KIFF: un algorithme de construction de graphe KNN générique, rapide et évolutif

Antoine Boutet, Anne-Marie Kermarrec, Nupur Mittal, François Taïani

KIFF: un algorithme de construction de graphe KNN générique, rapide et évolutif

Antoine Boutet[1] et Anne-Marie Kermarrec[2] et Nupur Mittal[2] et Francois Taiani[3]

[1] *University of Lyon, LIRIS, CNRS, INSA-Lyon, UMR5205, F-69621, France*
[2] *INRIA, Rennes, France*
[3] *University of Rennes 1, Rennes, France*

Les algorithmes de construction du graphe des plus proches voisins (K-Nearest-Neighbor,KNN) sont apparus comme des éléments fondamentaux de nombreux services en ligne tels que la recommandation, la recherche de similarité et de classification. La construction d'un tel graphe de manière précise reste cependant une tâche très coûteuse. Avec l'augmentation des volumes de données, le temps nécessaire à la construction d'un graphe KNN et son évolution sont devenus des facteurs critiques. Dans cette contribution, nous proposons un algorithme de construction de graphe KNN générique, rapide et évolutif. Cet algorithme exploite la nature bipartite de la plupart des ensembles de données auxquels les algorithmes KNN sont appliqués. Plus précisément, un pré-traitement est effectué permettant d'identifier les candidats les plus pertinents auxquels chaque noeud peut se comparer. Cette stratégie permet ainsi de limiter de manière drastique le coût de calcul nécessaire à la convergence vers un graphe KNN précis, en particulier pour les ensembles de données de faible densité. Basé sur plusieurs jeux de données, nous montrons de manière expérimentale que notre solution calcule rapidement une approximation proche du KNN idéal tout en réduisant le coût de calcul par rapport aux approches de l'état de l'art (en moyenne 14 fois plus rapide tout en améliorant la qualité du KNN de 18%). Cet article reprend en grande partie des résultats publiés à la conférence ICDE en 2016 [BKMT16].

Mots-clefs : graphe des plus proches voisins, graphe biparti, jeux de données éparses

1 Introduction

K-Nearest-Neighbor (KNN) graphs play a fundamental role in many web-based applications e.g., search, recommendation and classification. 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. 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) [DML11]. 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. 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 [DML11], and HyRec [BFG$^+$14]. The figure shows the breakdown, in seconds, of the computing time of both algorithms on a small Wikipedia dataset. 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).

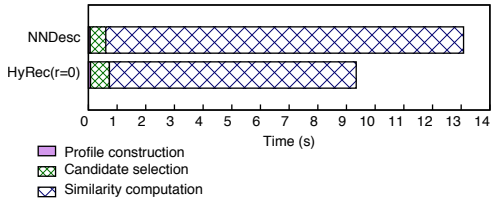


Fig. 1: State-of-the art greedy KNN-graph approaches spend over 90% of their computation time computing similarity values (Wikipedia dataset).

In this contribution, 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 [VvS13].

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 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 [DML11, BFG$^+$14], regardless of the value of k. We also assess the impact of graph density on *KIFF*'s performance, and show that *KIFF*'s performance is strongly correlated to a dataset's density.

2 KIFF: intuition and overview

The idea behind KIFF is to reduce substantially the time to build a KNN graph 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[†] 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 (i.e., in decreasing order). 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.

As shown in Algorithm 1, 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. In the refinement phase, the size of the ranked candidate sets decreases strictly until the sets possibly become empty. This behavior guarantees that the number of similarity computations is limited by the sizes of the ranked candidate sets. In addition, parameters γ and β control how aggressively *KIFF*

[†] For ease of exposition, we assume 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.

Tab. 1: Dataset description

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

approximates an optimal KNN graph. The former parameter bounds the computational cost paid at each iteration while the latter defines the termination threshold in term of total number of neighbors changes during an iteration.

3 Experiment

We extensively evaluate *KIFF* on four representative datasets. 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). Properties of these four datasets are presented in Table 1 including the density computed as $|E| \div (|U| \times |I|)$.

Table 2 shows the recall (i.e., the quality of the KNN computed as the ratio of exact KNN neighbors in the KNN approximation), the wall-time, the scan rate (i.e., the number of similarity evaluations required to produce the final approximated KNN graph expressed as a percentage of all possible KNN edge) and the number of iterations of NN-Descent, HyRec [‡] and *KIFF* on the four datasets using the cosine similarity metric and $k = 20$ (except for DBLP where we use $k = 50$). These gains, averaged over all datasets, are summarized in Table 3.

Tab. 2: 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*		

Tab. 3: Average speed-up and recall gain of *KIFF*

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

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 ∅*
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 $\quad\quad cs \leftarrow top\text{-}pop(RCS_u, \gamma)$ ▷ *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 $\quad\quad\quad c \leftarrow$ $c + \text{UPDATENN}(u, v, s) + \text{UPDATENN}(v, u, s)$
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

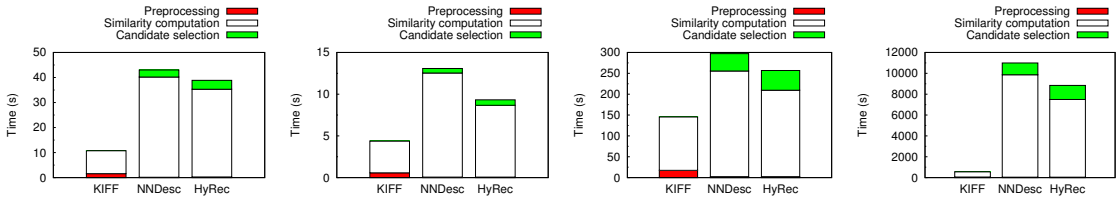
These results show that *KIFF* consistently outperforms NN-Descent and HyRec in terms of recall and wall-time across all datasets. The higher recall values obtained by *KIFF* validate the use of ranked candidate

[‡] For HyRec, by default, we consider no random nodes in the candidate set (r = 0).

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 3).

The reason why *KIFF* outperforms its competitors in terms of wall-time is visible in Figure 2, which charts the breakdown of the computation time of *KIFF*, NN-Descent, and HyRec into the three types of activities: (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.



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

Fig. 2: 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.

4 Conclusion

We have presented *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. 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, in the results published at ICDE [BKMT16], we have also assessed the sensitivity of *KIFF* and showed that its performance is independent of the value of k. We have also clearly showed that *KIFF* outperforms its competitors even more on sparse datasets.

References

[BFG+14] A. Boutet, D. Frey, R. Guerraoui, A.-M. Kermarrec, and R. Patra. HyRec: Leveraging Browsers for Scalable Recommenders. In *Middleware*, pages 85–96, 2014.

[BKMT16] A. Boutet, A. M. Kermarrec, N. Mittal, and F. Taiani. Being prepared in a sparse world: The case of knn graph construction. In *ICDE*, pages 241–252, 2016.

[DML11] W. Dong, C. Moses, and K. Li. Efficient k-nearest neighbor graph construction for generic similarity measures. In *WWW*, pages 577–586, 2011.

[VvS13] S. Voulgaris and M. van Steen. Vicinity: A pinch of randomness brings out the structure. In *Middleware*, pages 21–40, 2013.