Being prepared in a sparse world: the case of KNN graph construction

Antoine Boutet
CNRS, Laboratoire Hubert Curien
Saint-Etienne, France
antoine.boutet@telecom-st-etienne.fr

Anne-Marie Kermarrec
INRIA
Rennes, France
anne-marie.kermarrec@inria.fr

Nupur Mittal
INRIA
Rennes, France
nupur.mittal@inria.fr

Francois Taiani
University of Rennes 1
Rennes, France
francois.taiani@irisa.fr

***Abstract*—K-Nearest-Neighbor (KNN) graphs have emerged as a fundamental building block of many on-line services providing recommendation, similarity search and classification. Constructing a KNN graph rapidly and accurately is, however, a computationally intensive task. As data volumes keep growing, speed and the ability to scale out are becoming critical factors when deploying a KNN algorithm. In this work, we present *KIFF*, a generic, fast and scalable KNN graph construction algorithm. *KIFF* directly exploits the bipartite nature of most datasets to which KNN algorithms are applied. This simple but powerful strategy drastically limits the computational cost required to rapidly converge to an accurate KNN solution, especially for sparse datasets. Our evaluation on a representative range of datasets show that *KIFF* provides, on average, a speed-up factor of 14 against recent state-of-the art solutions while improving the quality of the KNN approximation by 18%.**

I. Introduction

K-Nearest-Neighbor (KNN) graphs play a fundamental role in many web-based applications e.g., search [1], [2], recommendation [3], [4], [5], [6] and classification [7]. A KNN graph is a directed graph of entities (e.g., users, products, services, documents etc.), in which each entity (or *node*) is connected to its k most similar counterparts or *neighbors*, according to a given *similarity metric*. In a large number of applications, this similarity metric is computed from a second set of entities (termed *items*) associated with each node in a bipartite graph (possibly extended with weights, such as ratings or frequencies). For instance, in a movie rating database, nodes are users, and each user is associated with the movies (items) she has already rated.

As data volumes continue to grow, constructing a KNN graph efficiently remains an ongoing and open challenge[1]. A brute force computation of a KNN graph requires $O(n^2)$ similarity computations, which does not scale, in particular when nodes are associated with large item sets. To address this problem, most practical approaches approximate the actual KNN and deliver an Approximate-Nearest-Neighbor graph, or ANN. This approximation can take several forms, which can often be combined. One such line of attack uses dimension reduction techniques to lower the cost of computing individual similarities [8], [9], [10].

Another complementary strategy avoids an exhaustive search altogether and executes a greedy, localized search to converge rapidly towards a KNN or ANN graph [1], [4], [11], [12], [13], [14]. These greedy approaches start from a random k-degree graph and exploit the fact that similarity functions tend to behave transitively over neighbors-of-neighbors links: if A is similar to B, and B to C, then A is likely to be similar to C. This greedy procedure is often completed by a low-probability random exploration of arbitrary nodes to avoid local minima [1], [4], [11], [15].

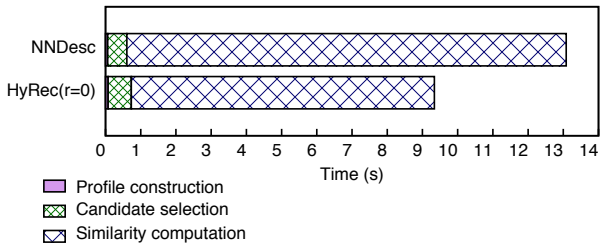


Fig. 1. State-of-the art greedy KNN-graph approaches spend over 90% of their computation time computing similarity values (Wikipedia dataset).

Some greedy KNN approaches have recently been found to perform particularly well [4], [13]. These approaches tend, however, to induce a substantial number of similarity computations, which directly impact their computing time. This point is illustrated in Figure 1 for two characteristic greedy KNN-graph algorithms, NNDescent [13], and HyRec [4]. The figure shows the breakdown, in seconds, of the computing time of both algorithms on a small Wikipedia dataset [16] (see Sec. V) using the standard cosine similarity metric. On average, both approaches spend more than 90% of their total execution time repeatedly computing the similarities values (large checkered bars in the figure).

In this paper, we propose to reduce this substantial cost by exploiting two observations that apply to most item-based similarity metrics: first, two nodes must usually share some items to be KNN neighbors; second, the more items two nodes have in common, the more likely they are to appear in each other's KNN neighborhood. Based on these observations, our solution, *KIFF* (*K-nearest neighbor Impressively Fast and eFficient*) first uses the node-item bipartite graph to compute a coarse but very cheap approximation of the similarity between two users. *KIFF* then uses this rough approximation to orient and to prune the search for a good KNN approximation using a greedy procedure, rather than starting from a random graph as many greedy solutions do [1], [4], [11], [15].

[1]Note that the problem of computing a complete KNN graph (which we address in this paper) is related but different from that of answering a sequence of KNN queries, a point we revisit when discussing related work.

The result is a novel, fast and scalable KNN graph construction algorithm that produces a very close approximation of the real KNN. More precisely, we make the following contributions:

- We motivate and present *KIFF*, a scalable, efficient, and accurate algorithm to compute approximate KNN graphs. *KIFF* is designed for datasets in which nodes are associated to items, and similarity is computed on the basis of these items. *KIFF* is generic, in the sense that it can be applied to any kind of nodes, items, or similarity metrics.
- We extensively evaluate *KIFF* on 4 representative datasets (from various domains and with different density values), and show that *KIFF* reduces the computational time by a speed-up factor of 14 on average, while improving the quality of the KNN approximation on average by 18% compared to recent state-of-the art solutions [13], [4], regardless of the value of k.
- We assess the impact of graph density on *KIFF*'s performance, and show that *KIFF*'s performance is strongly correlated to a dataset's density.

The rest of the paper is organized as follows. In Section II, we motivate and explain the main intuition behind *KIFF*, along with some background information. In Section III, we present the working of *KIFF* in detail. Our evaluation of *KIFF* follows in Sections IV and V. Section VI discusses related works, and Section VII concludes.

II. Intuition and Overview

Greedy KNN-graph approaches have been shown to offer a promising alternative to more traditional KNN-graph construction techniques, for instance based on Locality-Sensitive Hashing (LSH) or Recursive Lanczos Bisection (RLB) [17], [9], [10], [13]. Greedy approaches incrementally improve an approximation of the KNN graph, and thus avoid an exhaustive $O(n^2)$ search among potential KNN edges. They perform well for two main reasons: they are inherently parallel, and show a strong memory locality. As shown in Figure 1, they tend, however, to induce a substantial number of similarity computations, which directly impact their computing time.

A. Intuition

We propose to reduce this time substantially by limiting the number of similarity values a greedy approach needs to compute. Our intuition is based on two simple observations: (i) almost all similarity metrics used to construct KNN graphs (cosine, Jaccard's coefficient, Adamic-Adar's coefficient) return zero when two users[2] do not share any items ; and (ii) these metrics usually contain a term that grows with the number of items two users have in common.

Counting common items is typically much cheaper computationally than computing full blown similarity metrics. This is because (i) common items are usually a first step of more advanced similarity metrics ; and (ii) most similarity metrics use floating point operations (divisions, square roots, logarithms) that are much more expensive than the simple integer arithmetic involved in counting items. Our approach, therefore, employs a simple but powerful strategy: we use common item counts as a first coarse approximation of the similarity between two users to prune the pairs of users who have no items in common. We then refine this approximation by iterating over potential KNN neighbors in reverse order of this item count.

[2]For ease of exposition, we will assume for the rest of the paper that nodes are users who have rated items. Our approach can, however, be applied to any type of nodes, as long as node similarity is computed from items associated with these nodes.

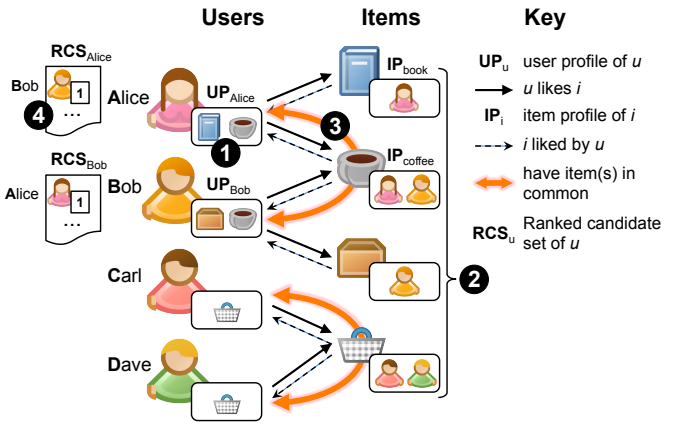


Fig. 2. *KIFF* exploits the bipartite nature of user-item datasets to pre-compute *Ranked Candidate Sets* (noted RCS_u).

Directly comparing every pair of users to count common items would, however, scale poorly. We, therefore, use an indirect method: we exploit the bipartite nature of the user-item graph (users are linked to items, and items to users) as an efficient structure to construct lists of users sharing items.

In depth, our approach (called *KIFF*) works in two phases, called the *counting* and the *refinement* phase. In the counting phase, *KIFF* preprocesses the user-item bipartite graph and builds a *Ranked Candidate Set* (RCS) for each user. Ranked candidate sets are ordered weighted sets that bring together users who share items, while counting how many items these users have in common. Ranked candidate sets are used in the refinement phase to initiate, and then iteratively refine the KNN approximation of each user. We describe both phases informally in the following on a small example, before providing a more formal presentation in the next section.

B. Counting Phase

The *Users* column of Figure 2 represents a toy user-item dataset in which the users of an online social network are associated with the objects or activities that they have liked (the items). Alice likes books and coffee, Bob coffee and cheese, and Carl and Dave like shopping. The items a user likes make up this user's *profile*. Here, Alice's profile (UP_{Alice}) contains $\{\text{book}, \text{coffee}\}$ (Label **1**).

The role of the counting phase is to compute the ranked candidate set (RCS) of each user. Because Alice and Bob share one common item (coffee), *KIFF* should add each to the RCS of the other with a count of 1[3]. To do so, *KIFF* first computes the item profiles (IP_i) of each item (Label **2**), i.e., the set of

[3]We explain later on how we avoid the redundancy caused by this symmetry.

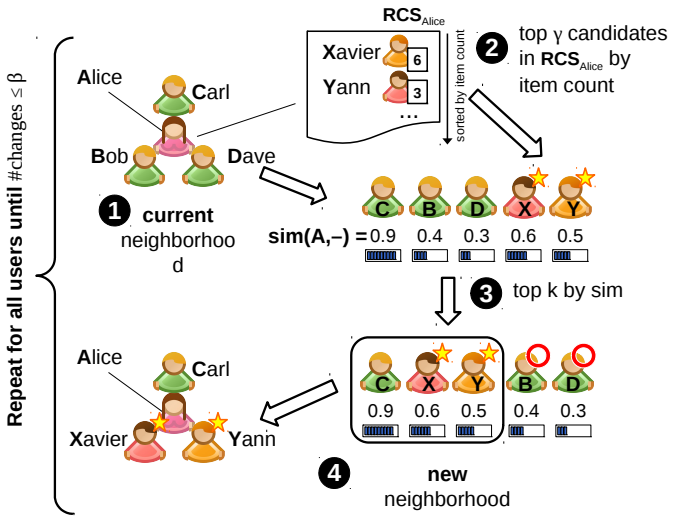


Fig. 3. The pre-computed *Ranked Candidate Sets* are used in an iterative greedy process to construct an approximation of the KNN.

users who have liked a particular item i. For instance, here, the item profile of coffee (IP_{coffee}) contains Alice and Bob, while the item profile of book (IP_{book}) contains only Alice. Item profiles reverse the node-to-item links (solid arrows) of the initial dataset into item-to-node edges (dashed arrows). Item profiles also provide a crude hashing procedure, in which users are binned into as many item profiles (which acts as buckets) as the items they possess.

Once the item profiles have been computed, *KIFF* constructs Alice's ranked candidate set (RCS_{Alice}) by navigating the bipartite graph induced by user and item profiles (Label **3**) in order to collect all users who share an item with Alice: here Bob. The count associated with Bob in RCS_{Alice} (Label **4**) is the number of common items between Bob and Alice (1 for coffee), i.e., the number of paths from Alice to Bob in the user-item graph.

C. Refinement Phase

Let's now imagine that the dataset of Figure 2 has been enriched with new items, so that Alice's ranked candidate set ends up containing the following users:

$$RCS_{\text{Alice}} = \{(\text{Bob}, 10), (\text{Carl}, 9), (\text{Dave}, 8), (\text{Xavier}, 6), (\text{Yann}, 3), ...\}$$

KIFF first fills up Alice's neighborhood with the top k users of her RCS, i.e., the k users with whom Alice shares the most items: in our example Bob, Carl, and Dave (if we assume $k = 3$). These k users are removed from the RCS.

KIFF then repeatedly applies the procedure depicted in Figure 3 to iteratively construct an approximation of Alice's KNN. *KIFF* first merges the current neighborhood of Alice ($\{\text{Carl}, \text{Bob}, \text{Dave}\}$, Label **1**) with the current top γ neighbors found in Alice's RCS (RCS_{Alice}, Label **2**). In the example, $\gamma = 2$, but we generally use a γ twice the value of k. Alice's current two top users in her RCS are Xavier with 6 items in common, and Yann with 3 items. Both Xavier and Yann are removed from RCS_{Alice} in the process, and added to Alice's current neighborhood.

The union of Alice's current neighborhood $\{\text{Carl}, \text{Bob}, \text{Dave}\}$, with the γ users extracted from RCS_{Alice} ($\{\text{Xavier}, \text{Yann}\}$) form the potential new neighborhood of Alice in this iteration. From then on, the procedure is very similar to that of other greedy approaches [4], [15], [13], [1], [11]: the potential new neighbors are ranked according to their similarity with Alice (Label **3**), and the top-k users (here Carl, Xavier, and Yann) are selected as Alice's new top-3 neighbors (Label **4**). This whole procedure is repeated for all users until the total number of neighbors changes during an iteration is smaller than a fixed termination parameter β.

D. Optimization and Discussion

In the basic procedure we have described, Ranked Candidate Sets are heavily redundant: if Bob is in Alice's RCS, then Alice is in Bob's. To limit the memory overhead of *KIFF*, and the cost of maintaining individual RCSs, we therefore use a pivot strategy: if Alice and Xavier have items in common, only the user with the lower ID (e.g. Alice) will store the other user in his/her RCS. The procedure of Figure 3 is adapted accordingly, so that, Alice is also considered as a potential neighbor for Xavier (and Yann).

In a second optimization, we simplify the initialization of neighborhoods at the start of the refinement phase: rather than handling this initialization as a special case, we treat it as a standard iteration that starts from empty neighborhoods that are filled up using the procedure of Figure 3.

In terms of computing time, *KIFF* trades off the cost of constructing item profiles and ranked candidate sets (Counting Phase, Figure 2), for the ability to limit similarity computations between users which (i) are guaranteed to have some items in common, and (ii) are considered in the reverse order of their shared item counts. In the example of Figure 3 for instance, Carl and Dave will never be considered as potential neighbors for Alice because they do not share any items. This trade-off pays off if item profiles (IP_{book}, IP_{coffee}) and ranked candidate sets (RCS_{Alice}, RCS_{Bob}) tend to be small.

This advantageous situation corresponds to datasets in which users rate few items from a large item set. These datasets are precisely those problematic for greedy KNN approaches such as HyRec and NN-Descent [4], [13]. That is because greedy approaches start from an initial random graph in which (i) initial neighbors are unlikely to be similar, and (ii) neighbors of neighbors will also tend to show a low similarity, resulting in a slow convergence.

III. The KIFF Algorithm

We now present formally the working of *KIFF*, starting with some notations (Section III-A) and a formal problem definition (Section III-B), before moving on to the algorithm itself (Sections III-C and III-D). As stated earlier, the presentation refers to users and items for the sake of readability, but *KIFF* is generally applicable to any kind of nodes associated with items, and can be applied to datasets containing ratings.

A. Notations

We consider a set of users $U = \{u_1, u_2, ..., u_{|U|}\}$ who have rated a set of items $I = \{i_1, i_2, ..., i_{|I|}\}$, as captured by

a partial rating function, $\rho : U \times I \nrightarrow R$ where R is the set of possible rating values. ρ is only partially defined as each user has only rated a subset of items.

A special case is when ratings are singled-valued, as in Figures 2 and 3. In this case, R is a singleton (e.g., $R = \{1\}$), and the function ρ simply captures whether or not a user u is associated to an item i (i.e., whether i is in her profile): $\rho(u, i)$ is defined if u is associated with i, undefined otherwise.

We denote by UP_u, the *user profile* of a user u. UP_u is a dictionary that associates each item i rated by u to its rating: $\forall i \in \mathrm{dom}(\rho(u, -)) : UP_u(i) = \rho(u, i)$, where dom is the domain of definition of $\rho(u, -)$. UP_u.keys denotes the items rated by user u. For brevity's sake, we will usually write UP_u to mean UP_u.keys when what is meant is clear, e.g., $i \in UP_u$.

The partial rating function ρ defines a *Labeled Bipartite Graph* $G = (V, E, \rho)$. The vertices of this graph are the users and items of the dataset $V = (U \cup I)$. A rating by a user u of an item i is represented in G by an edge $(u, i) \in U \times I$ labeled by $\rho(u, i)$: $E = \{(u, i) | i \in UP_u\}$.

B. Problem definition

Our objective is to approximate a KNN graph over U (noted G_{KNN}) according to some similarity functions sim computed over user profiles:

$$sim : \begin{array}{ccc} U \times U & \rightarrow & \mathbb{R} \\ (u, v) & & sim(u, v) = f_{sim}(UP_u, UP_v) \end{array}$$

where f_{sim} can be any similarity function over dictionaries. We use the cosine similarity in the rest of the paper [18]. Formally, G_{KNN} is a directed graph that connects each user $u \in U$ with a set knn_u of k other users that maximize the similarity function $sim(u, -)$:

$$knn_u \in \underset{v \in U \setminus \{u\}}{\mathrm{argtop}^k} f_{sim}(UP_u, UP_v) \equiv \mathrm{KNN}_u \qquad (1)$$

where argtop^k returns the set of k-tuples of $U \setminus \{u\}$ that maximize the similarity function $sim(u, -)$[4].

Because computing an exact KNN graph over a very large dataset can be computationally expensive, many scalable approaches seek to construct an approximate KNN graph $\widehat{G}_{\mathrm{KNN}}$, i.e., to find for each user u a set $\widehat{knn_u}$ that is as close as possible to an exact KNN set. The quality of this approximation is typically measured in terms of *recall*, i.e., the ratio of exact KNN neighbors in the current KNN approximation. If the exact KNN neighborhood of each user knn_u is unique, the recall for a user u is simply defined as:

$$recall_{\widehat{G}_{\mathrm{KNN}}}(u) = \frac{|\widehat{knn_u} \cap knn_u|}{k} \qquad (2)$$

In the more common case where knn_u is not unique, $recall_{\widehat{G}_{\mathrm{KNN}}}(u)$ becomes

$$recall_{\widehat{G}_{\mathrm{KNN}}}(u) = \max_{knn_u \in \mathrm{KNN}_u} \left(\frac{|\widehat{knn_u} \cap knn_u|}{k} \right) \qquad (3)$$

[4]In other words, argtop^k generalizes the concept of argument of the maximum (argmax for short) to the k top values of a function over a finite discrete set.

where KNN_u denotes the argtop^k expression of Equation (1). (Because $|\widehat{knn_u}| = |knn_u| = k$ in this particular case, *precision*, a second metric commonly considered in retrieval and recommendation problems is equal to recall here and not discussed further.)

This leads us to define the overall recall of an approximate KNN graph $\widehat{G}_{\mathrm{KNN}}$ as the average recall over all users:

$$recall(\widehat{G}_{\mathrm{KNN}}) = \mathop{\mathbb{E}}_{u \in U} recall_{\widehat{G}_{\mathrm{KNN}}}(u) \qquad (4)$$

where $\mathbb{E}$ represents the expectation operator, i.e., the mean of an expression over a given domain.

Given a dataset $D = (U, I, \rho)$ and an item-based similarity function f_{sim}, our goal is to construct $\widehat{G}_{\mathrm{KNN}}$ for D and f_{sim} in the shortest time with the highest recall.

C. KIFF

The pseudo code of *KIFF* is shown in Algorithm 1, which describes both the *counting* (lines 1-4) and *refinement* phases (lines 5-16). The counting phase constructs the item profiles $(IP_i)_{i \in I}$ and the ranked candidate sets $(RCS_u)_{u \in U}$ of the dataset. The item profiles reverse the user-item edges found in the bipartite graph G:

$$IP_i \equiv \{u \in U | i \in UP_u\}, \forall i \in I$$

(dashed arrows in Figure 2) and are computed while the dataset is being loaded (lines 1-2).

The ranked candidate sets are computed using the multiset union $\uplus$ (lines 3- 4). RCS_u only contains users whose ids are higher than that of u (constraint $v > u$ at line 4), as discussed in Section II-D. In terms of implementation, the ranked candidate sets RCS_u are maintained as explicit multisets only while they are being constructed (line 4). Once they are completed, they are sorted according to the multiplicity of their elements, and transformed into plain ordered lists (i.e., without multiplicity information), since only this order is used in the refinement phase (line 9). This stripping procedure lowers *KIFF*'s memory overhead, and improves the performance of the refinement phase.

The refinement phase (lines 5-16) exploits the ranked candidate sets RCS_u to iteratively construct an approximation of the dataset's KNN. In each iteration (**repeat**-**until** loop, lines 6-12), *KIFF* loops through each user u (line 8), and removes the top γ users from RCS_u (operation *top-pop*, line 9). The returned users are stored in the variable cs. They correspond to the γ users with higher ids than u, who have the most shared items with u among those not yet considered in earlier iterations.

In the inner **for** loop (lines 10-12), each of these top users v is considered as a potential neighbor for u, and reciprocally, u is considered as a potential neighbor for v. This symmetry is required because we have truncated ranked candidate sets to users with higher ids at line 4.

Neighborhoods are updated in the function UPDATENN (lines 14-16). The current approximation $\widehat{knn_u}$ of each user u's neighborhood is stored as a heap of maximum size k, with the similarity between u and its neighbors used as priority.

Algorithm 1: *KIFF*

Input: dataset U, user profiles $(UP_u)_{u\in U}$, similarity $f_{sim}()$, parameter k, termination β, parameter γ
Output: $(\widehat{knn}_u)_{u\in U}$ an approximated KNN

▷ *Counting Phase: building the Ranked Candidate Sets*

1 **for** $u \in U \wedge i \in UP_u$ **do** ▷ *Executed at loading time*
2 $\quad IP_i \leftarrow IP_i \cup \{u\}$ ▷ *IP_i initialized to $\varnothing$*
3 **for** $u \in U$ **do**
4 $\quad RCS_u \leftarrow \biguplus_{i\in UP_u} \{v \in IP_i | v > u\}$ ▷ *Multiset union*

▷ *Refinement Phase: greedy convergence*

5 $\widehat{knn}_u \leftarrow \varnothing, \forall u \in U$ ▷ *Initialize the KNN heaps (size k)*
6 **repeat**
7 $\quad c \leftarrow 0$ ▷ *Counts changes during one iteration*
8 $\quad$ **for** $u \in U$ **do**
9 $\quad\quad cs \leftarrow$ *top-pop*(RCS_u, γ) ▷ *Top γ users from RCS_u*
10 $\quad\quad$ **for** $v \in cs$ **do** ▷ *By construction $v > u$ (l. 4)*
11 $\quad\quad\quad s \leftarrow f_{sim}(UP_u, UP_v)$
12 $\quad\quad\quad c \leftarrow c+$UPDATENN$(u, v, s)+$UPDATENN$(v, u, s)$
13 **until** $\frac{c}{|U|} < \beta$
14 **Function** UPDATENN(user u, user v, similarity s) **is**
15 $\quad$ update $\widehat{knn}_u$ heap with (v, s)
16 $\quad$ **return** 1 if changed, or 0 if not

The number of changes made over an iteration are tracked in the variable c (lines 7 and 12). The algorithm stops when the average number of changes per user during an iteration has dropped below a predefined threshold β (line 12).

D. Convergence and discussion

In the refinement phase, the size of the ranked candidate sets (RCS_u) decreases strictly until the sets possibly become empty. This behavior guarantees that the number of similarity computations is limited by the sizes of the the ranked candidate sets, and cannot exceed $\sum_{u\in U} |RCS_u|$.

The parameter γ and β control how aggressively *KIFF* approximates an optimal KNN graph. One extreme case is when γ is chosen as ∞. In this case, all ranked candidate sets are exhausted during the first iteration of the **repeat-until** loop (lines 6-12). When this happens, the resulting KNN approximation $(\hat{knn}_u)_{u\in U}$ is optimal, assuming the similarity function f_{sim} fulfills the following two properties:

$$\forall A, B \in (I \nrightarrow R) : A \cap B = \varnothing \Rightarrow f_{sim}(A, B) = 0 \quad (5)$$

$$\forall A, B \in (I \nrightarrow R) : A \cap B \neq \varnothing \Rightarrow f_{sim}(A, B) \geq 0 \quad (6)$$

These two properties, which are fulfilled by most item-based similarity metrics, such as cosine similarity applied to positive ratings or Jaccard's coefficient, ensure that only users with at least one shared item can improve each other's KNN neighborhood. As a result *KIFF* does not miss out any potential good candidate by limiting and pruning the search to the ranked candidate sets $(RCS_u)_{u\in U}$.

IV. Experimental Setup

We evaluate *KIFF* by comparing it to two other state-of-the art approaches on a number of representative datasets. We have implemented *KIFF* and its competitors in Java[5] and execute them on a single machine. All implementations are multi-threaded to parallelize the treatment of individual users. Both the source code of all implementations and the datasets, are publicly available [19] on-line. Our experiments were conducted on a commodity server running a 64-bit Linux with an Intel Xeon E3-1246 v3, a 3.5GHz quad core processor totalling 8 hardware threads, with 32 GB of memory and a 256Gb PCI Express SSD.

A. Datasets

We use four publicly available user-item datasets [16] for our evaluation. These datasets are representative of a wide range of domains and applications, including bibliographic collections (Arxiv and DBLP), voting systems (Wikipedia) and on-line social networks with geo-location information (Gowalla). Some key properties of these four datasets are presented in Table I, including the number of users ($|U|$), number of items ($|I|$), and number of ratings ($|E|$, i.e., the number of weighted edges in the user-item bipartite graph). The density is the percentage of edges present in the dataset over a complete bipartite graph, i.e., a graph in which each user would have rated every item: $density = |E| \div (|U| \times |I|)$.

Figure 4a and 4b show the Complementary Cumulative Distribution Functions (CCDF) of the sizes of user and item profiles respectively, in these datasets. The long-tailed curves are consistent with earlier observation [20], [21], [22] and show that most users have very few ratings. In the following we briefly present each dataset.

1) Arxiv: The Arxiv dataset captures scientific collaborations among authors. In this dataset, authors play both the roles, i.e., of users and items: if two authors u_1 and u_2 have co-authored a paper, u_1 contains u_2 in her profile (UP_{u_1}) and vice-versa. This dataset does not include ratings (e.g., weights between authors corresponding to their co-publications). The dataset spans papers from January 1993 to April 2003 and represents the complete history of its GR-QC and ASTRO-PH sections [23].

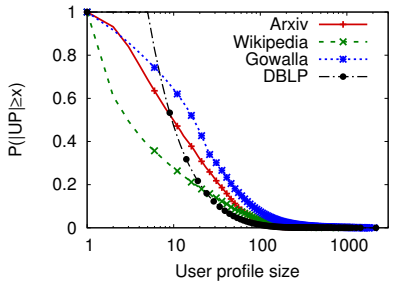
2) Wikipedia: Wikipedia is the well-known collaborative and free encyclopedia edited by volunteers. The promotion of a user to the role of administrator involves a consultation of other Wikipedia editors, who indicate whether they support a candidate or not. (This consultation is not, according to Wikipedia, exactly a vote.) Our dataset contains the complete Wikipedia consultation data from the origin of Wikipedia till January 2008 [24]. A positive vote of a Wikipedia editor (a user) for a candidate (an item) is encoded as a rating of 1, thereby resulting in binary ratings.

3) Gowalla: Gowalla was a location-based social networking website, acquired by Facebook in 2011, in which users could share their current location. This dataset consists of the check-ins of a subset of Gowalla users from February 2009 to October 2010 [25]. The profile of each user contains the list

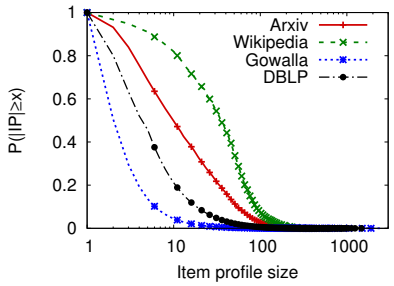
[5]The public sources of NN-Descent [13] are only available in C++. We have therefore re-implemented the published algorithm in Java.

TABLE I. DATASET DESCRIPTION (ALL DATASETS ARE AVAILABLE ON-LINE [19])

Dataset	#Users $\|U\|$	#Items $\|I\|$	#Ratings $\|E\|$	Density	Avg. $\|UP_u\|$	Avg. $\|IP_i\|$
Wikipedia	6,110	2,381	103,689	0.7127%	16.9	43.5
Arxiv	18,772	18,772	396,160	0.1124%	21.1	21.1
Gowalla	107,092	1,280,969	3,981,334	0.0029%	37.1	3.1
DBLP	715,610	1,401,494	11,755,605	0.0011%	16.4	8.3



(a) Size of the users profile.



(b) Size of the items profile.

Fig. 4. CCDF of the size of the user and item profiles, top and bottom respectively. Long-tailed curves show that most of the users have few ratings.

of locations (items) associated to her check-ins with a rating reflecting the number of time the location was visited.

4) DBLP: The DBLP computer science bibliography provides a comprehensive list of research papers in computer science. This dataset consists of the co-authorship network where two authors are connected if they have published a paper together. Our dataset is a snapshot of DBLP collected in September 2015 and contains information about users with at least five co-publications[6]. The profiles of the authors contain the list of co-authors, where the ratings reflect the number of publications they co-authored.

B. Competitors

KIFF relies on a preprocessing phase (the counting phase) that exploits the user-item bipartite graph to provide a first approximation of the KNN graph. This unique design places *KIFF* in a category of its own, which makes it difficult to choose obvious competitors to compare *KIFF* against. Since the refinement phase of *KIFF* relies on techniques similar to those used in greedy-based KNN construction algorithms, we picked two such recent state-of-the art approaches as comparison baselines: NN-Descent [13] (shown to be more efficient than LSH [10] and RLB [17] in [13]), and Hyrec [26].

NN-Descent [13] favors *node locality* to iteratively converge to a close approximation of an exact KNN. Starting from a random graph, NN-Descent iteratively refines the neighborhood of a user by considering at each iteration a candidate set composed of the direct neighborhood of the current bidirectional neighbors (both in-coming and out-going neighbors). To avoid repeated similarity computations, NN-Descent uses a system of flags to only consider new neighbors-of-neighbors during each iteration. Since such a candidate set may still be very large, only a sample might be considered to speed-up the process. NN-Descent also uses a pivot strategy similar to ours to ensure that the similarity computation between a pair of users is performed only once, by iterating on both the in-coming and out-going neighbors of the current pivot user.

HyRec [26] is a recommendation framework which relies on an approach similar to NN-Descent but distributes the KNN graph construction onto the browsers of users to improve the scalability of the KNN computation. Similar to NN-Descent, HyRec relies on node locality to iteratively converge to an accurate KNN from a random graph. During each iteration, HyRec considers the neighbors of neighbors of each user, as well as a set of few random users to avoid a local minimum as in [15]. In this approach, a parameter r is used to define the number of random users considered in the candidate set. For a fair comparison, although not mentioned in [26], we implement the same pivot mechanism as in NN-Descent and the early termination of *KIFF*.

C. Metrics

We assess the quality of the produced KNN graphs by measuring the recall of the graphs returned by each approach, as defined in Section III-B. For each dataset, an ideal KNN is constructed using a brute force approach. The recall is then obtained by comparing the similarity values of the ideal neighborhoods and those of the approximated ones.

We measure the computational cost of each solution using two metrics: the *scan rate*, and the *computation time*. The scan rate measures the number of similarity evaluations required to produce the final approximated KNN graph, normalized and expressed as a percentage of all possible KNN edges.

$$scanrate = \frac{\#(similarity\ \ evaluations)}{|U| \times (|U|-1)/2}$$

The computational time is the *wall-clock* time required to construct the KNN graph, excluding the JVM's start-up time, but including the loading time of the graph from persistent storage (i.e., measured from the JVM's entry into the `main` method). We also separately measure the time spent by each

[6]Note that this snapshot is different from that used in NN-Descent's original evaluation, which we were not able to reconstruct from the information provided in [13], in spite of our best efforts.

solution for different activities of each approach, namely *preprocessing* (loading the dataset, building user profiles, and in the case of *KIFF* building item profiles and performing the counting phase), *candidate selection* (constructing candidate neighborhoods, using the RCS sets in *KIFF* and using neighbor-of-neighbor links in Hyrec and NN-Descent) and *similarity computation*.

D. Default parameters

In our evalations, for the sake of simplicity we use the cosine similarity metric [18] but *KIFF* is designed to perform consistently regardless of the similarity metric considered. For a fair comparison, we set the parameters of each approach to their default values [13], [4]. More precisely, we set $k = 20$ (except for DBLP where we use $k = 50$), and use an early termination parameter β at 0.001 for *KIFF*. For NN-Descent, we report results without sampling (as in the original publication), and we consider $\gamma = 2k$ for *KIFF*. For HyRec, by default, we consider no random nodes in the candidate set ($r = 0$). Though not shown in the evaluation for space reasons, random nodes cause random memory accesses and drastically increase the wall-time (three times longer on average, with $r = 5$) while only slightly improving the recall (4% on average).

V. Evaluation

We now present the results obtained by *KIFF*, HyRec and NN-Descent under the experimental setup of the previous section. Our results show that *KIFF* efficiently converges to an approximate KNN graph, while outperforming its competitors both in terms of recall and computation time. We also assess the sensitivity of the three approaches to *k* and to the *density* of the datasets. We also evaluate the impact of γ on *KIFF*.

A. Performance Comparison

Table II shows the recall, wall-time, scan rate and number of iterations of NN-Descent, HyRec and *KIFF* on the four datasets considered for evaluation. The best recall and wall time values are highlighted in bold. The lines marked "*KIFF*'s Gain" show, for each dataset, the improvement in recall and the speed up achieved by *KIFF* over its competitors (averaged over NN-Descent and HyRec). These gains, averaged over all datasets, are summarized in Table III.

These results show that *KIFF* consistently outperforms NN-Descent and HyRec in terms of recall and wall-time across all datasets. While *KIFF* consistently achieves a recall of 0.99, the recall of NN-Descent and HyRec ranges from 0.56 (HyRec on Gowalla) to 0.97 (NN-Descent on Wikipedia), with an average of 0.85 for NN-Descent and 0.76 for HyRec. The higher recall values obtained by *KIFF* validate the use of ranked candidate sets which provide *KIFF* with a clear advantage in terms of KNN quality over the initial random graph used by HyRec and NN-Descent.

The results show that the scan rate (number of similarity computations required to converge) for *KIFF* is 7 and 6 times lower than that of NN-Descent and HyRec respectively. As *KIFF* requires lesser similarity computations, it is also faster than its competitors. The wall-time values of *KIFF* confirm our intuition that the counting phase of *KIFF*, by preprocessing the bipartite graph of the dataset, results in relatively faster convergence to an approximate knn graph. The speed-up achieved can be substantial: for instance, *KIFF* is 17 times faster than Hyrec and NN-Descent on the DBLP dataset, and 14 faster on average (Table III).

TABLE II. Overall perf. of NN-Descent, HyRec, & *KIFF*

datasets	Approach	recall	wall-time (s)	scan rate	#iter.
Arxiv	**NN-Descent**	0.95	41.8	17.6%	9
	HyRec	0.90	38.6	16.0%	12
	KIFF	**0.99**	**10.7**	2.5%	36
	KIFF's Gain	*+0.06*	*×3.7*		
Wikipedia	**NN-Descent**	0.97	13.1	51.69%	7
	HyRec	0.95	9.4	44.64%	8
	KIFF	**0.99**	**4.4**	7.37%	22
	KIFF's Gain	*+0.03*	*×2.5*		
Gowalla	**NN-Descent**	0.69	307.9	3.67%	16
	HyRec	0.56	253.2	2.69%	22
	KIFF	**0.99**	**146.6**	0.84%	115
	KIFF's Gain	*+0.36*	*×1.9*		
DBLP	**NN-Descent**	0.78	10,890.2	3.08%	19
	HyRec	0.63	8,829.9	2.37%	26
	KIFF	**0.99**	**568.0**	0.07%	33
	KIFF's Gain	*+0.28*	*×17.3*		

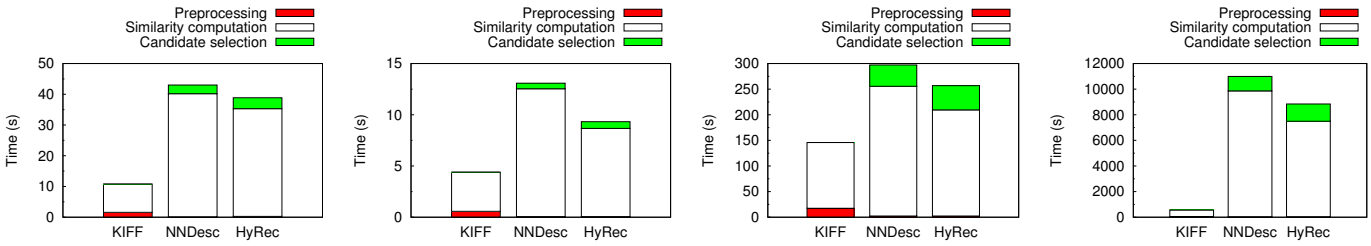
TABLE III. Average speed-up and recall gain of *KIFF*

	KIFF's gain	
Competitor	**speed-up**	**Δrecall**
NN-Descent	×15.42	+0.14
HyRec	×12.51	+0.23
Average	*×13.97*	*+0.19*

The reason why *KIFF* outperforms its competitors in terms of wall-time is visible in Figure 5, which charts the breakdown of the computation time of *KIFF*, NN-Descent, and HyRec into the three types of activities introduced in Section IV-C: (1) preprocessing (which includes *KIFF*'s counting phase); (2) candidate selection; and (3) similarity computations. The figure shows that the counting phase of KNN indeed introduces some overhead (ranging from 10.01% to 14.74% of *KIFF*'s overall computation time), but that this overhead is largely compensated by a much faster convergence, with fewer similarity computations, and lesser time spent on candidate selection.

In the following, we investigate in more detail how *KIFF* exploits ranked candidate sets to converge faster: we first analyze the overhead of *KIFF*'s preprocessing and the cost of computing RCSs; we then discuss the properties of the resulting RCSs; and we finally compare the convergence of *KIFF*'s refinement phase with the convergence of NN-Descent and HyRec.

1) Overhead of KIFF's preprocessing: *KIFF*'s preprocessing essentially comprises its *counting phase*, which computes *item profiles*, and *Ranked Candidate Sets* (RCSs). In *KIFF*, item profiles are built when the user profiles (required by all approaches) are created from the input data stream. Table IV compares the time taken to construct user profiles only ((UP) column) to that taken to construct both user and item profiles simultaneously, and contrasts the difference between the two (Δ column) with the overall running time of *KIFF* (last column), for all data sets. The table shows that the overhead caused by item profiles only represents a very small fraction



(a) Arxiv (b) Wikipedia (c) Gowalla (d) DBLP

Fig. 5. Although *KIFF* must pay higher preprocessing costs to constructs its Ranked Candidate Sets, this overhead is largely balanced out by a smaller number of similarity computations due to a much faster convergence compared to Hyrec and NN-Descent.

TABLE IV. OVERHEAD OF ITEM PROFILE CONSTRUCTION IN *KIFF*

Dataset	Wall-time (ms)		Δ(ms)	%age total time
	(UP_u)	$(UP_u)\&(IP_i)$		
Arxiv	135	185	50	0.5%
Wikipedia	59	69	10	0.2%
Gowalla	2,354	5,136	2,782	1.9%
DBLP	7,492	12,996	5,504	1.0%

TABLE V. OVERHEAD OF RCS CONSTRUCTION & STATISTICS (*KIFF*)

Dataset	RCS const. (ms)	%age total time	avg. |RCS|	max RCS scan rate
Arxiv	1,404	13.1%	247.0	2.63%
Wikipedia	465	10.6%	228.7	7.48%
Gowalla	12,255	8.4%	458.1	0.85%
DBLP	42,829	7.5%	267.8	0.07%

of the total running time of *KIFF* across all datasets (at most 1.9% on Gowalla).

The time taken to compute ranked candidate sets ($RCSs$) is shown in Table V, along with some additional statistics (which we return to just below). Constructing $RCSs$ takes substantially more time than constructing item profiles, and makes up, in effect, most of *KIFF*'s preprocessing overhead, amounting to up to 13.5% of *KIFF*'s overall running time (with Arxiv).

2) Properties of the resulting RCSs: The extent to which *KIFF* is able to recoup this overhead strongly depends on the capacity of the resulting RCSs to limit similarity computations by pruning pairs of users that have no items in common. This *pruning effect* is directly linked to the size of RCSs: short RCSs (in comparison to the total number of users in the dataset) eliminate many similarity computations and help accelerate *KIFF*'s refinement phase. To shed light on this aspect, Table V shows the average lengths of the RCSs computed for each dataset and the corresponding maximum scan rate induced by these RCSs

$$max\_scan = \frac{|U| \times |RCS|}{\frac{1}{2} \times |U| \times (|U|-1)} = 2 \times \frac{|RCS|}{|U|-1}$$

i.e., the scan rate one would obtain by iterating through the entire RCSs in the refinement phase ($\gamma = \infty$ in Algorithm 1). We note that this maximum scan rate is in fact very close to the actual scan rate reported in Table II: the value of the termination parameter β chosen for *KIFF* (0.001) causes most RCSs to be fully iterated through, which explains the high recall values obtained by *KIFF*.

TABLE VI. IMPACT OF *KIFF*'S TERMINATION MECHANISM

Dataset	#iters	$\|RCS\|_{\text{cut}}$	%user $\|RCS_u\| > \|RCS\|_{\text{cut}}$
Arxiv	36	720	9.57%
Wikipedia	22	440	16.24%
Gowalla	115	2300	4.82%
DBLP	33	660	10.32%

This phenomenon is confirmed by Table VI, and Figures 6 and 7. Table VI repeats the number of iterations executed by *KIFF* from Table II. This number corresponds to a maximum number of entries considered by *KIFF* from each RCS, which we note $|RCS|_{\text{cut}}$ (second column, this is simply #iters $\times$ γ): RCSs which are smaller that this size are fully iterated through, while larger RCSs are truncated. The percentage of users with truncated RCSs for each dataset is shown in the last column, and illustrated graphically using vertical bars in Figure 6 (which shows the CCDF of RCS sizes).

Only a minority of users experience truncated RCSs, but this minority can be important, as in the case of Wikipedia, where 16.24% of users are not compared with all their candidate KNN neighbors. This truncation could potentially hurt *KIFF*'s recall if too many ideal KNN neighbors are found towards the tail of the truncated RCSs. Figure 7 shows that this is not the case. (We only show the Wikipedia dataset due to space reasons.) Each point in the figure is a RCS with a size larger that Wikipedia's truncation value (440), which is plotted according to its size and Spearman's correlation between the RCS (ordered according to common item counts) and the same list of nodes sorted following either Jaccard's coefficient or the cosine similarity metrics. The figure shows that Spearman's correlation coefficient is generally high (on average 0.60 for Jaccard, and 0.63 for cosine) for truncated RCSs, and increases with RCSs sizes, suggesting that the approximation provided by the counting phase does not generally exclude good candidates from being considered as potential neighbors.

The quality of the RCSs is finally further illustrated by Table VII, which shows the recall obtained by *KIFF* without any convergence ($\beta = \infty$ in Algorithm 1), i.e., when using the top k users of each RCS as a KNN graph approximation. Although the resulting recall varies widely from 0.54 (Wikipedia) to 0.82 (Arxiv), these values are much larger than those obtained through a random initial graph (which peaks at 0.15 for Gowalla), as used by traditional greedy approaches, and illustrates the immediate benefit obtained by *KIFF* from its counting phase.

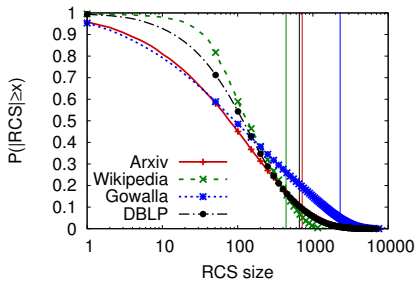


Fig. 6. CCDF of $|RCS|$. The vertical bars shows the cut-off sizes enforced by *KIFF*'s termination mechanism (also shown in Table VI)

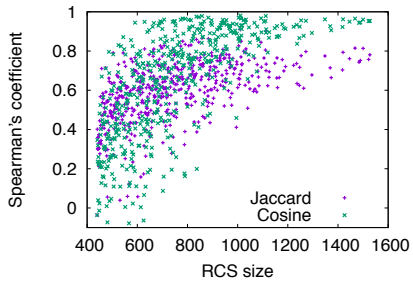
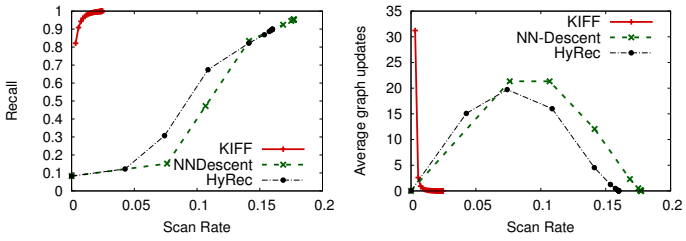


Fig. 7. Correlation between the rank of nodes in $RCSs$ and the cosine and Jaccard metrics for users with $|RCS_u|$ above $|RCS|_{\text{cut}} = 440$ for Wikipedia.

3) Convergence: Although the incremental convergence of *KIFF* derives its motivation from greedy-based approaches, its convergence pattern is clearly different from that of its greedy-based competitors, NN-Descent and HyRec, as shown in Figure 8. Figure 8a charts the quality of the KNN graph under construction as a function of scan rate for the three approaches on the Arxiv dataset. As indicated in Tables II and VII, *KIFF* immediately starts with a high recall value (0.82), and terminates after a very small scanrate (2.5%). By contrast, HyRec and NN-Descent start from a low recall value (0.08) induced by their random initial graph, and only converge

TABLE VII. IMPACT OF INITIALIZATION METHOD ON INITIAL RECALL

	Recall	
Dataset	Top k from RCS	Random
Arxiv	0.82	0.08
Wikipedia	0.54	0.01
Gowalla	0.55	0.15
DBLP	0.79	0.09



(a) scan rate vs recall (b) scan rate vs updates

Fig. 8. *KIFF* quickly provides an accurate KNN approximation by using relevant candidates compared to competitor approaches (Arxiv dataset).

at a much higher scan rate (resp. 16.0% and 17.6%). (*KIFF* shows similar behavior for the remaining datasets, which are not included here due to space reasons.) The rapid convergence of *KIFF* can be attributed to the effective pruning induced by $RCSs$ that leads to a reduction in similarity computations. The slow convergence of NN-Descent and HyRec by contrast can be credited to their random initial start and their transitive convergence mechanisms (exploiting neighbors-of-neighbors links), which requires more computational resources to converge to a good KNN graph approximation.

The effect of the convergence procedure (RCSs vs. neighbors-of-neighbors links) on the convergence speed is visible in Figure 8b, which shows the average number of updates performed on the KNN graph during each iteration. We observe that the use of RCSs to select potential candidates allows *KIFF* to perform many relevant updates to the KNN graph during its first iteration. The number of updates performed in subsequent iterations decreases rapidly in the case of *KIFF*, reflecting the fact that RCSs are ordered according to decreasing common item counts. This property in turn leads to a much faster convergence. In contrast, NN-Descent and HyRec present a three-step convergence pattern. First, the random nature of the initial graph significantly increases the number of beneficial updates during the first few iterations. Recall only improves slightly, however, during this first step (Figure 8a). During the second step, more and more potential KNN candidates are encountered, leading to an improved approximation with a better recall. Finally, during the third phase, the last iterations before convergence, fewer updates are performed as good potential candidates become harder to discover in the graph, a fact reflected in a slowly increasing recall till convergence.

B. Sensitivity analysis: k, γ and density

1) Impact of parameter k: The number of similarity computations NN-Descent and HyRec execute is closely related to the size of the neighborhoods they maintain (parameter k). This is because both approaches leverage neighbors-of-neighbors relationships to improve their KNN approximation. Smaller neighborhoods constrain their search for potential neighbors. As a result, smaller values of k should reduce the number of similarity values both approaches need to compute, and lead to better computation times. There is however a trade-off, as too small neighborhoods might disrupt the greedy convergence of NN-Descent and HyRec and yield degraded KNN results.

Contrary to NN-Descent and HyRec, *KIFF* does not use its intermediate neighborhoods (the variables $\hat{knn}_u$ of Algorithm 1) to drive its convergence, but relies instead on pre-computed ranked candidate sets (RCS_u). One might therefore expect its computation time and recall to be less impacted by lower k values.

To better understand how *KIFF* and its two competitors behave for smaller values of k, Table VIII presents their performance in terms of computation time, recall, and scan rate when we lower k from 20 to 10 (from 50 to 20 for DBLP) on the same datasets as Table II. These results confirm that a smaller k does lead to substantial gains in computation times for NN-Descent and HyRec (speed-up ratios ranging from 2.35 to 4.07), with a less pronounced improvement for

TABLE VIII. IMPACT OF k ON RECALL AND WALL-TIME (k=10, DBLP: k=20)

datasets	Approach	recall	wall-time (s)	scan rate
Arxiv	**NN-Descent**	0.74 (-0.20)	17.7 (÷2.36)	5.49% (-12.1%)
	HyRec	0.55 (-0.34)	16.4 (÷2.35)	4.66% (-11.3%)
	KIFF	**0.99** (=)	**7.8** (÷1.36)	1.97% (-0.4%)
	KIFF's Gain	*+0.34* (+0.28)	*×2.18* (÷1.70)	
Wikipedia	**NN-Descent**	0.86 (-0.10)	5.3 (÷2.46)	16.39% (-35.3%)
	HyRec	0.74 (-0.20)	3.6 (÷2.55)	13.98% (-30.6%)
	KIFF	**0.99** (=)	**3.2** (÷1.36)	6.86% (-0.5%)
	KIFF's Gain	*+0.19* (+0.16)	*×1.39* (÷1.80)	
Gowalla	**NN-Descent**	0.35 (-0.33)	117.8 (÷2.61)	0.89% (-2.7%)
	HyRec	0.26 (-0.29)	**98.7** (÷2.70)	0.61% (-2.0%)
	KIFF	**0.99** (=)	120.4 (÷1.21)	0.73% (-0.1%)
	KIFF's Gain	*+0.68* (+0.32)	*×0.89* (÷2.13)	
DBLP	**NN-Descent**	0.20 (-0.57)	2,673.4 (÷4.07)	0.43% (-2.6%)
	HyRec	0.11 (-0.52)	2,272.5 (÷3.88)	0.26% (-2.1%)
	KIFF	**0.99** (=)	**516.6** (÷1.09)	0.07% (=)
	KIFF's Gain	*+0.83* (+0.55)	*×4.78* (÷3.62)	

Evolution of wall-time, recall, and scan rate when reducing k from 20 to 10 (DBLP: 50 to 20). The numbers in brackets show the change w.r.t to the results with $k = 10$ (DBLP:$k = 20$) presented in Tab. II. *KIFF* shows little sensitivity to the value of k, with slightly lower computation times, and identical recall values. By contrast NN-Descent and HyRec are strongly impacted, with much reduced wall-times accompanied by degraded recall values. *KIFF* remains the best performing solution across the board.

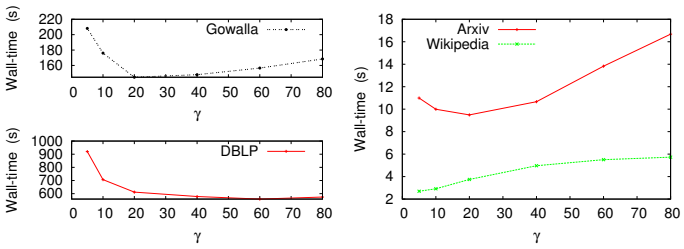


Fig. 9. The impact of γ on the wall-time of *KIFF* remains low.

KIFF (speed-up varying from 1.09 to 1.36). Simultaneously, however, the recall values of NN-Descent and HyRec degrade substantially (with losses varying from 11% to 35%), while those of *KIFF* remain stable. Across the board, *KIFF* continues to outperform its competitors in terms of computation times and recall on 3 datasets out of 4 (Arxiv, Wikipedia, and DBLP). On Gowalla, NN-Descent and HyRec have become faster, but with much degraded recalls (0.35 and 0.26), that can no longer compete with that of *KIFF* (0.99).

It is worth noting that the time spent in similarity computations depicted in Figure 5 depends linearly on the complexity of the similarity metric used to compute the KNN graph. Therefore, using a metric requiring twice as long to evaluate the similarity between two profiles will double the global time of similarity computation, and will further increase the benefits of our solution compared to NN-Descent and HyRec.

2) Impact of parameter γ: The parameter γ in *KIFF* defines the number of elements removed from the ranked candidate sets of each user and considered as potential neighbors at each iteration. If the ranked candidate set of user u, RCS_u, contains less than γ elements, only these elements are considered and RCS_u becomes empty. Consequently, γ impacts the number of similarity computations performed at each iteration which is bound by $|u|\gamma$. The number of considered neighbors which become actual neighbors over an iteration are tracked (variable c in Algorithm 1) and *KIFF* stops if the average number of changes is bellow a predefined termination threshold β.

On the one hand, the parameter γ impacts the number of iterations that *KIFF* needs to converge. A larger γ will reduce the number of iterations, in contrast a smaller γ will force *KIFF* to perform more iterations to converge. While the number of iterations does not impact the scan rate which depends on the global number of similarity computations actually performed, a too small γ will tend to increase the wall-time due to iteration overheads as shown in Figure 9 which depicts the wall-time of *KIFF* according to the value of γ. On the other hand, at the last iteration where $\frac{c}{|u|} < \beta$, the probability to perform more similarity computations than actually required to reach the termination threshold is a function of γ. As a consequence, a too large γ tends to increase the scan rate. However, as depicted in Figure 9, the impact of γ on the wall-time remains low.

Although we cannot fully investigate its impact for space reasons, let us note for completeness that the termination threshold β also has an important influence on the performance of *KIFF*. The value of β represents a trade-off between recall and scan rate (and hence wall-time). For instance, on the Arxiv dataset, increasing β hundredfold to 0.1 (from 0.001) causes *KIFF* to take 36% less time to converge by halving its scan rate to convergence. Recall is mildly impacted, being reduced by 0.01, down to 0.98.

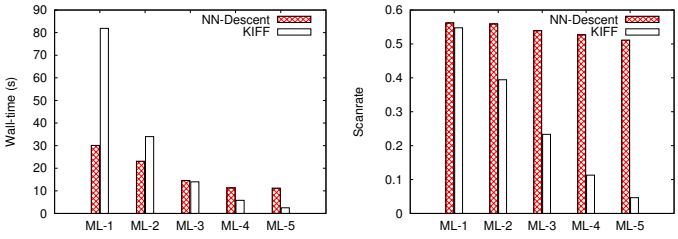
3) Impact of density: *KIFF* is designed to work well on datasets with small user and item profiles, relatively to the total number of items and users. This situation typically corresponds to a low density of the associated bipartite graph. The datasets we have considered (Table I) have this property, with density values remaining below 1% (our highest density is 0.71% for the Wikipedi dataset). For completeness' sake, we investigate in this last evaluation, how *KIFF* behaves on denser datasets, and contrast its performance to that of NN-Descent. For this comparison, we derive from the same underlying dataset, a family of bipartite graphs with different density characteristics.

We start from a dataset from the MovieLens (ML) repository [27]. MovieLens provides movie-rating data collected on the ML recommender website over a 7-month period. ML ratings range over a 5-star scale, with half-star increments. The ML dataset we use (called ML-1 in the following) contains 6,040 users and 3,706 items (movies), in which each user has at least made 20 ratings, with an average of 165.1 ratings per user. ML-1 is substantially denser than the datasets we have considered so far, with a density of 4.47%. Starting from ML-1, we progressively remove randomly chosen ratings and obtain four additional datasets (numbered ML-2 to ML-5) showing decreasing density values (ranging from 2.23% to 0.30%, Table IX).

For a fair comparison between *KIFF* and NN-Descent we first measure the recall obtained by NN-Descent on ML1-5 with the default parameters mentioned in Section IV-D. NN-Descent yields a consistent recall value of 0.93 for ML1, ML2,

TABLE IX. MOVIELENS DATASETS WITH DIFFERENT DENSITY.

Dataset	Ratings	Density	average $\lvert RCS \rvert$
ML-1	1,000,209	4.47%	2,892.7
ML-2	500,009	2.23%	2,060.6
ML-3	255,188	1.14%	1,125.4
ML-4	131,668	0.59%	510.8
ML-5	68,415	0.30%	202.5



(a) wall-time (b) scan rate

Fig. 10. a) *KIFF* is faster than NN-Descent for sparse datasets ; b) the scan rate for *KIFF* decreases according to the dataset density compared to NN-Descent which is not correlated to density.

ML3, and ML4, and a recall of 0.97 for ML-5. We then set the β parameter of *KIFF* for each dataset so as to obtain the same recalls as NN-Descent.

The overall execution time (wall-time) and scan rate obtained by the two competitors when applying this procedure are reported in Figures 10a and 10b. These results show that *KIFF* is slower than NN-Descent for dense datasets (ML-1 and ML-2), but that the situation reverses itself for sparser versions (ML-4 and ML-5), with the two approaches being close on ML-3 (density of 1.14%). The evolution of *KIFF*'s computation time can be directly related to its scan rate (Fig. 10b) and the average size of the ranked candidate sets *KIFF* constructs (Table IX). While the scan rate of NN-Descent remains roughly stable in the $5\%-6\%$ range across all datasets, that of *KIFF* decreases sharply with dataset density, showing the importance of low density values (the common case) for *KIFF* to perform well.

VI. RELATED WORKS

The construction of a KNN graph is related to Nearest Neighbor (NN) search [28], [29], [30]. NN search addresses, however, a different problem: it seeks to find the k nearest neighbors of a small number of individual elements (the *queries*), rather than constructing a complete KNN graph as *KIFF* does. NN Search typically relies on complex precomputed indexes, and aims to minimize both the size of these indexes, and the computation time of individual queries. NN Search has been extensively researched, in particular to address what is known as the *curse of dimensionality*, i.e., the observation that traditional space-partitioning indexes become too large in large dimensions, a situation commonly encountered in multimedia databases [31], [32]. Solutions to this problem rely on advanced hashing techniques such as *Locality-Sensitive Hashing* (LSH) [10], [32] or on compact metric approximations such as *product quantization* [31]. These approaches are, however, optimized for very dense data sets: all images of a media database are usually associated with non-zero values along all dimensions. By contrast, *KIFF* targets sparse datasets characterized by many zero values in a very large dimensional space (the items).

Also related to but different from the problem addressed in this paper, *clustering* [33] seeks to partition a dataset in k clusters while minimizing the graph cut between successive clusters. The bipartite graph underlying node-item datasets (e.g., containing users and items, or documents and words) has been exploited in this context to cluster such datasets simultaneously along their node and item dimensions, a technique known as *co-clustering* [34]. These approaches share a common intuition with *KIFF* by using the dataset's bipartite graph, but they apply to a completely different problem.

The idea of combining a coarse (in our case the number of overlapping items) and a finer metric (in our experiments cosine similarity) as *KIFF* does, has been proposed in the context of hybrid recommender systems [35] as a generic pattern to combine two similarity metrics. The approach is, however, different from ours in that the finer metrics are only used to resolve ties created by the coarse metric. In particular, it is not supposed to overturn a coarse ranking, which has higher priority. Further, and as far as we know, the approach has not been applied to KNN graph construction.

Because the construction of a KNN graph generally implies the computation of a large number of similarity measures, several works have proposed to use space decomposition techniques (similar a divide and conquer strategy) to reduce the complexity of the graph construction [17], [36], [7], [37]. This techniques are unfortunately limited to particular metric spaces, typically based on an Euclidean distance, and do not generalize to other similarity measures.

Originally designed for overlay construction in fully decentralized systems, a number of greedy approaches have been proposed to compute KNN graphs [11], [14], [26]. These methods leverage the transitivity of most similarity functions (if A is close to B, and B to C, A is usually close to C as well) to iteratively refine and converge to a good KNN approximation, or even produce the true KNN. NN-Descent [13] extends this strategy with several optimizations designed to fully exploit multi-core machines. NN-Descent has shown to deliver a better recall in a shorter computational time than a space partitioning technique using Recursive Lanczos Bisection (RLB) [17], or than an approach using Locality Sensitive Hashing (LSH) [10]. NN-Descent in particular performs well on very dense datasets such as image databases. As our evaluation shows, however, it is outperformed by *KIFF* sparse datasets, and its performance strongly depends on the value of k. Finally, HyRec [4] also exploits a similar greedy-based approach to propose a generic recommendation framework based on user-based KNN graph.

VII. CONCLUSION

We have presented the design and evaluation of *KIFF*, a novel, generic, scalable, and efficient algorithm to construct KNN graphs. *KIFF* leverages the bipartite node-item graph found in many datasets to build ranked candidate sets. These ranked sets, in turn, drastically reduce the number of similarity computations performed, and hence the convergence time. We have provided an extensive evaluation of *KIFF* over a variety of datasets. Our evaluation demonstrates that the proposed

solution achieves a substantial speed-up in comparison to the state-of-the art approaches while improving the quality of the KNN graph. Moreover, we have also assessed the sensitivity of *KIFF* and show that its performance is independent of the value of k. We can also clearly conclude than *KIFF* outperforms its competitors even more on sparse datasets.

Furthermore, leveraging a heuristic to limit the insertion of elements in the ranked candidate sets is an interesting perspective. Preliminary works considering a naive threshold on multiple-ratings to insert, in the ranked candidate sets, only those users who have positively rated items, reduces the $RCSs'$ size and improves the performance of *KIFF*.

REFERENCES

[1] M. Bertier, D. Frey, R. Guerraoui, A. Kermarrec, and V. Leroy, "The gossple anonymous social network," in *Middleware*, 2010, pp. 191–211.

[2] X. Bai, M. Bertier, R. Guerraoui, A.-M. Kermarrec, and V. Leroy, "Gossiping personalized queries," in *EDBT*, 2010, pp. 87–98.

[3] J. J. Levandoski, M. Sarwat, A. Eldawy, and M. F. Mokbel, "Lars: A location-aware recommender system," in *ICDE*, 2012.

[4] A. Boutet, D. Frey, R. Guerraoui, A.-M. Kermarrec, and R. Patra, "HyRec: Leveraging Browsers for Scalable Recommenders," in *Middleware*, 2014, pp. 85–96.

[5] G. Linden, B. Smith, and J. York, "Amazon.com recommendations: item-to-item collaborative filtering," *Internet Computing*, vol. 7, no. 1, pp. 76–80, 2003.

[6] Y. Park, S. Park, W. Jung, and S. Lee, "Reversed cf: A fast collaborative filtering algorithm using a k-nearest neighbor graph," *Expert Systems with Applications*, vol. 42, no. 8, pp. 4022 – 4028, 2015.

[7] N. Nodarakis, S. Sioutas, D. Tsoumakos, G. Tzimas, and E. Pitoura, "Rapid aknn query processing for fast classification of multidimensional data in the cloud," *CoRR*, vol. abs/1402.7063, 2014.

[8] E. Bingham and M. Heikki, "Random projection in dimensionality reduction: Applications to image and text data," in *KDD*, 2001, pp. 245–250.

[9] A. Gionis, P. Indyk, and R. Motwani, "Similarity search in high dimensions via hashing," in *VLDB*, 1999, pp. 518–529.

[10] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li, "Multi-probe lsh: Efficient indexing for high-dimensional similarity search," in *VLDB*, 2007, pp. 950–961.

[11] S. Voulgaris and M. v. Steen, "Epidemic-style management of semantic overlays for content-based searching," in *Euro-Par*, 2005, pp. 1143–1152.

[12] Y. Malkov, A. Ponomarenko, A. Logvinov, and V. Krylov, "Approximate nearest neighbor algorithm based on navigable small world graphs," *Information Systems*, vol. 45, pp. 61–68, 2014.

[13] W. Dong, C. Moses, and K. Li, "Efficient k-nearest neighbor graph construction for generic similarity measures," in *WWW*, 2011, pp. 577–586.

[14] M. Jelasity, A. Montresor, and O. Babaoglu, "T-man: Gossip-based fast overlay topology construction," *Computer Networks*, vol. 53, no. 13, pp. 2321–2339, 2009.

[15] S. Voulgaris and M. van Steen, "Vicinity: A pinch of randomness brings out the structure," in *Middleware*, 2013, pp. 21–40.

[16] J. Leskovec and A. Krevl, "SNAP Datasets: Stanford large network dataset collection," http://snap.stanford.edu/data, Jun. 2014.

[17] J. Chen, H. Fang, and Y. Saad, "Fast approximate knn graph construction for high dimensional data via recursive lanczos bisection," *Journal of Machine Learning Research*, vol. 10, pp. 1989–2012, 2009.

[18] C. J. van Rijsbergen, *Information retrieval*. Butterworth, 1979.

[19] KIFF: Knn Impressively Fast and eFficient java library, https://gitlab.com/antoine.boutet/KIFF.

[20] M. Cha, H. Haddadi, F. Benevenuto, and K. P. Gummadi, "Measuring user influence in twitter: The million follower fallacy," in *ICWSM*, 2010.

[21] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee, "Measurement and analysis of online social networks," in *IMC*, 2007, pp. 29–42.

[22] M. Cha, A. Mislove, and K. P. Gummadi, "A measurement-driven analysis of information propagation in the flickr social network," in *WWW*, 2009, pp. 721–730.

[23] J. Leskovec, J. Kleinberg, and C. Faloutsos, "Graph evolution: Densification and shrinking diameters," *Transactions on Knowledge Discovery from Data*, vol. 1, no. 1, 2007.

[24] J. Leskovec, D. Huttenlocher, and J. Kleinberg, "Signed networks in social media," in *CHI*, 2010, pp. 1361–1370.

[25] E. Cho, S. A. Myers, and J. Leskovec, "Friendship and mobility: User movement in location-based social networks," in *KDD*, 2011.

[26] A. Boutet, D. Frey, R. Guerraoui, A. Jégou, and A.-M. Kermarrec, "WhatsUp Decentralized Instant News Recommender," in *IPDPS*, 2013, pp. 741–752.

[27] B. Amento, L. Terveen, and W. Hill, "Does "authority" mean quality? predicting expert quality ratings of web documents," in *SIGIR*, 2000, pp. 296–303.

[28] S. Yang, M. Cheema, X. Lin, and Y. Zhang, "Slice: Reviving regions-based pruning for reverse k nearest neighbors queries," in *ICDE*, 2014, pp. 760–771.

[29] C. Yu, B. C. Ooi, K.-L. Tan, and H. V. Jagadish, "Indexing the distance: An efficient method to knn processing," in *VLDB*, 2001, pp. 421–430.

[30] M. Muja and D. G. Lowe, "Scalable nearest neighbor algorithms for high dimensional data," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, 2014.

[31] H. Jegou, M. Douze, and C. Schmid, "Product quantization for nearest neighbor search," *Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 1, pp. 117–128, 2011.

[32] M. Norouzi, A. Punjani, and D. Fleet, "Fast search in hamming space with multi-index hashing," in *CVPR*, 2012, pp. 3108–3115.

[33] D. Bortner and J. Han, "Progressive clustering of networks using structure-connected order of traversal," in *ICDE*, 2010, pp. 653–656.

[34] I. S. Dhillon, "Co-clustering documents and words using bipartite spectral graph partitioning," in *KDD*, 2001, pp. 269–274.

[35] R. Burke, "Hybrid recommender systems: Survey and experiments," *User Modeling and User-Adapted Interaction*, vol. 12, no. 4, pp. 331–370, 2002.

[36] M. Connor and P. Kumar, "Fast construction of k-nearest neighbor graphs for point clouds," *Transactions on Visualization and Computer Graphics*, 2010.

[37] M. Otair, "Approximate k-nearest neighbour based spatial clustering using k-d tree," *IJDMS*, vol. 5, no. 1, 2013.