficiency and effectiveness. It was shown that the performance is dramatically bad when applying them to generic ranking problems [42]. The reason is that these methods require the definition between base data to construct a proximity graph, which are not well defined under the setting of fast neural ranking.

[26] proposes CANTOR, which utilizes user coresets (like centroids of clusters) to improve the performance of traditional ANN search. User coresets are used as "short-cuts" in CANTOR: results for users/queries in coresets are recorded and will be returned directly as results for queries similar with them. Query samples in our method are exploited differently from the user corsets in CANTOR. We use query samples as "bridges" to connect base data. CANTOR works well on ANN search, based on the assumption: similar queries (in metric spaces) will have similar retrieval results. This assumption may not hold for the neural ranking problems: distances between queries are not well defined.

### 2.3 Why not SL2G?

As analyzed above, most previous fast vector ranking work focuses on simple ranking measures – mainly in metric measures and inner product– and is difficult to be extended for generic ranking measures. The exception is SL2G [42], which is designed for the generic fast OBFS problem. The **basic idea of SL2G** is:

(i) No matter what the given binary function $f$ is, they construct a Delaunay graph (or an approximate one) with respect to $\ell_2$ distance (which is defined on searching/base data $X$ and independent of queries) in the indexing step.
(ii) Then SL2G performs the greedy search on this graph by the binary function $f$ in the searching step.

The theoretical basis of SL2G is that, the performance of greedy search on $\ell_2$ graph is similar to optimizing OBFS by "coordinate" descent in Euclidean space. If the $f$ is smooth and the data are dense enough, SL2G will reach an approximate local optimum. The fast ranking problem is always a trade-off between effectiveness and efficiency. The theoretical analysis of SL2G guarantees the effectiveness but not the efficiency. Besides, SL2G utilizes $\ell_2$ distance to approximate the relationships among base data, which are undefined, as shown in Figure 2 (a) and (b). Neural network models only learn the relevance/distance function between queries and base data. We cannot determine relationships between two base data points. Consider base data points are movies in the recommendation system and queries are individual users. Some users may think the two movies are closely related because they are directed by the same director while another user may do not agree for they are in two different categories. Therefore, constructing graph indices based on estimated base data distances is problematic.

In this paper, we extend previous graph-based indices and propose bipartite graph indices for generic OBFS problems, with two types of vertices – base data and sample queries. With the proposed fast ranking algorithm, lots of generic searching measures can be applied for online ranking services, say those neural network based measures [8, 18, 39, 44, 47]. More advanced semantic information will be captured in the ranking/searching procedure and the user experience would be improved greatly.

Although the proposed algorithm is in sub-linear complexity, it is still not applicable to very complicated ranking measures on large data. For example, it is time consuming to go through the BERT-style models [2, 9], even if hundreds of times. Possible improvements are: (i) take advantage of GPU to accelerate the computations further [27, 49]; (ii) compress the neural network based measures by knowledge distillation techniques [19, 43, 48].

## 3 BIPARTITE GRAPH INDICES

### 3.1 Index Construction

Different from SL2G, we try to construct a graph index only based on the ranking binary function and bypass estimating the distance between base data. Specifically, we build the index as a bipartite graph as illustrated in Figure 2 (c), namely **BipartitE Graph INdices (BEGIN)**. Besides base data points, we exploit a set of query samples (will be introduced later) as the other kinds of nodes. Edges in the graph connect these two kinds of nodes, query samples and base data points. Taking the example in recommendation systems, each item (base data) will connect to users (queries) who rate this item in higher scores. Each user will connect to her favorite items.

The algorithm of BEGIN construction can be found in Algorithm 1. BEGIN is constructed iteratively. The nodes of base data and queries are inserted alternately. The graph is initialized as an empty graph. The first node will not find any neighbors, so it will be inserted as an isolated node. Later, for the following nodes, they will find at least one neighbor by the search algorithm. Note that, there are two respective greedy search algorithms for base data search and query search, as shown in Algorithm 2 and Algorithm 3. For base search, a candidate set of queries will be returned while for query search, a candidate set of base nodes will be returned. Most of the two algorithms are the same, except for the starting point selection (line 2 of Algorithm 2 and 3) and the way we call the binary function $f$ (line 7 of Algorithms 2 and 3).

### 3.2 Two-Hop Edge Selection

Graph-based ANN methods commonly employ some edge selection algorithm which restricts node degrees to improve the searching efficiency. Prior studies [10, 33] show that a carefully designed edge selection method is vital for the searching effectiveness with restricted degrees.

**Algorithm 1** BEGIN Construction

**input:** Base vector set $S$, sample query vector set $Q$, maximum vertex degree $M_x$ for base data, maximum vertex degree $M_q$ for queries, priority queue size $k$ for searching neighbors and ranking measure $f(x, q)$.
Initialize graph $G = \emptyset$
**for** each $x$ in $S$ **do**
    Create a node for $x$ in G.
    Search $k$ vertices $\{p_i\}$ on $G$ by $SearchB(x, G, k, f)$ that have largest values with $x$ in $f(x, p_i)$, place them in descending order.
    $C \leftarrow \emptyset$. $H \leftarrow \emptyset$.
    **for** i $\leftarrow 1$ **to** $k$ **do**
        **if** $p_i$ not in $H$ **then**
            $C \leftarrow C \cup \{p_i\}$
            Add all neighbors' neighbors of $p_i$ to $H$.
            Add an edge from $x$ to $p_i$ in $G$.
        **if** $|C| = M_x$ **then**
            **break**
**for** each $q$ in $Q$ **do**
    Create a node for $q$ in G.
    Search $k$ vertices $\{p_i\}$ on $G$ by $SearchQ(q, G, k, f)$ that have largest values with $q$ in $f(p_i, q)$, place them in descending order.
    $C \leftarrow \emptyset$. $H \leftarrow \emptyset$.
    **for** i $\leftarrow 1$ **to** $k$ **do**
        **if** $p_i$ not in $H$ **then**
            $C \leftarrow C \cup \{p_i\}$
            Add all neighbors' neighbors of $p_i$ to $H$.
            Add an edge from $q$ to $p_i$ in $G$.
        **if** $|C| = M_q$ **then**
            **break**
**output:** index graph $G$

**Algorithm 2** Base Data Search on BEGIN $SearchB(x, G, k, f)$

**Input:** the base data point $x$, the bipartite graph $G = (V_q, V_x, E)$, the priority queue size $k$ and the similarity measure $f(x, q)$.
Randomly choose a vertex $p \in V_q$ as the start point and initialize the priority queue $A \leftarrow \{< f(x, p), p >\}$.
Set $p$ as checked and the rest of vertices as unchecked.
**while** $A$ does not converge **do** //*Not converge: A is still updated with further greedy search.*
    Add unchecked **neighbors' neighbors** of the top element in $A$ to $A$. // *Neighbors' neighbors of a query are queries.*
    Set vertices in $A$ as checked.
    $A \leftarrow$ top-$k$ elements of $v \in A$ in descending order of $f(x, v)$.
**Output:** $A$.

**Algorithm 3** Query Search on BEGIN $SearchQ(q, G, k, f)$

**Input:** the query element $q$, the bipartite graph $G = (V_q, V_x, E)$, the priority queue size $k$ and the similarity measure $f(x, q)$.
Randomly choose a vertex $p \in V_x$ as the start point and initialize the priority queue $A \leftarrow \{< f(p, q), p >\}$.
Set $p$ as checked and the rest of vertices as unchecked.
**while** $A$ does not converge **do**
    Add unchecked **neighbors' neighbors** of the top element in $A$ to $A$. //*Base data's neighbors' neighbors are base data again.*
    Set vertices in $A$ as checked.
    $A \leftarrow$ top-$k$ elements of $v \in A$ in descending order of $f(v, q)$.
**Output:** $A$.

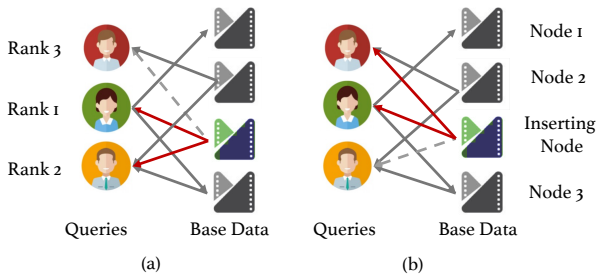



**Figure 3: Illustrations for edge selection methods: (a) connect to top $M$ candidates. (b) connect to diverse candidates by two-hop edge selection.**

For BEGIN, we design a two-hop edge selection method for the bipartite graph structure, as shown in Figure 3 (a) and (b). Say we are inserting a base data point (the colored inserting node) now. We first utilize the base data greedy search (i.e., Algorithm 2) to get top query candidates, labeled as Rank 1, 2 and 3. Figure 3 (a) represents the native non-selection method: the top $M_x$ ($M_x = 2$ in the figure) candidates are connected. This non-selection method would connect too many similar candidates.

To diversify the connection, a two-hop selection method is proposed as shown in Figure 3 (b). It works as follows: The top 1 candidate (i.e., the Rank 1 one) will be connected first. For other candidates, we will check whether they can be reached from any selected candidates in two hops. For example, in Figure 3 (b), the Rank 2 candidate can be reached from the Rank 1 candidate via Node 3. So the Rank 2 candidate will not be selected while the Rank 3 candidate will be connected since it can not be accessed via the Rank 1 candidate in two hops. The example is taken in inserting base data. Edge selection for inserting sample queries is similar. Algorithm details for the two-hop edge selection can be found in Algorithm 1 Line 5-12 and Line 15-22.

To ensure the constructed graph is connected, we preserve one outgoing edge to a randomly picked node, for each inserting data point. Other edges are constructed by the two-hop edge selection algorithm. The idea is similar to the long-range edge in HNSW [33]. In this way, isolated clusters would be connected.

## 3.3 Random Query Generation

BEGIN requires query vector samples to construct bipartite graphs. These query vectors can be generated in model training. For example, when training models for question answering, we will get some intermediate embedding vectors for questions (queries) in the training dataset. If the existing queries are not enough, we can generate query samples randomly based on existing query samples as the following methods.

- **Uniform**. We scan all the existing query vectors to obtain the minimum and maximum values on each dimension. Then we generate vectors such that the values on each dimension are uniformly distributed in $[min, max]$.
- **Normal**. Similar to the uniform case, we compute the sample mean and standard deviation of the existing query vectors. Vectors are generated from a Normal distribution according to the sample mean and standard deviation for each dimension.

- **Duplicate**. To generate a query vector, we randomly select an existing query vector and add a random noise (±1%) to each dimension independently.
- **Midpoint**. For each generated query, an existing query $q_1$ is randomly selected. Then we randomly choose 100 vectors from the existing queries, in which we find the furthest one from $q_1$, noted as $q_2$. The generated query is the middle point of $q_1$ and $q_2$, i.e., $(q_1 + q_2) \times 0.5$. The intuition is: we anticipate that those midpoints compensate the gap between different sample query clusters so that the query vector space is filled well.

It is unclear how to theoretically determine which random query sample generation method might perform the best. We compare these alternatives experimentally in Section 4.5. Note that query samples used in bipartite graph construction or random query generation are separated from testing queries, for fair comparisons.

### 3.4 Fast Search on BEGIN

The online query search is a greedy search on the constructed bipartite graph. Basically, we can directly employ the query search in Algorithm 3. However, this algorithm is inefficient. In each step, we need to check all neighbors' neighbors of the current base data point $\tilde{x}$ as shown in Figure 4 (a). It is $M_x * M_q$ in total. As depicted in the figure, the current node $\tilde{x}$ has two query neighbors and each query neighbor has three base data neighbors. We need to check the six candidates (i.e., $x$'s, in blue) by evaluating $f(x, q)$. And then the next best base data $\tilde{x}$ will be chosen from these six candidates.

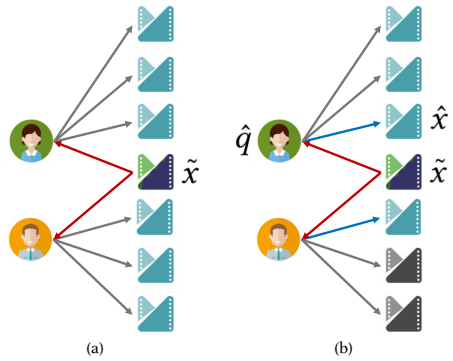



**Figure 4: Illustrations for online query search algorithms: (a) check all neighbors' neighbors of the current base data point $\tilde{x}$. (b) Fast Search: only check top neighbor's ($\hat{q}$) neighbors.**

To improve the ranking performance, we designed a fast search algorithm as represented in Algorithm 4 and Figure 4 (b). As shown in the figure by blue arrows, we will first find the top two-hop neighbor $\hat{x}$ by only checking each one-hop neighbor's first neighbor (neighbors are stored in order). And then we get the corresponding one-hop neighbor $\hat{q}$. All other (unchecked) neighbors of $\hat{q}$ will be checked then. As can be seen, we only need to check at most $M_x + M_q - 1$ nodes in this fast search algorithm (blue colored in Figure 4 (b)). In real implementations, $M_x$ and $M_q$ would be larger than those in the figure, say 16 or 32. Then the superiority of fast search will be more obvious (i.e., $M_x * M_q$ vs. $M_x + M_q - 1$).

**Algorithm 4** Fast Query Search on BEGIN $FastSearch(q, G, k, f)$

**Input:** the query element $q$, the bipartite graph $G = (V_q, V_x, E)$, the priority queue size $k$ and the similarity measure $f(x, q)$.
Randomly choose a vertex $p \in V_x$ as the start point and initialize the priority queue $A \leftarrow \{< f(p, q), p >\}$.
Set $p$ as checked and the rest of vertices as unchecked.
**while** $A$ does not converge **do**
    Get the top element $\tilde{x}$ of A.
    Check $\tilde{x}$'s neighbors' **first unchecked elements** ($\tilde{x}$'s two-hop neighbors) and find $\max_{\hat{x}} f(\hat{x}, q)$. Record the corresponding $\tilde{x}$'s one-hop neighbor $\hat{q}$. //*See Figure 4 (b).*
    Add unchecked neighbors of $\hat{q}$ to $A$.
    Set vertices in $A$ as checked.
    $A \leftarrow$ top-$k$ elements of $v \in A$ in order of $f(v, q)$.
**Output:** $A$.

## 4 EXPERIMENTS

We evaluate the proposed BEGIN with 5 ranking measures, including 2 synthetic measures and 3 neural network based measures.

**1. All-Element-Sum**: $f_{All-Element-Sum}(x, q) = \sum_i x_i + \sum_j q_j$. This measure accumulates all elements of the base vector and the query vector, then returns the sum.

**2. Round-Sum**. This function re-processes the All-Element-Sum result and produces a more complex and non-convex measure: $f_{Round-Sum}(x, q) = round\left(\left(\sum_i x_i + \sum_j q_j\right) \times 10000\right)$ **mod** 100.

**3. MLP-Concate**. The third measure is a neural network. It is based on a state-of-the-art neural network based recommendation method, **MLP**, which was introduced in [18]. The MLP model first concatenates user latent vectors and item latent vectors before going through the Multi-Layer Perceptron network.

**4. MLP-Em-Sum**. The fourth measure is introduced in [42], which transforms two kinds of vectors into a common space by an additional embedding layer and conducts element-wise sum operation before going through the MLP network. The vector dimensions of MLP-Concate and MLP-Em-Sum are set as 32 for users and items.

**5. DeepFM**. DeepFM was proposed in [14], which combines the power of factorization machines for recommendation and deep learning for feature learning in a new neural network architecture. We set the factorization part dimension as 8 and deep learning part as 32. The total dimension for users and items is both 40.

**Table 1: Dataset Statistics.**

| Datasets | # Index Vec | # Queries | # Dim |
|---|---|---|---|
| Yelp | 25,815 | 25,677 | 32 |
| MovieLens | 209,172 | 162,542 | 40 |
| Yelp-1m | 1,058,415 | 25,677 | 32 |
| Amazon | 3,826,085 | 182,032 | 40 |

Table 1 summarizes the datasets in our study.[1] We train MLP-Concate and MLP-Em-Sum on **Yelp**. DeepFM is trained on MovieLens25M (**MovieLens**) and Amazon Movies & TV (**Amazon**). We also include the simulation dataset used in [42]:**Yelp-1m**, which was generated based on MLP-Concate trained on Yelp: randomly generating 40 simulation data points by Gaussian distribution with

[1]https://www.yelp.com/dataset/challenge. http://jmcauley.ucsd.edu/data/amazon. https://grouplens.org/datasets/\movielens/25m.

the original data as mean and 0.1 as the standard deviation. The two measures All-Element-Sum and Round-Sum can be tested on any randomly generated vectors, and here we use vectors generated by MLP-Concate on Yelp. For each dataset, we use 1000 queries as searching queries and the rest for random query generation.

## 4.1 Baselines

To the best of our knowledge, few previous methods focus on the OBFS problem. **SL2G** [42] is considered as the first solution. Additionally, we design another baseline based on sample queries.

**SL2G**. As introduced in Section 2, SL2G constructs $\ell_2$ graph only on based data and conducts a greedy search on the graph by the focused ranking measure $f$.

**Sample-Ave**. As we argued, constructing a graph index by distance/relevance between base data (e.g., in SL2G) is problematic, for complex ranking measures. To verify our argumentation, we design another method based on estimating base data distances:

(i) Generate a query sample set $Q$, such as 1,000 query vectors, separately from the final test set.

(ii) Approximate the distance between two base data points, $x_i$ and $x_j$ by: $\hat{Dis}(x_i, x_j) = \frac{\sum_{q \in Q} |f(x_i,q) - f(x_j,q)|}{|Q|}$.

The graph construction of this method is time-consuming—we need to call the neural network 2,000 (i.e., $2 * |Q|$) times to compute one base-to-base $(x_i, x_j)$ distance. The estimated distance of Sample-Ave should be more proper than the one estimated by $\ell_2$ distance.

Note that we do not compare with other traditional ANN search algorithms (e.g., ANNOY(https://github.com/spotify/annoy) and HNSW [33]) since most of them are not designed for the generic ranking problem, OBFS or specifically fast neural ranking. It was demonstrated that these methods are dramatically worse than SL2G when applying them on fast neural ranking [42].

## 4.2 Experimental Settings

To generate labels, we calculate most relevant base data points for each query by each corresponding binary function $f$. Experiments on top-1 and 10 labels are recorded. Note that for neural network based ranking measures, one binary function $f$ corresponds to one network structure (e.g., MLP-Concate) trained on one dataset.

For evaluation measures, we use **Recall vs. Time** (queries per second) and **Recall vs. Computations** (percentage of pairwise computations) to evaluate the searching performance of different algorithms. Recall vs. Time reports how many queries the algorithm can process per second at each recall level. Recall vs. Computations reports the amount/percentage of pair-wise computations that the search algorithm costs at each recall level. We will show both of these perspectives in the following experiments for a comprehensive evaluation. Recall is $\frac{|A \cap B|}{|B|}$, where $A$ is the return item set and $B$ is the ground truth set. Both $A$ and $B$ have $N$ items for Top-$N$ Recall. Note that we always return $N$ items for the Top-$N$ case, so Recall@$N$ = Precision@$N$ here.

The three method, SL2G, Sample-Ave and BEGIN have three common parameters: $M$ (i.e., $M_x$ in BEGIN), $k_{\text{construction}}$ and $k_{\text{search}}$, which control the degrees of vertices and the number of search attempts. BEGIN has one more parameter $M_q$ for the degree of query points. To make a fair comparison, we vary these parameters over a fine grid. For each algorithm in each experiment, we will have multiple points scattered on the plane. To plot curves, we first find out the best recall number, *max-recall*. Then 100 buckets are produced by splitting the range from 0 to *max-recall* evenly. For each bucket, the best result along the other axis (e.g., the highest queries per second or the lowest percentage of pair-wise computations) is chosen. In other words, we select the parameters that maximize the performance in a given recall range.

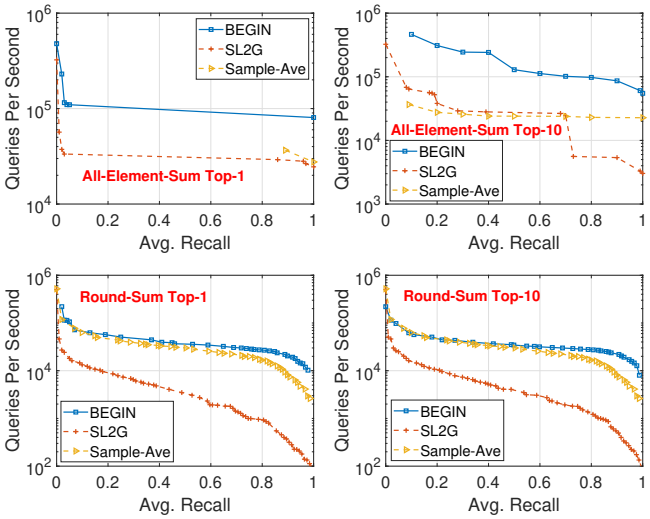



**Figure 5: Experimental results for synthetic measures –All-Element-Sum and Round-Sum– in Recall vs. Time. The best results are in the upper right corner.**

## 4.3 Results for Synthetic Measures

We first show experimental results on synthetic ranking measures. Figure 5 shows results in Recall vs. Time. Each row is for each ranking measure, All-Element-Sum or Round-Sum. Results for Top-1 and Top-10 labels are represented by the two columns. Correspondingly, Figure 6 represents results via the view of Recall vs. Computations.

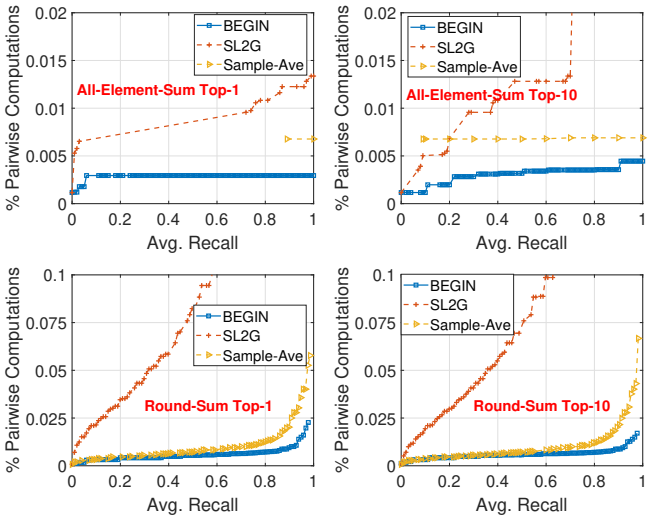



**Figure 6: Experimental results for synthetic measures in Recall vs. (percentage of pairwise) Computations. The best results are in the lower right corner.**

Both of the two evaluation metrics tell similar trends in comparison. The designed baseline Sample-Ave works better than previous state-of-the-art, SL2G. As introduced above, Sample-Ave estimates the distance between base data by sampled queries. That will make more sense than estimating the distance by $\ell_2$ distance. Although Sample-Ave shows superiority over SL2G, it is difficult to apply it in real systems. Sample-Ave requires multiple times of $f(x, q)$ computations (i.e., the size of query sample set), for each pair of base data points. It will take a long time to construct indices for large datasets. The proposed BEGIN works significantly better than the two baselines. No matter the ranking measure is a simple function (i.e., All-Element-Sum) or a complex non-convex function (i.e., Round-Sum), BEGIN works consistently well. The reason is that BEGIN does not estimate the distance between base data but explore ranking function directly. As analyzed above, distances between base data are not well-defined, recalling the example of recommendation in Section 2.3. An estimated distance between base data may be good for some queries but may hurt the ranking performance for other queries. Bypassing estimating the distance, the proposed bipartite graph represents relationships between base data in the neighbor intersection, which is more flexible for different queries.

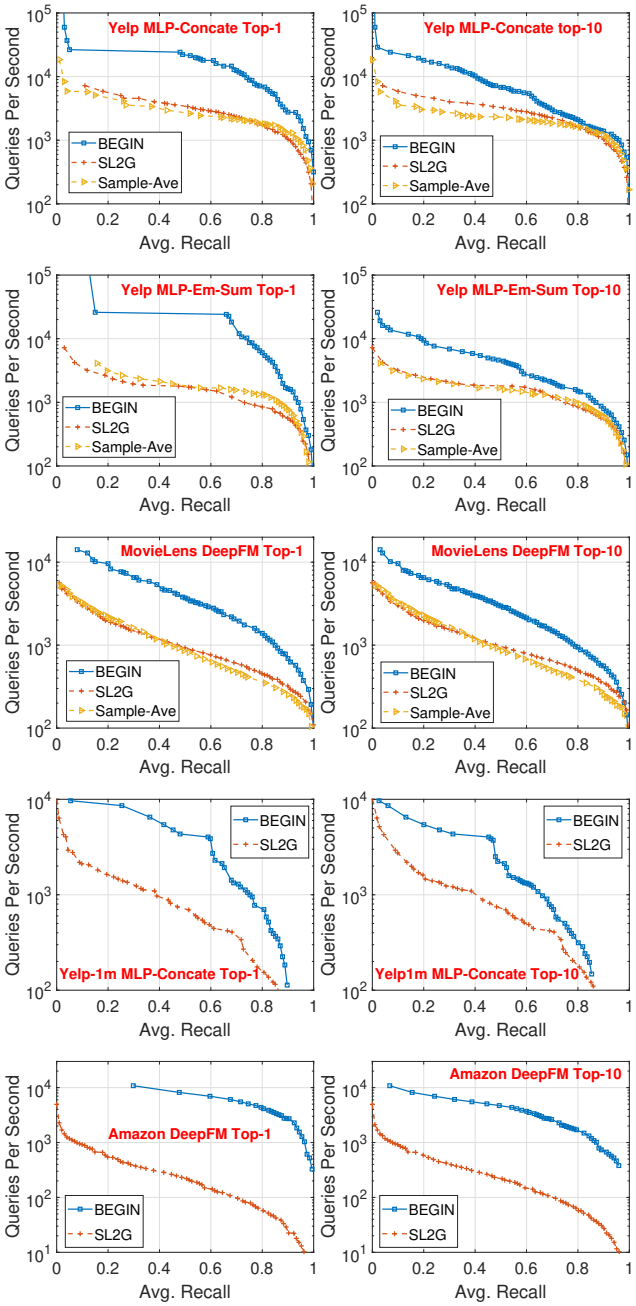



**Figure 7: Experimental results for the neural network measures from the view of Recall vs. Time. The best results are in the upper right corner.**

## 4.4 Results Neural Network Measures

In this section, we introduce experimental results for neural network based ranking measures. Experimental results by the two evaluation metrics are shown in Figure 7 and Figure 8. Results via the view of Recall vs. Computations are only shown for partial datasets due to the limited space. Other results have similar trends. Note that results for Sample-Ave are not reported on the last two larger datasets since Sample-Ave is too time-consuming in index construction for these sizes of datasets. As can be seen, the proposed method works better than baselines, especially for Top-1 recalls. On these neural network ranking measures, Sample-Ave works similarly with SL2G, not better than SL2G as on synthetic ranking measures. The reason is that neural network based ranking measures are much more complicated than synthetic ranking

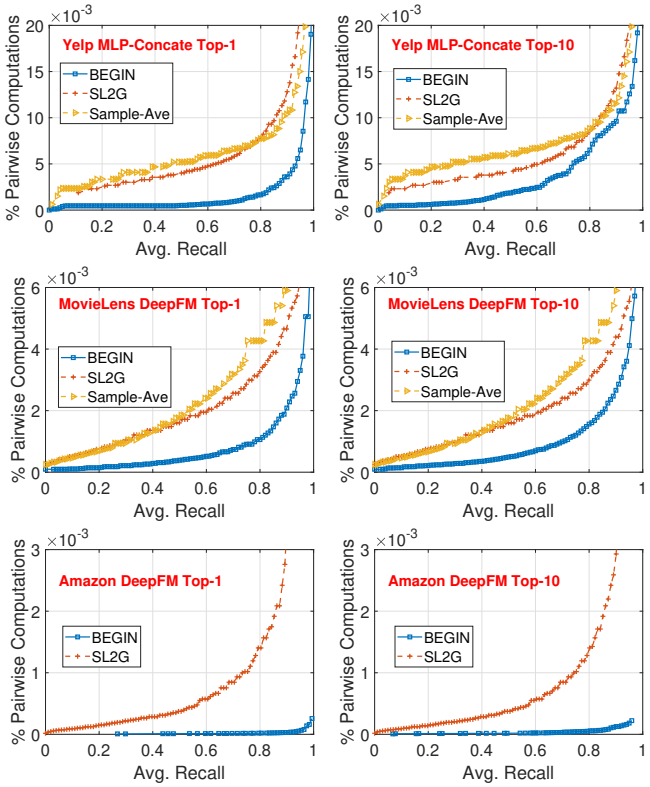



**Figure 8: Experimental results for the neural network measures from the view of Recall vs. Computations. The best results are in the lower right corner.**

ones. As we analyzed above, SL2G and Sample-Ave are problematic in estimating the distance between base data. The issue becomes more obvious on these complex ranking measures. However, the proposed BEGIN works consistently well in various cases. The superiority of the bipartite graph structure is demonstrated.

**Scalability of BEGIN.** The last two datasets, Yelp-1m and Amazon, are much larger, with more than 1 million and 3 millions base vectors. As can be seen, the gap between BEGIN and SL2G is even larger on these larger datasets. For example, on the Amazon dataset, to achieve 80% Top-1 recall, SL2G can deal with 57 queries per second. To get the same recall level, BEGIN can process 4,366 queries per second: 76 times faster than SL2G. It is clear that BEGIN poses a better performance in scalability. That is vital for real applications.

## 4.5 Evaluation of Query Generation Methods

As introduced in Section 3.3, there are multiple methods to generate query vectors. In this section, we study the performance of generating methods. The performance results of different generating methods are shown in Figure 9. Due to the limited space, we only show results for Yelp MLP-Concate.

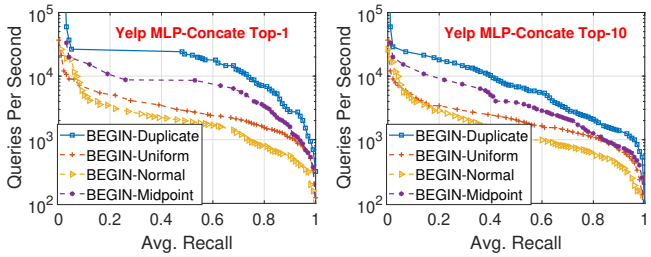



**Figure 9: Experimental results for different query sample generation method, introduced in Section 3.3.**

As shown in Figure 9, among the four methods, Duplicate shows superiority in the ranking performance. The second good method is Midpoint. Normal works worst in the four. The results tell that the original distribution of query data matters in bipartite graph construction. Duplicate works best since it keeps the original data distribution well. Midpoint has similar distribution with the original data too. Normal generates the set in completely new distribution, that would be very different from the original distribution. That is why it is even worse than Uniform. We use the method, Duplicate in other experiments.

We also study the performance in different amounts of generated query samples. Results for Duplicate on Amazon DeepFM are shown in Figure 10. The number of base vectors in Amazon dataset is about 3.8m. The result of BEGIN-Duplicate-4m is a little better than others. In other experiments, we generated similar amounts of query samples as base vectors by the method of Duplicate. As can be seen, the performance of BEGIN is not sensitive to the query sample size. Generally, we recommend the size is comparable to the base vector size (i.e., $0.5-2$ times of base vectors). The reason is degrees of nodes are restricted in the bipartite graph. If one kind of nodes (query samples or base vectors) is extremely less than the other, it may cause the graph unconnected.

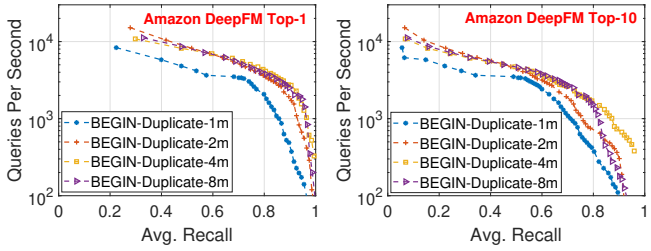



**Figure 10: Experimental results for different amounts of generated query samples.**

## 4.6 Evaluation of Search Algorithms

After constructing the bipartite graph indices, we design algorithms to conduct online/fast query search, as represented in Algorithm 3 and Algorithm 4 respectively. We show the performance comparison of these two algorithms in Figure 11. Only results on MLP-Concate are reported due to the limited space. As can be seen, the **FastSearch** algorithm is much more efficient than the native QuerySearch. As discussed in Section 3.4, FastSearch reduces the comparison amount from $M_x * M_q$ to $M_x + M_q - 1$ (i.e., from $O(N^2)$ to $O(N)$, where $N$ is the degree of nodes). We enable this optimization–FastSearch–in all comparisons with SL2G.

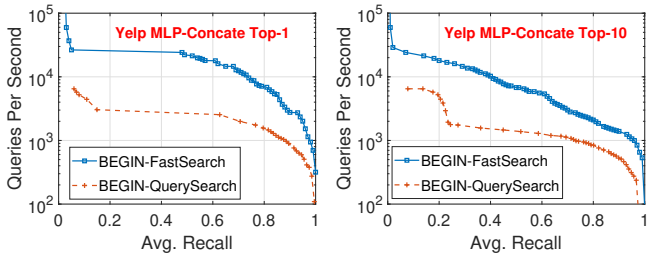



**Figure 11: Results for the two search algorithms: QuerySearch (i.e., Algorithm 3) and FastSearch (i.e., Algorithm 4).**

## 5 CONCLUSIONS

Neural networks are more powerful to model complex relationships between vectors than simple vector distances such as $\ell_2$ distance, cosine similarity and inner product. However, it is hard to efficiently rank vectors according to neural network measures—neural network measures are computation-intensive. The complexities of these measures bring challenges in constructing indices for fast vector retrieval. Previous work extends traditional approximate near neighbor search and defines the fast neural ranking problem formally as, Optimal Binary Function Search (OBFS) and also provide a simple solution, SL2G. While SL2G is problematic since it tries to estimate distances between base data, which is undefined. In this paper, we propose bipartite graph indices (BEGIN) for the problem. BEGIN bypasses estimating distances among base data and constructs indices only by the given ranking measure $f$. Based on bipartite graph indices, a FastSearch algorithm is introduced. In experiments, we evaluate the proposed algorithm on 2 synthetic ranking measures and 3 neural network measures. Experimental results show that the ranking performance of the proposed BEGIN is significantly better than the previous state-of-the-art method.

## REFERENCES


[1] Lawrence Cayton. 2008. Fast nearest neighbor retrieval for bregman divergences. In *Proceedings of the Twenty-Fifth International Conference on Machine learning (ICML)*. Helsinki, Finland, 112–119.

[2] Wei-Cheng Chang, Felix X Yu, Yin-Wen Chang, Yiming Yang, and Sanjiv Kumar. 2020. Pre-training Tasks for Embedding-based Large-scale Retrieval. In *Proceedings of the 8th International Conference on Learning Representations (ICLR)*. Addis Ababa.

[3] Moses S. Charikar. 2002. Similarity estimation techniques from rounding algorithms. In *Proceedings on 34th Annual ACM Symposium on Theory of Computing (STOC)*. Montreal, Canada, 380–388.

[4] Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. 2017. Reading Wikipedia to Answer Open-Domain Questions. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL)*. Vancouver, Canada, 1870–1879.

[5] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep Neural Networks for YouTube Recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems (RecSys)*. Boston, MA, 191–198.

[6] Ryan R Curtin and Parikshit Ram. 2014. Dual-tree fast exact max-kernel search. *Statistical Analysis and Data Mining: The ASA Data Science Journal* 7, 4 (2014), 229–253.

[7] Ryan R Curtin, Parikshit Ram, and Alexander G Gray. 2013. Fast exact max-kernel search. In *Proceedings of the 13th SIAM International Conference on Data Mining (SDM)*. Austin,TX, 1–9.

[8] Mostafa Dehghani, Hamed Zamani, Aliaksei Severyn, Jaap Kamps, and W. Bruce Croft. 2017. Neural Ranking Models with Weak Supervision. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*. Shinjuku, Tokyo, 65–74.

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*. Minneapolis, MN, 4171–4186.

[10] Cong Fu, Chao Xiang, Changxu Wang, and Deng Cai. 2019. Fast approximate nearest neighbor search with the navigating spreading-out graph. *Proceedings of the VLDB Endowment* 12, 5 (2019), 461–474.

[11] Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. 2013. Optimized Product Quantization for Approximate Nearest Neighbor Search. In *Proceedings of the 2013 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Portland, OR, 2946–2953.

[12] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. 1999. Similarity Search in High Dimensions via Hashing. In *Proceedings of 25th International Conference on Very Large Data Bases (VLDB)*. Edinburgh, Scotland, UK, 518–529.

[13] Michel X. Goemans and David P. Williamson. 1995. Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems Using Semidefinite Programming. *J. ACM* 42, 6 (1995), 1115–1145.

[14] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. 2017. DeepFM: A Factorization-Machine based Neural Network for CTR Prediction. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence (IJCAI)*. Melbourne, Australia, 1725–1731.

[15] Jiafeng Guo, Yixing Fan, Qingyao Ai, and W. Bruce Croft. 2016. A Deep Relevance Matching Model for Ad-hoc Retrieval. In *Proceedings of the 25th ACM International Conference on Information and Knowledge Management (CIKM)*. Indianapolis, IN, 55–64.

[16] Jiafeng Guo, Yixing Fan, Liang Pang, Liu Yang, Qingyao Ai, Hamed Zamani, Chen Wu, W. Bruce Croft, and Xueqi Cheng. 2020. A Deep Look into neural ranking models for information retrieval. *Inf. Process. Manag.* 57, 6 (2020), 102067.

[17] Kiana Hajebi, Yasin Abbasi-Yadkori, Hossein Shahbazi, and Hong Zhang. 2011. Fast Approximate Nearest-Neighbor Search with k-Nearest Neighbor Graph. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI)*. Barcelona, CA, 1312–1317.

[18] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *Proceedings of the 26th International Conference on World Wide Web (WWW)*. Perth, Australia, 173–182.

[19] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531* (2015).

[20] Piotr Indyk and Rajeev Motwani. 1998. Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing (STOC)*. Dallas, TX, 604–613.

[21] Sergey Ioffe. 2010. Improved Consistent Sampling, Weighted Minhash and L1 Sketching. In *Proceedings of the 10th IEEE International Conference on Data Mining (ICDM)*. Sydney, Australia, 246–255.

[22] Masajiro Iwasaki. 2016. Pruned Bi-directed K-nearest Neighbor Graph for Proximity Search. In *Proceedings of the 9th International Conference on Similarity Search and Applications (SISAP)*, Vol. 9939. Tokyo, Japan, 20–33.

[23] Masajiro Iwasaki and Daisuke Miyazaki. 2018. Optimization of indexing based on k-nearest neighbor graph for proximity search in high-dimensional data. *arXiv preprint arXiv:1810.07355* (2018).

[24] Hervé Jégou, Matthijs Douze, and Cordelia Schmid. 2008. Hamming Embedding and Weak Geometric Consistency for Large Scale Image Search. In *Proceedings of the 10th European Conference on Computer Vision (ECCV), Part I*. Marseille, France, 304–317.

[25] Hervé Jégou, Matthijs Douze, and Cordelia Schmid. 2011. Product Quantization for Nearest Neighbor Search. *IEEE Trans. Pattern Anal. Mach. Intell.* 33, 1 (2011), 117–128.

[26] Jyun-Yu Jiang, Patrick H. Chen, Cho-Jui Hsieh, and Wei Wang. 2020. Clustering and Constructing User Coresets to Accelerate Large-scale Top-K Recommender Systems. In *Proceedings of the Web Conference (WWW)*. Taipei, 2177–2187.

[27] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2021. Billion-Scale Similarity Search with GPUs. *IEEE Trans. Big Data* 7, 3 (2021), 535–547.

[28] Ping Li. 2017. Linearized GMM Kernels and Normalized Random Fourier Features. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. Halifax, Canada, 315–324.

[29] Ping Li, Xiaoyun Li, Gennady Samorodnitsky, and Weijie Zhao. 2021. Consistent Sampling Through Extremal Process. In *Proceedings of the Web Conference (WWW)*. Virtual Event / Ljubljana, Slovenia, 1317–1327.

[30] Ping Li, Michael Mitzenmacher, and Anshumali Shrivastava. 2014. Coding for Random Projections. In *Proceedings of the 31th International Conference on Machine Learning (ICML)*. Beijing, China, 676–684.

[31] Xiaoyun Li and Ping Li. 2021. One-Sketch-for-All: Non-linear Random Features from Compressed Linear Measurements. In *Proceedings of the 24th International Conference on Artificial Intelligence and Statistics (AISTATS)*. Virtual Event, 2647–2655.

[32] Zhengdong Lu and Hang Li. 2013. A Deep Architecture for Matching Short Texts. In *Advances in Neural Information Processing Systems (NIPS)*. Lake Tahoe, NV, 1367–1375.

[33] Yury A. Malkov and Dmitry A. Yashunin. 2020. Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. *IEEE Trans. Pattern Anal. Mach. Intell.* 42, 4 (2020), 824–836.

[34] Mark Manasse, Frank McSherry, and Kunal Talwar. 2010. *Consistent Weighted Sampling*. Technical Report MSR-TR-2010-73. Microsoft Research.

[35] Bhaskar Mitra and Nick Craswell. 2018. An introduction to neural information retrieval. *Foundations and Trends® in Information Retrieval* (2018).

[36] Stanislav Morozov and Artem Babenko. 2018. Non-metric Similarity Graphs for Maximum Inner Product Search. In *Advances in Neural Information Processing Systems (NeurIPS)*. Montreal, Canada, 4726–4735.

[37] Ali Rahimi and Benjamin Recht. 2007. Random Features for Large-Scale Kernel Machines. In *Advances in Neural Information Processing Systems (NIPS)*. Vancouver, Canada, 1177–1184.

[38] Anand Rajaraman and Jeffrey David Ullman. 2011. *Mining of massive datasets*. Cambridge University Press.

[39] Aliaksei Severyn and Alessandro Moschitti. 2015. Learning to Rank Short Text Pairs with Convolutional Deep Neural Networks. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*. Santiago, Chile, 373–382.

[40] Anshumali Shrivastava and Ping Li. 2012. Fast Near Neighbor Search in High-Dimensional Binary Data. In *Proceedings of European Conference on Machine Learning and Knowledge Discovery in Databases (ECML-PKDD)*. Bristol, UK, 474–489.

[41] Shulong Tan, Zhaozhuo Xu, Weijie Zhao, Hongliang Fei, Zhixin Zhou, and Ping Li. 2021. Norm Adjusted Proximity Graph for Fast Inner Product Retrieval. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*. Virtual Event, Singapore, 1552–1560.

[42] Shulong Tan, Zhixin Zhou, Zhaozhuo Xu, and Ping Li. 2020. Fast item ranking under neural network based measures. In *Proceedings of the 13th International Conference on Web Search and Data Mining (WSDM)*. 591–599.

[43] Jiaxi Tang and Ke Wang. 2018. Ranking Distillation: Learning Compact Ranking Models With High Performance for Recommender System. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD)*. London, UK, 2289–2298.

[44] Yi Tay, Luu Anh Tuan, and Siu Cheung Hui. 2018. Latent Relational Metric Learning via Memory-based Attention for Collaborative Ranking. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web (WWW)*. Lyon, France, 729–739.

[45] Xiang Wu, Ruiqi Guo, Ananda Theertha Suresh, Sanjiv Kumar, Daniel N. Holtmann-Rice, David Simcha, and Felix X. Yu. 2017. Multiscale Quantization for Fast Similarity Search. In *Advances in Neural Information Processing Systems (NIPS)*. Long Beach, CA, 5745–5755.

[46] Yubao Wu, Ruoming Jin, and Xiang Zhang. 2014. Fast and unified local search for random walk based k-nearest-neighbor query in large graphs. In *Proceedings of the International Conference on Management of Data (SIGMOD)*. Snowbird, UT, 1139–1150.

[47] Chenyan Xiong, Zhuyun Dai, Jamie Callan, Zhiyuan Liu, and Russell Power. 2017. End-to-end neural ad-hoc ranking with kernel pooling. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*. Shinjuku, Tokyo, 55–64.

[48] Ying Zhang, Tao Xiang, Timothy M. Hospedales, and Huchuan Lu. 2018. Deep Mutual Learning. In *Proceedings of the 2018 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Salt Lake City, UT, 4320–4328.

[49] Weijie Zhao, Shulong Tan, and Ping Li. 2020. SONG: Approximate Nearest Neighbor Search on GPU. In *Proceedings of the 36th IEEE International Conference on Data Engineering (ICDE)*. Dallas, TX, 1033–1044.

[50] Zhixin Zhou, Shulong Tan, Zhaozhuo Xu, and Ping Li. 2019. Möbius Transformation for Fast Inner Product Search on Graph. In *Advances in Neural Information Processing Systems (NeurIPS)*. Vancouver, Canada, 8216–8227.