Fast Item Ranking under Neural Network based Measures

Shulong Tan, Zhixin Zhou, Zhaozhuo Xu, Ping Li
Cognitive Computing Lab
Baidu Research
1195 Bordeaux Dr, Sunnyvale CA, 94089, USA
10900 NE 8th St, Bellevue WA, 98004, USA
{shulongtan,zhixinzhou,v_xuzhaozhuo,liping11}@baidu.com

ABSTRACT

Recently, plenty of neural network based recommendation models have demonstrated their strength in modeling complicated relationships between heterogeneous objects (i.e., users and items). However, the applications of these fine trained recommendation models are limited to the off-line manner or the re-ranking procedure (on a pre-filtered small subset of items), due to their time-consuming computations. Fast item ranking under learned neural network based ranking measures is largely still an open question.

In this paper, we formulate ranking under neural network based measures as a generic ranking task, Optimal Binary Function Search (OBFS), which does not make strong assumptions for the ranking measures. We first analyze limitations of existing fast ranking methods (e.g., ANN search) and explain why they are not applicable for OBFS. Further, we propose a flexible graph-based solution for it, Binary Function Search on Graph (BFSG). It can achieve approximate optimal efficiently, with accessible conditions. Experiments demonstrate effectiveness and efficiency of the proposed method, in fast item ranking under typical neural network based measures.

ACM Reference Format:
Shulong Tan, Zhixin Zhou, Zhaozhuo Xu, Ping Li. 2020. Fast Item Ranking under Neural Network based Measures. In *The Thirteenth ACM International Conference on Web Search and Data Mining (WSDM '20), February 3–7, 2020, Houston, TX, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3336191.3371830

1 INTRODUCTION

In recent years, more and more deep learning models are applied for recommendation and information retrieval systems [30, 43]. Deep neural networks, especially the Siamese network architecture [32], have demonstrated their powerful ability in high-level abstraction and semantic understanding between heterogeneous objects. Among these models, there are two main model design paradigms [43]: (a) **representation learning**; (b) **matching function learning**. Both paradigms are widely adopted in various neural network based matching/ranking methods. As shown in Figure 1, the first paradigm learns representations for the two objects (e.g., user and item) via separate models. Then a classical matching function such as cosine similarity or inner product is chosen to generate the output score. Different from representation learning, the second paradigm combines features or embedding vectors of the two objects earlier and learns matching functions from labeled data.

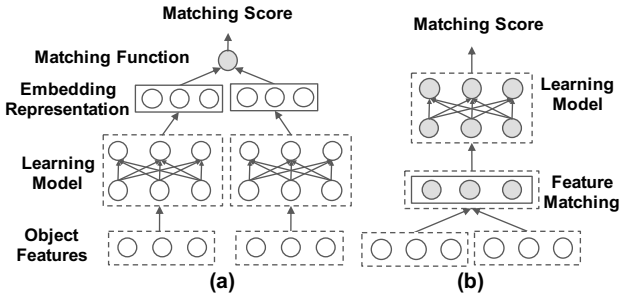

Figure 1: Neural network based matching models: (a) representation learning; (b) matching function learning.

The simple matching functions (usually bi-linear or metric functions) adopted in representation learning models often limit the performance in modeling complicated relationships [28]. While matching function learning models usually provide more promising performance in prediction precision because of their powerful model capacity with non-convex neural network based matching functions. However, as the computation of neural networks is both time and resources consuming, the inference process is often done in an off-line manner, such as generating the recommendation lists every week instead of on-line generating. This expediency may hurt user experience since users' most recent ratings or posted queries critically influence user profiles. Another common trick is to build a two-step service. Firstly, a small subset of items is generated based on fast and simple filters. Then a re-ranking procedure is conducted on this subset via fine trained advanced ranking measures [6, 34]. This re-ranking trick restricts the performance since relevant items may be filtered out in the first step.

To more effectively exploit neural network based matching functions, in this paper we try to speed up the ranking procedure after the matching functions being learned, referred to as **fast neural ranking**. We formulate fast neural ranking as a generic ranking task, **Optimal Binary Function Search (OBFS)**. We state the ranking function f here as a generic continuous binary function and do not further restrict the forms of it. Different from our setting, traditional Approximate Nearest Neighbor (ANN) search problems, which can be considered as special cases of fast OBFS, have strong assumptions for ranking measures. They mainly concentrate on searching by metric distances, such as ℓ_2 distance or angular distance. Beyond that, a few works try to extend the searching measure

WSDM '20, February 3–7, 2020, Houston, TX, USA

ACM ISBN 978-1-4503-6822-3/20/02…$15.00
https://doi.org/10.1145/3336191.3371830

to non-metric ones, such as inner product [2, 33, 36, 37, 49], Mercer kernel [8] and Bregman divergence [4]. Note that previous approaches take the advantages of linearity or convexity of ranking measures. While neural network based ranking measures are usually nonlinear and non-convex, sometimes asymmetric. To the best of our knowledge, this is the first work focus on fast ranking under generic measures, such as neural network based ones.

For approaches, we select **search on graph** as the basic searching methodology. Search on graph methods show significant superiority in searching in metric distances [29] and inner product [31, 38, 49] in previous studies. Besides, graph-based index structures have the flexibility that users can define any (symmetric) matching function as the similarity measure in graph construction and searching. Under the definition of OBFS, performing a traditional search on graph methods can be summarized as a common methodology: constructing the (approximate) Delaunay graph with respect to the binary function f for all indexing data and then greedy searching most relevant vertices on the graph by the same f, for each query q. Outperforming efficiency of this methodology on metric distances was previously validated [29]. However, constructing or approximately constructing Delaunay graphs for complicated binary functions is extremely difficult. Besides, it requires the searching function is symmetric (for the two input vectors). It is difficult to extend it for more generic OBFS problems.

To overcome these limitations, we develop a specific implementation of Binary Function Search on Graph (BFSG), called *Search on L2 Graph (SL2G)*. Different from other methods, SL2G constructs the index graph by ℓ_2 distance among searching data, no matter which searching measure f we are working on. In the searching phase, SL2G searches by the focus binary function f. This paradigm bypasses the infeasible task, i.e., constructing Delaunay graphs w.r.t. complicated measures, and breaks the symmetric limitation. From the theoretical perspective, SL2G approximates the coordinate descent in Euclidean space. Even if f is non-convex (e.g., deep neural models), SL2G can reach an approximate local optimum, based on affordable assumptions. In real applications, we often search on the graph in multiple paths at the same time. As shown in empirical results, the global optimum is also not difficult to be achieved by SL2G.

In summary, the contributions of this paper are as below:

- We consider a challenging problem, fast neural ranking and formulate it as a generic ranking task, Optimal Binary Function Search (OBFS). Under OBFS, more advanced matching measures can be considered in large-scale search.
- We propose BFSG and a sub-linear solution, SL2G, to tackle the fast OBFS problem. SL2G breaks the limitations of existing search on graph methods and is applicable for any complex searching measures. Theoretical analysis for SL2G is provided.

The organization of this paper is as below: in the next section we review classical neural network based ranking models and definite the problem of fast neural ranking formally. Potential solutions for fast neural ranking are reviewed in Section 3 and we will explain why these previous methods are inapplicable for complicated ranking measures. The proposed method, SL2G and its theoretical analysis are represented in Section 4. In section 5, we study two neural network based ranking measures and conduct experiments. The whole paper is concluded in Section 6.

2 FAST NEURAL RANKING

We first review popular neural network based ranking models and state the motivation of fast neural ranking.

2.1 Neural Ranking Models

In recent years, with the advance of deep learning technology, we have witnessed a growing body of work in applying shallow or deep neural networks to the ranking problems (referred to as **neural ranking models**) [30], such as Information Retrieval [15, 16, 42] and recommendation [6, 18]. As analyzed in the above section, these models are usually based on Siamese architecture and can be categorized into representation learning methods and matching function learning methods [43].

Representation learning approaches [19, 35, 44, 47], as shown in Figure 1 (a), assume the similarity between two heterogeneous vectors depends on their positions in an unobserved latent space. Therefore, the goal of representation learning is to find this latent space as well as two parallel mapping models for each input vector. Although representation learning models achieve promising performance in some tasks, researchers found that they are often incapable of modeling the semantic matching between complicated objects [28]. Firstly, representing complicated objects with meaningful vectors is difficult. Secondly, simple matching functions like cosine or inner product cannot capture the complicated interactions between objects, which often in nonlinear and non-convex manners, such as a user may like very different two categories of movies. Thus, more and more works focus on learning meaningful matching functions, especially in forms of deep neural networks.

Matching function learning approaches [15, 28] assume that there exists a mapping between user and item (or query and document). Therefore, the goal of matching function learning is to learn the mapping by well-designed models. As shown in Figure 1(b), matching function learning models has flexible feature matching methods before going through the learning model, which is beneficial for modeling complicated interactions between objects. For the matching models (i.e., matching functions to be learned), Multi-Layer Perceptron (MLP) is commonly adopted [9, 18, 34, 39]. Kernel Pooling is also studied previously [42]. It has been shown, matching function learning methods outperform representation learning methods in some tasks [43]. However, brute force ranking by these learned measures is too time-consuming for on-line services. It motivates us to speed up the neural ranking procedure.

2.2 Problem Definition

Here, we formulate the neural ranking task in a more generic way, **Optimal Binary Function Search** (**OBFS**). It will be shown that traditional ANN search problems are special cases of OBFS.

Definition 1. *(OBFS) Let X and Y be subsets of Euclidean spaces (possibly with different dimensions), given a data set $S = \{x_1, \ldots, x_n\} \subset X$ and a continuous binary function, $f : X \times Y \to \mathbb{R}$, given $q \in Y$, OBFS aims to find*

$$\arg\max_{x_i \in S} f(x_i, q). \tag{1}$$

In the same manner, we aims to find the set of top-k solutions T, with $|T| = k$ and satisfies

$$\min_{x \in T} f(x, q) \geq \max_{x' \notin T} f(x', q). \tag{2}$$

Let us consider $X \subset \mathbb{R}^d$ and $Y \subset \mathbb{R}^k$. Note that we consider subsets of Euclidean space because some functions' domain is not the whole space, for example, cosine is undefined at origin. In the recommendation framework, we consider items as searching data in X and the user set as queries in Y. The matching function (i.e., the binary function f) is specified or learned, which takes a pair of item and user as input. The output of f is the ranking score for recommendation. ℓ_2 distance, angular distance and inner product are special cases of binary functions when $d = k$.

As introduced above, the efficiency of ranking/searching is also crucial besides of effectiveness. The brute force scanning method is often too time-consuming in real applications. So many previous efforts are paid in finding sub-linear or approximate fast searching approaches. In the next section, we will explain why previous fast ranking/searching frameworks are inapplicable for complicated searching measures, such as fast neural ranking.

3 WHY NOT PREVIOUS METHODS?

In this section, we will explain why previous methods cannot be extended to generic fast OBFS. Firstly, from the ranking measure respective, we will review some classical measures which previously studied and then distinguish the particularity of generic measures (e.g., neural network based ones). Then, the main fast search methodologies will be checked whether can be extended for fast generic ranking. Search on graph will be highlighted at last.

3.1 Previous Studied Ranking Measures

Section 2.2 defines a family of ranking measures, Optimal Binary Function Search (OBFS), which contains classical ones (e.g., cosine and inner product) and neural network based measures as special cases. In this family, measures in metric space are widely studied, based on which fast or approximate search has been a classical data science task for decades, often referred as Approximate Nearest Neighbor (ANN) search or Similarity Search. Two most popular metric measures are ℓ_2 distance and cosine similarity.

Efficient searching beyond metric distances is considered hard. Only a few simple non-metric measures were studied in the literature, among which searching by the inner product referred as Maximum Inner Product Search (MIPS) is popular because of its wide applicability in recommendation and classification tasks [2, 31, 33, 36, 37, 45, 46]. The Mercer kernels are the extension of inner product in Hilbert space $\mathcal{H}$. Searching by Mercer kernels is often referred as Max-Kernel Search [7, 8]. Another example is searching by Bregman divergence [4]. For more complicated matching measures, efficient searching is still an open question.

It is clear that previously studied ranking measures are mainly symmetric ones, especially metric distances. Bregman divergence requires a convex ranking function, and both variables must come from the same convex set. Recall that those representation learning models (introduced in Section 2.1) mainly choose cosine or inner product as matching functions. So fast ranking based on these models can be solved by existing methods. However, neural network based measures (from matching function learning models) are usually in forms of Multi-Layer Perceptron (MLP) which are nonlinear and non-convex. To the best of our knowledge, there is no work study fast searching via these generic and complicated measures.

3.2 Fast Searching Methodologies

For Approximate Nearest Neighbor (ANN) search paradigms, each given query is compared with a subset instead of the whole dataset or comparing by shorter codes, which reduce the time complexity significantly while promise high searching recalls. To achieve this purpose, specific index structures are proposed for dense continuous vectors [5, 21] or high dimensional sparse data [3, 25, 26]. For the dense case, which is our focus in this paper, various searching methodologies and indexing structures are applied:

- Hashing (Locality-Sensitive Hashing, LSH) [5, 14, 20, 25, 27].
- Quantization (i.e., Product Quantization, PQ) [12, 21, 22, 40].
- Ball tree or KD tree based [4, 7, 8, 11, 33]
- Graph based (i.e., search on graph) [17, 29, 41].

Most of these methodologies are not flexible to extend to advanced searching measures. For example, one particular LSH algorithm is usually designed for one specific measure, such as SimHash for cosine similarity [5] and "sign Cauchy projections" for chi-square similarity [27]. Ball tree-based methods are proposed for searching by some non-metric measures, such as Max-kernel search [7, 8] and searching by Bregman divergence [4]. These methods depend on a well designed bound corresponding to the particular searching measure, in telling which sub-tree can be ignored in scanning. How to design tighter bound is usually very difficult so the searching efficiency is typically not guaranteed. Besides, it is infeasible to design bounds for neural network based measures.

One exception is graph-based methods, also referred to as search on graph. Graph-based index structures have the flexibility that the user can define any symmetric matching function as similarity measure in graph construction and searching. Recently, researchers extend search on graph from metric space to inner product and achieve promising performance in the MIPS problem [31, 38, 49]. In the next section, we will analyze whether the previous search on graph methods can be applied for fast neural ranking.

3.3 Search on Graph

Graph structures provide a natural way to partition a high dimensional continuous space into discrete regions. Theoretically, formulation of Voronoi Diagram and its dual graph, Delaunay Graph [1, 10] provide the foundation for nearest neighbor search on graph. As specific implementations, k-Nearest Neighbor (kNN) graph was claimed to be an approximation of Delaunay Graph [17]. Inspired by the small world phenomenon, Navigable Small World (NSW) network [23] and Hierarchical NSW (HNSW) [29] are introduced and show their powerful potentials in ANN search.

While search on graph methods often claim that there are no constraints on searching measures (actually must be symmetric) [17, 29], most existing search on graph methods, however, mainly focus on searching by metric distances, with a few exceptions, e.g., [29, 31]. The common methodology behind these search on graph methods can be summarized as: constructing the (approximate) Delaunay graph with respect to binary function f for all indexing data S and then greedy searching most relevant vertices on the graph for each query q by the same f. In constructing approximate Delaunay graphs, HNSW was shown that it is a proper approximate solution for Delaunay graph with respect to metric measures [29] and inner product [31, 38, 49]. However, how to construct the Delaunay graph

algorithmically with respect to complicated f is still unsolved, as it would be difficult to extend HNSW to complicated binary functions. For instance, when X and Y are in different dimensions, HNSW has no clue how to approximate the Delaunay graph. Our experiments will show that HNSW cannot construct proper Delaunay graphs with respect to neural network based measures and works badly in fast neural ranking. To overcome this limitation, we bypass building Delaunay graphs w.r.t. complicated binary functions and propose a new search on graph algorithm in the next section, which only requires to construct Delaunay graphs with respect to ℓ_2 distance.

4 BINARY FUNCTION SEARCH ON GRAPH

In this section, we introduce a new graph based solution for generic OBFS tasks. Before detailing the implementations, we explain why the proposed method can solve OBFS from a theoretical perspective.

4.1 The Proposed Approach

Generally, our methodology tries to solve the binary function search problem by graph-based index, which can be summarized as: **Binary Function Search on Graph** (**BFSG**). As analyzed above, building Delaunay graphs is algorithmically difficult for most of the binary functions except some easier cases, such as ℓ_2 distance. To bypass constructing Delaunay graphs with respect to complicated binary functions, we proposed a specific method, **Search on L2 Graph** (**SL2G**), for the OBFS problem. The basic idea of SL2G is: no matter what the given binary function f is, we construct a Delaunay graph (or an approximate one) with respect to ℓ_2 distance (which is defined on searching data X and independent of queries) in the indexing step and then perform the greedy search on this graph by the binary function f in the searching step. SL2G only constructs Delaunay graphs with respect to ℓ_2 distance but independent with the focus function f. As building and approximating Delaunay graph with respect to ℓ_2 distance appears in many previous works, SL2G is applicable for any generic binary functions. If f is the negative ℓ_2 distance as in Example 1, SL2G is equivalent to the previous search on graph methods, such as HNSW.

In short, the proposed fast ranking framework can be summarized as the following two steps: (i) Constructing an approximate Delaunay graph with respect to ℓ_2 distance. (ii) For query q, performing greedy search on the graph by the focus search measure $f_q = f(\cdot, q)$. In the next section, we will explain, why the proposed Sl2G can solve OBFS approximately.

4.2 Theoretical Analysis

Firstly, we definite Voronoi cell and its dual graph, Delaunay graph. Let $S = \{x_1, \ldots, x_n\} \subset X$ be the dataset.

Definition 2. *The Voronoi cell R_i, for the query space Y, with respect to f and x_i is the set*

$$R_i := \{q \in Y : f(x_i, q) \geq f(x_j, q) \text{ for all } j \in [n]\}. \tag{3}$$

Voronoi cells form a diagram of Y. Its dual graph in X, namely Delaunay graph, is defined as follows.

Definition 3. *The Delaunay graph G with respect to f and S is an undirected graph with vertices S satisfies $\{x_i, x_j\}$ is an edge of G if and only if the corresponding Voronoi cells satisfy $R_i \cap R_j \neq \emptyset$.*

Thus, two data points in the Delaunay graph will be connected if their Voronoi cells are "adjacent" to each other. Here, adjacency means their boundary has a nonempty intersection. It is worth noting that, the Voronoi diagram is built on Y, while the Delaunay graph is constructed with vertices S on X. This generalized Delaunay graph to any binary function f. As proved in previous work [31], searching on such Delaunay graph achieves global optimum by greedy search, while its construction and approximation are difficult in general, especially for complicated f.

Example 1. *Let $f : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ be negative ℓ_2 distance, i.e., $f(x, y) = -\|x - y\|$, then the Delaunay graph defined in Definition 3 coincides with classical definition, for instance, the one in [24]. In this case, Delaunay graph refers to ℓ^2-Delaunay graph.*

The key factor of SL2G is not only the feasibility of Delaunay graph building but also the effectiveness in solving the OBFS problem. We will show that when the (finite) dataset S is sufficiently dense in the certain region, the performance of greedy search on Delaunay graph is similar to optimizing (1) by "coordinate" descent in Euclidean space, where the direction "coordinates" are the incident edges of the data points. The analysis will focus on f with certain convex assumptions. Generic non-convex f will be demonstrated empirically in Section 5. Next, Lemma 1 introduces a key property of Delaunay graph, equivalent to *empty sphere criterion* [13].

Lemma 1. *Let B be an open ball in $\mathbb{R}^d$, $x \in \partial B$ (i.e., x is on the boundary of B), and $T := S \cap B \neq \emptyset$ (i.e., there exists at least one data point in ball B), then the Delaunay graph with respect to S connects x with at least one point in T.*

Proof. We consider an internally tangent spheres $\partial B'$ of ∂B that has common external intersect at x such that there exists $y \in S \cap \partial B'$ and $S \cap B' = \emptyset$. Let z be the center of B', then the Voronoi cells R_x of x and R_y of y intersects at z. By Definition 3, x and y are connected in the ℓ^2-Delaunay Graph (see Figure 2(a)). □

Equipped with Lemma 1, we now analyze the conditions under which SL2G returns local optimum. Our analysis relies on distribution of the data points. Generally speaking, the dataset needs to be dense enough on a compact region. We simplify our assumptions by considering the region $[0, 1]^d$.

Lemma 2. *Let p be any density function on $[0, 1]^d$ such that $\inf_{x \in [0,1]^d} p(x) \geq \lambda > 0$. Let S be i.i.d. samples following density p, then every d-balls with radius r in $[0, 1]^d$ contains at least one data point with probability at least*

$$1 - \lambda \exp\left(-n\left(\frac{r}{\sqrt{d}}\right)^d + d \log \frac{\sqrt{d}}{r}\right). \tag{4}$$

Proof. We divide $[0, 1]^d$ into d-cubes with diameter r, then the side lengths of these cubes are $l = r/\sqrt{d}$, so their volumes are $(r/\sqrt{d})^d$. Hence the probability that a single cube does not contains any data points is bounded by

$$\lambda\left(1 - \left(\frac{r}{\sqrt{d}}\right)^d\right)^n \leq \lambda \exp\left(-n\left(\frac{r}{\sqrt{d}}\right)^d\right).$$

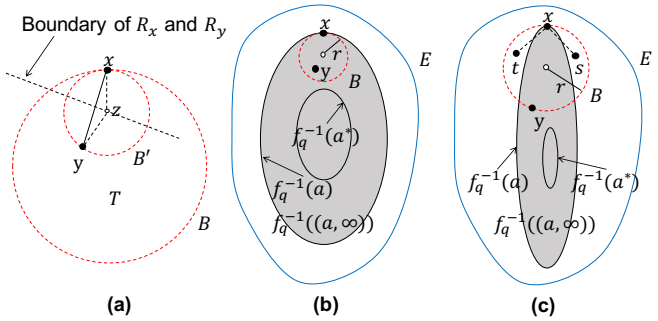

Figure 2: (a) is the schematic diagram for the proof of Lemma 1. (b) and (c) are for the proof of Theorem 1. (b) is for the assumptions of the theorem are satisfied. While if the shape f is strange and the data is not dense enough, the theorem will not hold as shown in (c). In (c), there are two data points t and s may take place y and connect with x. Then the optimization will stop at x.

The volume of $[0,1]^d$ is 1, so there are $(\sqrt{d}/r)^d$ such cubes. Therefore, with probability at least

$$1-\lambda\left(\frac{\sqrt{d}}{r}\right)^d \exp\left(-n\left(\frac{r}{\sqrt{d}}\right)^d\right) = 1-\lambda\exp\left(-n\left(\frac{r}{\sqrt{d}}\right)^d + d\log\frac{\sqrt{d}}{r}\right),$$

every cube contains at least a data point. In this case, every d-ball with radius r includes a cube with diameter r, so it also contains a data point in S. □

For fixed $q \in Y$, we let $f_q(x) := f(x,q)$. We want to show that search on graph can at least retrieve $x_i \in S$ around local optimum of f_q. For simplicity, we consider concave f_q defined on $[0,1]^d$.

Theorem 1. *Suppose S is generated i.i.d. uniformly from $[0,1]^d$ and f_q be a concave function defined on $[0,1]^d$. We assume there exists $r>0$ and $a^* < \max_{x\in[0,1]^d} f_q(x)$ such that for every $a \le a^*$, $f_q^{-1}(a)$ is a smooth manifold with radius of curvature at least r for every point on the manifold. Then with probability no smaller than the one in* (4)*, the greedy search on the Delaunay graph with respect to S achieves value at least a^*.*

Proof. Consider an $x \in S$ such that $f_q(x) = a \le a^*$. By the radius of curvature assumption and convexity of the superlevel set $f_q^{-1}((a,\infty))$, there exists a r-ball $B \subset f_q^{-1}((a,\infty))$ with $x \in \partial B$. See Figure 2 (b). By Lemma 2, with probability at least p, there exists at least one point in S belongs B. By Lemma 1, x connects with at least one point in B, say y. Since $y \in B \subset f_q^{-1}((a,\infty))$, $f_q(y) > a = f_q(x)$. Hence greedy search found a data point with higher evaluation. This procedure only stops after it achieves a value greater than a^*. □

In Figure 2 (b), suppose the current candidate is x. Since Delaunay graph has an edge $\{x,y\}$ and $f_q(y) > f_q(x)$, greedy search on graph will discover y and update y as the new candidate. In the case of Figure 2 (c), assumption of Theorem 1 is not satisfied. In that figure, x is only connected to t and s, but $f_q(t), f_q(s) < f_q(x)$, so greedy search will stop at x. Note that, searching for such a local optimum only require edges on Delaunay graph with length less than r, so a complete Delaunay graph is not necessary. This property allows us to only find short edges of the graph. We can also weaken concavity of f_q to quasi-concavity as long as other assumptions still hold.

Given fixed r and d, as the size of dataset S increases, i.e., $n \to \infty$, the failing probability in (4) converges to 0 exponentially, so the greedy search succeeds to reach some value close to local optimum a^*. In an asymptotic setting, i.e., r and d depend on n, it requires $n\left(\frac{r}{\sqrt{d}}\right)^d \to \infty$ to ensure failing probability converge to 0. In this setting, $r \asymp \sqrt{d}$ is a natural assumption since diameter of $[0,1]^d$ is $\sqrt{d}$.

4.3 Implementation and Algorithms

In practice, constructing a perfect Delaunay graph (with respect to any measure) for large-scale high dimensional data is computationally infeasible. As mentioned in Section 3.3, there are previous works that tried to construct approximate Delaunay graphs. To meet the efficient searching purpose, the vertex degrees of these graphs are often restricted to low values, which sacrifices the properties of the graph. There are no theoretical guarantees that search on graph methods will return exact optimal results via Delaunay graph approximations. According to empirical experiments, the effectiveness of approximate Delaunay graphs with respect to ℓ_2 distance was empirically proven by previous works [17, 29]. However, for generic binary function f's beyond ℓ_2 distance, such as asymmetric f's or neural network based f's, approximate and efficient Delaunay graph constructing is still an open problem. Nevertheless, SL2G only requires Delaunay graphs with respect to ℓ_2 distance, which breaks through the format constraints of searching measure f.

Algorithm 1 Building Index Graphs for SL2G

```
1:  Input: the data set S, the maximum vertex degree M, and the
    search depth k.
2:  Initialize graph G = ∅.
3:  for x_i ∈ S do
4:      A ← Search_on_Graph(x_i, G, k, −ℓ_2).
5:      if |A| ≤ M then
6:          Connect x_i to vertices in A.
7:      else
8:          m ← |A|. B ← ∅.
9:          Order y_j ∈ A in descending order of −ℓ_2(x_i, y_j).
10:         for j = 1 to m do
11:             if ‖x_i − y_j‖ ≤ min_{z∈B} ‖z − y_j‖ then
12:                 B ← B ∪ {y_j}.
13:             if |B| = M then
14:                 Break.
15:         Connect x_i to vertices in B.
16: Output: G.
```

In this paper, we extend the implementation of HNSW [29] (and SONG [48]) to SL2G, for two reasons: (i) HNSW can approximate proper Delaunay graph with respect to ℓ_2 distance. (ii) HNSW approximates both Small World graph and Delaunay graph. The long-range edges of Small World graphs would be helpful in avoiding local optima, which SL2G may suffer from. The graph construction algorithm for SL2G is shown in Algorithm 1. The edge selection method represented in Line 7-15 is from the original HNSW method, which was shown that it improves the performance greatly in the trade-off of Recall vs. Time. No matter what binary function f we focus on, SL2G constructs graphs by negative ℓ_2 distance. The greedy search algorithm of SL2G is similar to the original HNSW

Algorithm 2 Search_on_Graph (q, G, k, f)

Input: the query element q, the graph $G = (V, E)$, the search depth k, and the searching measure f.
Randomly choose a vertex $p \in V$. $A \leftarrow \{p\}$.
Set p as checked and the rest of vertices as unchecked.
while A does not converge **do**
    Add unchecked neighbors of vertices in A to A.
    Set vertices in A as checked.
    $A \leftarrow$ top-k elements of $v \in A$ in descending order of $f(v, q)$.
Output: A.

but replacing the metric searching measures by the focus binary function f, which is defined on the user-item pair in searching. Details for greedy search on graph, for both graph construction and query searching, can be found in Algorithm 2. Note that, to simplify the description by pseudo codes, we only use one set A to store the current best results. In practice, two priority queues will be used to achieve the same purpose but more efficient.

4.4 From Local Optimum to Global Optimum

As analyzed in theoretical part, SL2G guarantees to reach an approximate local optimum. But many of generic ranking measures are non-convex, with multiple local optimums. How to avoid local optimum and reach the approximate global optimum is challenging.

Fortunately, there are two features will help SL2G to avoid local optimum. The first one is, in practical implementations, we record multiple current best candidates but not only one as shown in Algorithm 2. Based on this setting, multiple searching paths will be scanned instead of only one path. For example, we now check the top candidate a in A. Supposed that a is a local optimum and all its neighbors are worse than a. If we only keep one current best result, we will stop at a, where is a local optimum. Luckily, we have multiple candidates in A and the stop condition is whether A is converged. We will put all unvisited neighbors of a to A and produce the new best candidate. In the next step, we will keep checking the new best candidate in A until A converges. In this way, the algorithm will not stuck at the local optimum.

The second feature is that HNSW approximates both Small World graph and Delaunay graph. The long-range edges of Small World graphs would be helpful in avoiding local optima. As can be seen in the experiments, SL2G works well to reach the global optimum.

5 EXPERIMENTS

In this section, we evaluate the performance of SL2G in fast ranking under generic ranking measures. Specifically, we explore two neural network based ranking measures for the recommendation problem. Note that the comparison of various recommendation models is beyond the scope of this paper. We suppose that the neural network based ranking measures (i.e., f's) are pre-trained, and we only focus on the fast item ranking procedure.

Datasets. For experimental datasets, we choose two widely used datasets for recommendation: **Yelp**[1] and Amazon Movie (**Amovie**)[2]. For Yelp, we did not further filter it since it was processed before. It

[1]https://www.Yelp.com/dataset/challenge
[2]http://jmcauley.ucsd.edu/data/amazon

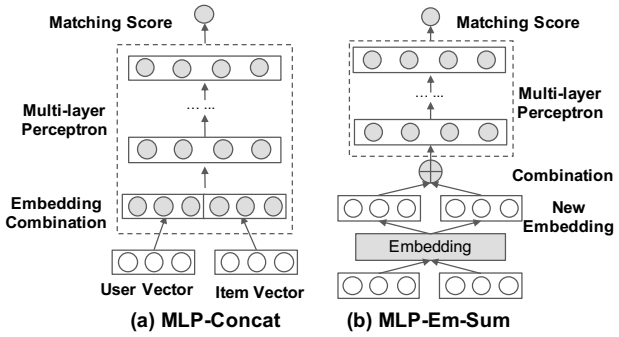

Figure 3: Two variants of the MLP model for recommendation. (a) MLP-Concat; (b) MLP-Em-Sum.

contains 25,677 users, 25,815 items and 731,670 ratings. For Amovie, we filtered the dataset in the way that users with at least 30 interactions are retained. At last Amovie contains 7,748 users, 104,708 items and 746,397 ratings. The item amount in these two datasets is much bigger than that in others, which is more appropriate for searching efficiency exploration.

5.1 Neural Network based Ranking Measures

To evaluate the performance of SL2G on neural network based searching measures, we leverage a state-of-the-art neural network based recommendation method, **MLP**, which was introduced in [18]. The original MLP model concatenates user latent vectors and item latent vectors before going through the Multi-Layer Perceptron network (referred as **MLP-Concate**, Figure 3 (a)). The concatenating operation is asymmetric. We design one more variant of MLP with symmetric operations on the input vectors: **MLP-Em-Sum** (Figure 3 (b)). MLP-Em-Sum transforms two kinds of vectors into a common space by an additional embedding layer, before the merge operation. In this way, user vectors and item vectors will lie on the same manifold on which element-wise sum will be operated. After training, we obtain NN models with fixed weights. These fixed weight models can be considered as binary function f's. For each pair of input vectors, (user, item), the binary function f will output a real number. These model-based binary functions are referred as $f_{\text{MLP-Concate}}$ and $f_{\text{MLP-Em-Sum}}$. Note that, for both models, only the networks above the vector combination (i.e., the Multi-Layer Perceptron networks within dashed boxes shown as Figure 3) are regarded as the matching function. The part from raw data to the embedding layer does not belong to the matching function. For implementation, after weights of each model is learned, we re-implement and integrate the model-based matching functions into the searching framework. The dimensions of input vectors for MLP-Em-Sum are set as 64 and for MLP-Concate are set as 32.

5.2 Baselines

Since this is the first work for fast ranking under neural network based measures and there are few previous comparable algorithms. To the best of our knowledge, SL2G is the only formal solution until now. For baselines, we adopt two state-of-the-art methods for traditional ANN search: **ANNOY**[3] and **HNSW** [29]. ANNOY is a popular open source ANN search project which mainly focuses on

[3]https://github.com/spotify/annoy

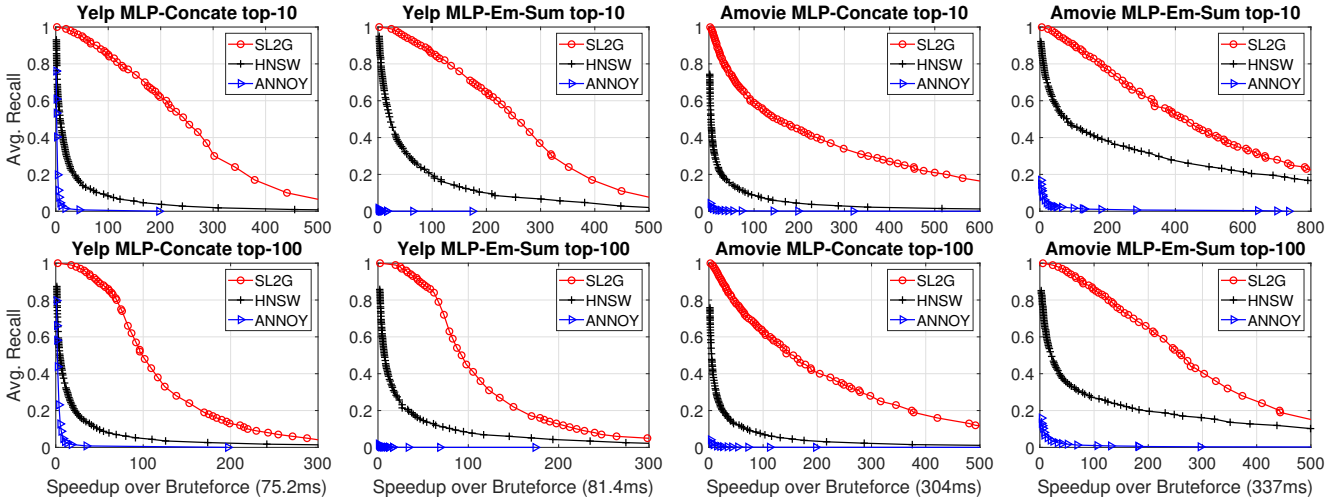

Figure 4: Experimental results for efficient ranking by neural network based ranking measures. The evaluation method is Recall vs. Speedup over Bruteforce (i.e., time). The best results are in the upper right corner.

metric measures. For the fast neural ranking task, we first retrieve candidates by ANNOY via ℓ_2 distance (here user vectors and item vectors are required to be the same dimension) and then re-rank the candidates by the focus ranking measure f. For HNSW, we adjust it for complicated binary functions as below: we input a pair of item vectors to the binary function forcibly as $f(x, x)$. Then we use this $f(x, x)$ as relevance measure to construct the index graph by the algorithm for HNSW. The graph built like this may be dramatically different from the Delaunay graph with respect to $f(x, q)$. So there is no performance guarantee. Note that we restrict dimensions for users and items the same for MLP-Concrete as mentioned above. In this way, HNSW can be implemented. Otherwise, if the dimensions of users and items are different, $f(x, x)$ is invalid. Obviously, SL2G has no such limitations. In the algorithm level, HNSW shares most of contents with SL2G as represented in Algorithms 1 and 2. But HNSW uses f to the constructed graph but not negative ℓ_2 distance.

5.3 Experimental Settings

To generate evaluating labels, we calculate most relevant items for each user by the corresponding learned binary function f, i.e., $f_{\text{MLP-Concate}}$ and $f_{\text{MLP-Em-Sum}}$. Experiments on top-10 and 100 labels are recorded. We do not test recommendation precisions by true labels, as we focus on searching efficiency of neural ranking tasks.

There are two popular ways to evaluate ANN search algorithms (or generally OBFS methods): (a) **Recall vs. Time**; (b) **Recall vs. Computations**. Recall vs. Time reports how many times the algorithm can speed up over naive brute force scanning at each recall level. Recall vs. Computations reports the amount/percentage of pair-wise computations that the ANN Search algorithm costs at each recall level. We will show both of these perspectives in the following experiments for a comprehensive evaluation.

All methods have parameters. ANNOY has one key parameter which is the number of index trees. SL2G and HNSW have three parameters: M, $k_{\text{construction}}$ and k_{search}, which control the degrees of vertices and the number of search attempts. 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 largest times speed up over the brute force method or the lowest percentage of pair-wise computations) is chosen. If there are no data points in the bucket, the bucket will be ignored. In this way, we shall have multiple pairs of data for drawing curves.

All time-related experiments were performed on a 2X 3.00 GHz 8-core i7-5960X CPU server with 32GB memory.

5.4 Experimental Results

Results for Recall vs. Time and Recall vs. Computations are shown in Figure 4 and Figure 5 respectively. For Recall vs. Time, results for top-10 and top-100 labels are represented, while for Recall vs. Computations, only results for top-10 are shown due to the limited space. As can be seen, ANNOY which is designed for metric measures work badly for the challenging task of fast neural ranking. We will pay main attention to compare the other two methods, HNSW and SL2G. Let us first look at the results for the symmetric $f_{\text{MLP-Em-Sum}}$. As can be seen, HNSW can speed up over the brute force ranking in some lower recall levels but it works much worse than the proposed SL2G. SL2G speeds up the searching hundreds of times than brute force ranking, with high recalls. The reason is that HNSW cannot approximate proper Delaunay graph with respect to neural network based measures. For the asymmetric $f_{\text{MLP-Concate}}$, the superiority of SL2G over HNSW is more significant. We can see that, it is difficult for HNSW to achieve 80% recalls. The asymmetric characteristic poses even more challenges for HNSW. Constructing Delaunay graph with respect to asymmetric binary functions is extremely hard. But SL2G, which is proposed for generic fast ranking and only requires to build Delaunay graph with respect to ℓ_2 distance, can overcome the limitations naturally. As can been seen, SL2G works consistently well both on $f_{\text{MLP-Em-Sum}}$ and $f_{\text{MLP-Concate}}$.

As analyzed above, if there are multiple local optimal candidates, it would be challenging for SL2G. It is the common case for neural

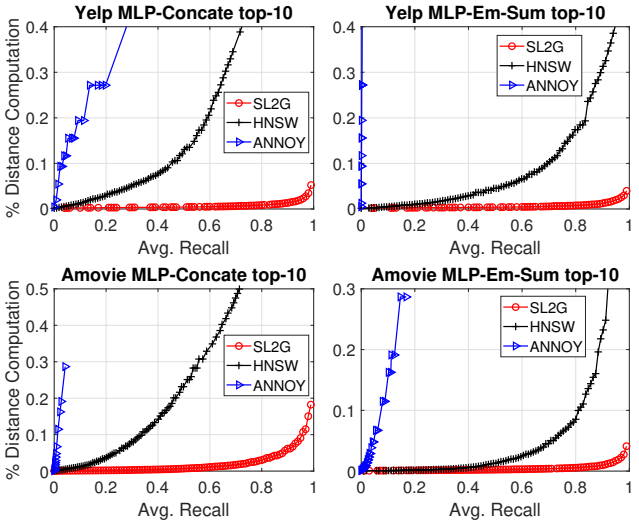

Figure 5: Experimental results for Recall vs. Computation. The best results are in the lower right corner.

network based searching measures, such as one person may be interested in multiple types of movies. But as can be seen, SL2G does not stop at local optimum and achieves higher recalls easily. Reasons can be found in Section 4.4.

5.5 Scalability of SL2G

To test the scalability of the proposed method, we evaluate its performance on a simulation dataset based on Yelp MLP-Concate. For each data point of Yelp MLP-Concate, we randomly generate 40 simulation data points by Gaussian distribution with the original data as mean and 0.1 as the standard deviation. In this way we get a dataset with 1,058,415 data points (including the original ones). This dataset is referred as Yelp1M MLP-Concate. The results of it are shown in Figure 6. As can be seen, the proposed method, SL2G, shows good scalability on this larger dataset. For example, to get 60% top-100 recall, SL2G achieves less than 100 times speedup on the original smaller dataset, Yelp MLP-Concate. While on this larger dataset, SL2G is 1887 times faster than brute force ranking. The two baselines do not show good scalability on this dataset.

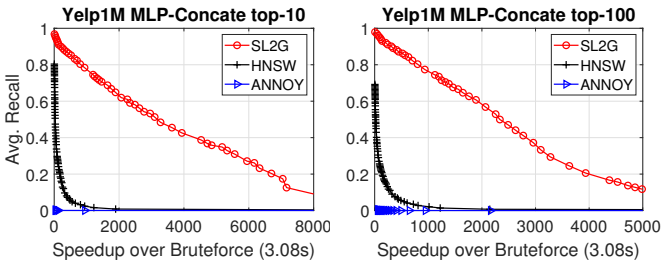

Figure 6: Results on the larger dataset Yelp1M MLP Concate.

5.6 Graph Construction Time Analysis

The graph construction for SL2G (with respect to ℓ_2 distance computations) is much more efficient than HNSW (with respect to complicate f computations). The time analysis is listed in Table 1.

Table 1: Graph Construction Time (in seconds) Analysis for $M = 16$, $k_{\text{construction}} = 100$ and 16 parallel threads.

Datasets	Time of HNSW	Time of SL2G
Yelp MLP-Concate	50.22	0.24
Yelp MLP-Em-Sum	75.04	0.27
Amovie MLP-Concate	251.89	1.03
Amovie MLP-Em-Sum	285.65	1.28
Yelp1M MLP-Concate	25058.43	64.13

In this experiments, we fix the priority queue size as 100 in graph construction and set the maximum outgoing degree as 16. On our server, 16 parallel threads work at the same time to insert nodes. As shown the construction time of SL2G is much less than HNSW, more than 200 times faster. For example, for the case of Yelp and $f_{\text{MLP-EM-Sum}}$, HNSW takes 75.04 seconds to construct the graph but SL2G only takes 0.27 seconds, 0.36% of the former one.

5.7 Implementation by SONG

To exclude bias from implementation, we also implement SL2G and HNSW by another search on graph platform, SONG [48]. The results are shown in Figure 7. Due to the limited space, only results for Amovie MLP-Concate are shown. Results for other cases have similar trends. As can be seen, the implementation of SONG is more efficient, both for SL2G and HNSW, but their priority order keeps the same. SL2G works much better than HNSW under both two implementations.

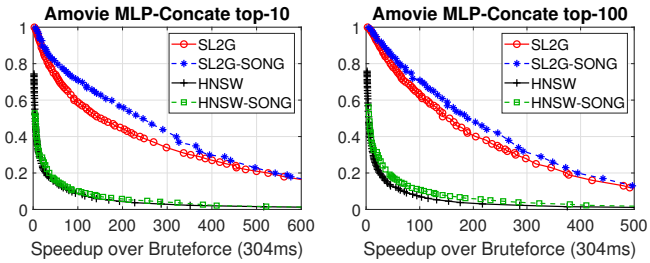

Figure 7: Implementing SL2G and HNSW by SONG [48].

6 CONCLUSION

In this paper, we define a vital task, generic fast ranking, formally Optimal Binary Function Search (OBFS). A graph-based methodology, Binary Function Search on Graph (BFSG), is introduced for OBFS. Algorithmically, we proposed a specific method, Search on L2 Graph (SL2G) under BFSG. Approximate ranking by generic relevance measures, such as neural network based ones, can be sped up significantly by the proposed method. Theoretically, SL2G approximates coordinate descent in Euclidean space. Even for non-convex matching measures, SL2G guarantees that at least an approximate local optimum can be found based on some weak assumptions. In experiments, two neural networks for recommendation are selected as the ranking measures to evaluate SL2G. Results show that SL2G can speed up the searching efficiency hundreds of times, comparing with the brute force ranking.

REFERENCES

[1] Franz Aurenhammer. 1991. Voronoi diagrams - a survey of a fundamental geometric data structure. *ACM Computing Surveys (CSUR)* 23, 3 (1991), 345–405.

[2] Yoram Bachrach, Yehuda Finkelstein, Ran Gilad-Bachrach, Liran Katzir, Noam Koenigstein, Nir Nice, and Ulrich Paquet. 2014. Speeding up the xbox recommender system using a Euclidean transformation for inner-product spaces. In *Eighth ACM Conference on Recommender Systems (RecSys)*. Foster City, CA, 257–264.

[3] Andrei Z. Broder. 1997. On the Resemblance and Containment of Documents. In *the Compression and Complexity of Sequences*. Positano, Italy, 21–29.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] Steven Fortune. 2004. Voronoi diagrams and Delaunay triangulations. In *Handbook of Discrete and Computational Geometry, Second Edition*. 513–528.

[11] Jerome H. Friedman, F. Baskett, and L. Shustek. 1975. An Algorithm for finding nearest neighbors. *IEEE Trans. Comput.* 24 (1975), 1000–1006.

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

[13] Paul-Louis George and Houman Borouchaki. 1998. Delaunay triangulation and meshing. (1998).

[14] 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.

[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. 2019. A deep look into neural ranking models for information retrieval. *arXiv preprint arXiv:1903.06902* (2019).

[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)*, Vol. 22. Barcelona, Spain, 1312.

[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] Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry P. Heck. 2013. Learning deep structured semantic models for web search using clickthrough data. In *22nd ACM International Conference on Information and Knowledge Management (CIKM)*. San Francisco, CA, 2333–2338.

[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] 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)*. Marseille, France, 304–317.

[22] Herve Jegou, Matthijs Douze, and Cordelia Schmid. 2011. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence* 33, 1 (2011), 117–128.

[23] Jon M. Kleinberg. 2000. The small-world phenomenon: an algorithmic perspective. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing (STOC)*. Portland, OR, 163–170.

[24] Der-Tsai Lee and Bruce J Schachter. 1980. Two algorithms for constructing a Delaunay triangulation. *International Journal of Computer & Information Sciences* 9, 3 (1980), 219–242.

[25] 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, NS, Canada, 315–324.

[26] Ping Li, Art B Owen, and Cun-Hui Zhang. 2012. One Permutation Hashing. In *Advances in Neural Information Processing Systems (NIPS)*. Lake Tahoe, NV, 3122–3130.

[27] Ping Li, Gennady Samorodnitsky, and John Hopcroft. 2013. Sign Cauchy Projections and Chi-Square Kernel. In *Advances in Neural Information Processing Systems (NIPS)*. Lake Tahoe, NV, 2571–2579.

[28] 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.

[29] Yury A Malkov and Dmitry A Yashunin. Early Access. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence* (Early Access).

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

[31] 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.

[32] Jonas Mueller and Aditya Thyagarajan. 2016. Siamese Recurrent Architectures for Learning Sentence Similarity. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI)*. Phoenix, AZ, 2786–2792.

[33] Parikshit Ram and Alexander G. Gray. 2012. Maximum inner-product search using cone trees. In *The 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. Beijing, China, 931–939.

[34] 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.

[35] Yelong Shen, Xiaodong He, Jianfeng Gao, Li Deng, and Grégoire Mesnil. 2014. Learning semantic representations using convolutional neural networks for web search. In *Proceedings of the 23rd International World Wide Web Conference (WWW)*. Seoul, Korea, 373–374.

[36] Anshumali Shrivastava and Ping Li. 2014. Asymmetric LSH (ALSH) for Sublinear Time Maximum Inner Product Search (MIPS). In *Advances in Neural Information Processing Systems (NIPS)*. Montreal, Canada, 2321–2329.

[37] Anshumali Shrivastava and Ping Li. 2015. Asymmetric Minwise Hashing for Indexing Binary Inner Products and Set Containment. In *Proceedings of the 24th International Conference on World Wide Web (WWW)*. Florence, Italy, 981–991.

[38] Shulong Tan, Zhixin Zhou, Zhaozhuo Xu, and Ping Li. 2019. On Efficient Retrieval of Top Similarity Vectors. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 5235–5245.

[39] 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.

[40] 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.

[41] 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 *International Conference on Management of Data (SIGMOD)*. Snowbird, UT, 1139–1150.

[42] 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.

[43] Jun Xu, Xiangnan He, and Hang Li. 2018. Deep Learning for Matching in Search and Recommendation. In *WWW Tutorials*.

[44] Hong-Jian Xue, Xin-Yu Dai, Jianbing Zhang, Shujian Huang, and Jiajun Chen. 2017. Deep matrix factorization models for recommender systems. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence (IJCAI)*. Melbourne, Australia, 3203–3209.

[45] Xiao Yan, Jinfeng Li, Xinyan Dai, Hongzhi Chen, and James Cheng. 2018. Norm-Ranging LSH for Maximum Inner Product Search. In *Advances in Neural Information Processing Systems (NeurIPS)*. Montreal, Canada, 2956–2965.

[46] Hsiang-Fu Yu, Cho-Jui Hsieh, Qi Lei, and Inderjit S Dhillon. 2017. A Greedy Approach for Budgeted Maximum Inner Product Search. In *Advances in Neural Information Processing Systems (NIPS)*. Long Beach, CA, 5453–5462.

[47] Hamed Zamani, Mostafa Dehghani, W. Bruce Croft, Erik G. Learned-Miller, and Jaap Kamps. 2018. From Neural Re-Ranking to Neural Ranking: Learning a Sparse Representation for Inverted Indexing. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management (CIKM)*. 497–506.

[48] Weijie Zhao, Shulong Tan, and Ping Li. 2020. SONG: Approximate Nearest Neighbor Search on GPU. In *35th IEEE International Conference on Data Engineering (ICDE)*. Dallas, TX.

[49] 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.