Connectivity Structure of Bipartite Graphs via the KNC-Plot

Ravi Kumar Andrew Tomkins Erik Vee
Yahoo! Research
701 First Ave
Sunnyvale, CA 94089.
{ravikumar,atomkins,erikvee}@yahoo-inc.com

ABSTRACT

In this paper we introduce the *k-neighbor connectivity plot*, or KNC-plot, as a tool to study the macroscopic connectivity structure of sparse bipartite graphs. Given a bipartite graph $G = (U, V, E)$, we say that two nodes in U are *k-neighbors* if there exist at least k distinct length-two paths between them; this defines a *k-neighborhood graph* on U where the edges are given by the k-neighbor relation. For example, in a bipartite graph of users and interests, two users are k-neighbors if they have at least k common interests. The KNC-plot shows the degradation of connectivity of the graph as a function of k. We show that this tool provides an effective and interpretable high-level characterization of the connectivity of a bipartite graph.

However, naive algorithms to compute the KNC-plot are inefficient for $k > 1$. We give an efficient and practical algorithm that runs in sub-quadratic time $O(|E|^{2-1/k})$ and is a non-trivial improvement over the obvious quadratic-time algorithms for this problem. We prove significant improvements in this runtime for graphs with power-law degree distributions, and give a different algorithm with near-linear runtime when V grows slowly as a function of the size of the graph.

We compute the KNC-plot of four large real-world bipartite graphs, and discuss the structural properties of these graphs that emerge. We conclude that the KNC-plot represents a useful and practical tool for macroscopic analysis of large bipartite graphs.

Categories and Subject Descriptors

G.2.2 [**Discrete Mathematics**]: Graph Theory—*Graph algorithms*; H.2.8 [**Data Management**]: Database Applications—*Data Mining*; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Computations on discrete structures*; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Clustering*

WSDM'08, February 11–12, 2008, Palo Alto, California, USA.

General Terms

Algorithms, Experimentation, Measurement

Keywords

bipartite graphs, connectivity, connected components

1. INTRODUCTION

The use of graph-theoretic tools and techniques to study massive data has been tremendously successful in improving our understanding of the latent properties of the data. Graphs have been applied to modeling a wide range of phenomena, ranging from the internet and the world wide web to social networks and protein interactions. Depending on the application domain, an appropriate type of graph must be selected to model the data. In this paper we focus on bipartite graphs in which the nodes of the graph may be split into two groups and the edges must connect a member of one group to a member of the other group. Bipartite graphs have been employed to capture the relationships between users and groups, terms and queries, web pages and topics, as well as many others.

Once data has been modeled as a graph, two forms of analysis may be applied. In microscopic analysis, the goal is to discover interesting structures in the graph, such as directed bipartite cliques [17], dense subgraphs [14], or localized communities [12, 2]. Typically, the existence of such an object may be proven by simply exhibiting the object, meaning that witnesses for microscopic structures are very small. In macroscopic analysis, on the other hand, the goal is to uncover high-level properties of the entire graph, such as its connectivity structure [5], clustering coefficient [19], or degree distribution [3]. These analyses often seek to bring a global understanding of the data. There are no commonly-applied high-level macroscopic analyses that make strong use of the structure of bipartite graphs. We attempt in this paper to provide such an analysis.

We begin with an analogy. For directed graphs, the *bow tie model* [5] breaks the graph into four regions. First, the *SCC* of the graph is the largest strongly connected component; that is, the largest set of nodes such that each pair is connected in both directions. Second, there are regions called *IN* and *OUT*, which correspond to all nodes that can reach the SCC, or be reached from the SCC, respectively. Finally, the remaining nodes are classified as *TENDRILS* or in degenerate cases, *DISCONNECTED*. Broder et al. [5] show that for a large crawl of the web graph, these regions capture almost all the pages and have roughly equal size.

This simple and succinct model is a useful tool in thinking about the web graph. For example, it is clear from the model that about half the graph is not reachable from a web crawler that begins with well-known seed pages. Also, it is clear that for a uniformly chosen start and end node, the probability that a path exists between them is around 1/4, which is much smaller than we might guess. Similarly, the bow tie makes it clear that it is not possible to add self-loops and a few edges to the graph in order to make it ergodic. Such a model helps us to think about high-level properties of the data, and hence is successful as a macroscopic analysis.

In attempting to map this model to bipartite graphs, we immediately encounter difficulties. For most natural bipartite graphs, edges are either undirected or are directed from one bipartition to the other. Thus, the component structure is trivial — almost all nodes participate in a single giant component, and there is no meaningful notion of strong connectivity. To generate a macroscopic picture of a bipartite graph, we must look elsewhere.

For general graphs, there are notions of the extent to which two nodes are connected, but these notions fall into two categories. First, there are clean combinatorial definitions such as k-connectivity, but for $k > 1$, these are infeasible to compute for large graphs. And second, there are continuous notions of connectivity based on random walks or flows [18, 2], but these require large global computations and are somewhat difficult to interpret in a simple way.

For bipartite graphs, however, we argue that there is a natural notion of two nodes being connected more or less strongly. Consider a bipartite graph with users on the left and interests on the right. We claim that two users who share interests in photography, renaissance baking, and random walks are more strongly connected than two users who share an interest in photography alone. In this paper we adopt the cleanest form of this definition: users with k interests in common are more strongly connected than those with $k-1$ interests in common. It is possible to extend our model to more nuanced versions of connectivity that take into account the relative prevalence of different interests in the general population.

k-neighborhood graph. We therefore propose the following model. We say that two nodes in a bipartition are k-neighbors if they share at least k neighbors; that is, two users are *k-neighbors* if they have at least k interests in common. We define the *k-neighborhood graph* G_k on the nodes of one bipartition as follows: two nodes have an edge in G_k if and only if they are k-neighbors in the original bipartite graph. We expect the following behavior. G_1 should be a very highly connected graph with high edge density. G_2, defined on the same nodes, may be less well connected, and will have a subset of the edges of G_1. For larger k, G_k will become increasingly sparse, and its connectivity structure will begin to deteriorate, until it becomes completely disconnected. By analyzing G_k, we can tell whether the amount of shared interests between two users is common or not, and perhaps more importantly, whether restricting to this level of connectivity would result in a graph with many well-connected users, or a large number of tiny islands, which would imply that our two users should be seen as well-connected relative to other users in the graph. Thus, we argue that the structure of G_k is a useful tool in thinking about the nature of a bipartite graph. We define the k-neighborhood connectivity plot, or KNC-plot, as the graph showing some measure of the connectivity of G_k as a function of k. In the results that follow, we will plot the size of the largest component (decreasing) and the number of components (increasing), but we could also include the entropy of the component distribution, the size of the second component, and so forth.

Notice that this approach reflects the asymmetry in naturally-occurring bipartite graphs. One can define the neighbor graphs G_k on users, or on interests (two interests are neighbors if many users share both interests). The KNC-plot in each case may be completely different, and may unearth different aspects of the data.

Given this simple model, the technical challenge is to compute the KNC-plot in an efficient manner. Surprisingly, even for $k = 2$ the problem is nontrivial, and for large graphs, the natural approaches are ineffective. In this paper we study approaches to this problem that have good asymptotic performance for all graphs, but that provably perform very efficiently for commonly-occurring graphs including those with power law degree sequences.

Technical contributions. Our main contribution is the definition of the KNC-plot as a tool to understand the macroscopic structure of a bipartite graph, and the development of an efficient and practical algorithm to compute the KNC-plot. Specifically, we present an algorithm that can compute the KNC-plot in time $\tilde{O}(m^{2-1/k})$, where m is the number of edges in the underlying bipartite graph. This sub-quadratic running time is a non-trivial improvement over the naive quadratic algorithms for this problem. Furthermore, our algorithm is simple and the constants in the $\tilde{O}(\cdot)$ small enough to permit an extremely efficient implementation in practice.

Technically, our algorithm is developed by carefully combining two relatively simpler algorithms for the problem. The first algorithm works well when the total number of edges in the graph is small, regardless of the maximum degree of the nodes. The second algorithm works well when the maximum degree of nodes on the left-hand side is bounded. By choosing an appropriate degree threshold to implicitly decompose the k-neighborhood graph, we run the two algorithms in tandem on the two decompositions and combine the results of these independent runs to ascertain the connected components of the entire k-neighborhood graph.

We also provide much improved algorithms for two special cases that are of practical interest. The first is when the degree of the nodes are distributed according to a power law. In this case we show that the running time of our algorithm improves favorably with an increased skewness of the power law. The second is when the right-hand side of the bipartite graph is relatively small. In this case using a completely different approach, we obtain a near-linear time algorithm for the problem.

We apply our algorithm to generate the KNC-plot of four large real-world bipartite graphs. The first is a subgraph of the user–interests relation from the LiveJournal blogspace. The second is a subgraph of the user–queries relation from queries performed on Yahoo! web search. The third is a subgraph of the page–ads relation from the Yahoo! content match data. The fourth is a subgraph of the photo–tags relation from Flickr. The experiments show that our algorithm is extremely practical, and its efficient running time makes it even possible to study the KNC-plot of these large graphs. We also implement the naive algorithm to demonstrate the actual running time improvements.

Organization. Section 2 discusses the related work. Section 3 contains the formal problem description and the basic notation used throughout the paper. Section 4 presents the development of our main algorithm and its proof. It also presents improvements to the main algorithm for two special cases. Section 5 contains the experimental results on two large real-world bipartite graphs. Finally, Section 6 contains concluding remarks.

2. RELATED WORK

Perhaps the closest family of work related to this paper is those on understanding massive graphs that arise in the context of the internet and the world-wide web. In a seminal paper, Faloutsos et al [10] studied the internet topology and discovered that the indegrees of nodes were distributed according to a power law. This phenomenon was also observed subsequently in the web graph, i.e., directed graph corresponding to hyperlinks on the world-wide web [5, 3]. As mentioned earlier, Broder et al. [5] tried to characterize the web graph via the bow tie model. Dill et al. [8] refined this model to account for the self-similar nature of various logical sections of the web graph.

Another line of work that is related to ours is that of finding dense subgraphs in massive graphs. Kumar et al. [17] studied the problem of finding dense communities in the web graph. To this end, they developed efficient heuristic algorithms to quickly identify dense bipartite subgraphs in the web graph; see also [13]. Flake et al. [12] adopted a network flow approach to the problem of identifying communities. Gibson et al. [14] developed a technique using hashing and shingling to identify large dense subgraphs in massive graphs, later extended by Dourisboure et al. [9]. For a more theoretical treatment of the dense subgraph problem, see the paper by Charikar [6].

A third line of work related to our paper is the problem of partitioning and representing a graph into bipartite cliques or other succinct representations. This problem was considered by Feder and Motwani [11], who obtained a "compressed" representation of the graph and used it to speed up basic graph algorithms for matching, node connectivity, edge connectivity, and shortest paths. A slightly different notion of graph compression was studied by Kao et al. [15]. Agarwal et al. studied the problem of representing the visibility graph of line segments as a union of cliques and bipartite cliques [1]. For more references and a survey of the area, see the thesis by Bezakova [4]. Our problem is also somewhat related to the bit vector intersection problem studied by Karp et al. [16], where given a large collection of sparse bit vectors, the goal is to find all the pairs with at least k ones in common, for a given parameter k. The assume that the number of ones common to any two vectors is significantly smaller than k, except for an unknown set that is linear in the size of the collection.

3. PRELIMINARIES

Let $G = (U, V, E)$ be a bipartite graph with set of nodes U on the left-hand side, and a different set of nodes V on the right-hand side. Let $n = |U|$ be the number of nodes in U and let $m = |E|$ be the number of edges. For simplicity, we will assume that $|V| = O(n)$ and $m = n \cdot \text{poly}\log(n)$.

The goal of this work is to understand the connectivity structure of this bipartite graph, parametrized by a connectivity requirement quantity, namely, k. First, we establish a notion of connectivity that we use throughout the paper.

DEFINITION 1 (k-NEIGHBOR). *Given a bipartite graph $G = (U, V, E)$ and $k \geq 1$, the pair of nodes $(u, u') \in U$ in G are said to be* k-neighbors *in G if there are distinct nodes $v_1, \ldots, v_k \in V$ such that $(u, v_i) \in E$ and $(u', v_i) \in E$ for every $i = 1, \ldots, k$.*

In other words, the two nodes u and u' are k-neighbors if there exists at least k distinct length-two paths between them. By definition, k-neighbor is a symmetric relation.

DEFINITION 2 (k-NEIGHBORHOOD GRAPH). *Given a bipartite graph $G = (U, V, E)$ and $k \geq 1$, the* k-neighborhood graph *$G_k = (U, E_k)$ is a graph such that $(u, u') \in E_k$ if and only if (u, u') are k-neighbors in G.*

Figure 1 shows an example of k-neighborhood graphs of a particular bipartite graph. Notice that G_k is by definition undirected. The KNC-plot shows statistics of the component structure of G_k as a function of k; in the following section, we give efficient algorithms to produce this plot.

Notation. For a node $w \in U \cup V$, let $\Gamma(w) = \{w' \in U \cup V \mid (w, w') \in E\}$ be the set of nodes adjacent to w. Let $\deg(w) = |\Gamma(w)|$ be the degree of the node w.

For a subset $S \subseteq U$, let $G_{k,S} = (U, E_k \setminus (\overline{S} \times \overline{S}))$ be the node-induced subgraph of G_k and let $G_{k,\overline{S}} = (\overline{S}, E_k \cap (\overline{S} \times \overline{S}))$. In words, if S is a subset of nodes, then $G_{k,S}$ contains the edges that are incident on at least one node in the subset and $G_{k,\overline{S}}$ contains only the edges between nodes not in the subset. It is easy to see that $G_{k,S}$ and $G_{k,\overline{S}}$ fully determine G_k and so the connected components of G_k can be inferred easily from the connected components of $G_{k,S}$ and $G_{k,\overline{S}}$.

Clique cover view. An alternate view of the problem setting is the following. Let $U = \{u_1, \ldots, u_n\}$ denote the nodes of an undirected graph. Let $\mathcal{C} = \{C_1, \ldots, C_\ell\}$, where $C_i \subseteq U$; here, the subset C_i is to be thought of as implicitly defining a clique on the nodes of C_i. Given a parameter k, we define a graph G_k implicitly in the following way: an edge (u, u') in the graph $G_{\mathcal{C},k}$ is given by the predicate $|\{i \mid u, u' \in C_i\}| \geq k$. In other words, the edge (u, u') is present in G_k if and only if at least k distinct cliques in $\mathcal{C}$ contain both u and u'. There is a natural correspondence between the bipartite view and this clique cover view. Given a bipartite graph, set $\ell = |V|$ and $\mathcal{C} = \{C_v\}$ where $C_v = \{u \mid (u, v) \in E\}$. The goal now is to obtain efficient algorithms for computing the connectivity of G_k, for instances, without explicitly enumerating all the edges in all the cliques in $\mathcal{C}$.

4. ALGORITHMS

In this section we develop efficient and practical algorithms for finding the connected components and connectivity structure of the k-neighborhood graph for $k \geq 1$. At first blush, this problem looks easy since this can be solved by explicitly constructing G_k and then performing a breadth-first search on G_k. However, this is not efficient since this algorithm will take time $\Omega(n^2)$, even to implicitly construct G_k. We seek algorithms that run in time $o(mn)$; since we assume $m = \tilde{O}(n)$, this amounts to running times that are $o(m^2)$. We use $\tilde{O}(\cdot)$ notation to ignore factors that are logarithmic in m and n.

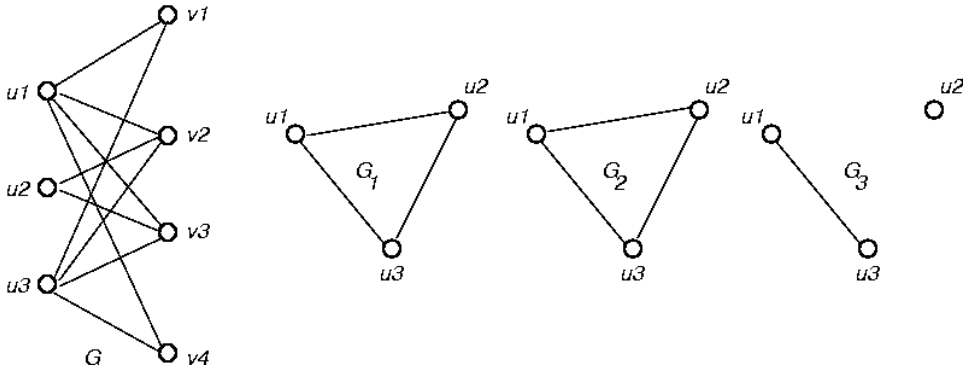

Figure 1: Example figure showing a bipartite graph G and its k-neighborhood graphs for $k = 1, 2, 3$.

In Section 4.1, we state our main result, which is a sub-quadratic algorithm for finding the connected components in G_k. Our result is built upon two simple quadratic-time algorithms for the problem described in Section 4.2 and Section 4.3. Using these algorithms, we develop the sub-quadratic time algorithm in Section 4.4 and prove a bound on its running time. We then consider two special cases that are of interest in practice. First, suppose the degree distribution for the nodes on the left-hand side conforms to a power law. In Section 4.5, we provide a sharper analysis of our algorithm in terms of the exponent of this power law. Second, suppose the right-hand side is logarithmically smaller compared to the left-hand side. In Section 4.6, we provide an (entirely different) algorithm that runs in almost linear time.

4.1 Main result

First if $k = 1$, then the problem becomes easy. The only observation is that there is a bijection between the connected components of G and the connected components of G_1. Hence by running a breadth-first search on G, it is easy to compute its connected components and hence that of G_1. The time taken is $O(m+n)$, which is linear in the input size. The difficulty in the problem arises in the case of $k > 1$.

We show the following result.

THEOREM 3. *There is an algorithm to compute the connected components of $G_k, k > 1$ that runs in time $\tilde{O}(m^{2-1/k})$.*

Note that even though the running time approaches $O(m^2)$ as k increases, it is still beneficial for large values of m (which is of practical interest) even for reasonably moderate values of k. A second advantage comes from the fact that we are able to completely analyze the components of G_k for small k. To obtain the components of G_{k+1}, we can run our algorithm on each component from G_k. Although this does not improve asymptotic running time in the worst case, it appears to help for many graphs, since these components can potentially be much smaller than the original graph.

Before proving this main result, we first develop two natural but naive algorithms for this problem.

4.2 First naive algorithm: ALG-INTERSECT

The first algorithm we consider, called ALG-INTERSECT, works in the following manner.

The main idea is to see if the intersection size of the neighborhood of two nodes is at least k, i.e., if an edge between the two nodes exists in G_k. If this predicate can be computed easily and on the fly, then the connectivity on G_k can be solved by performing a breadth-first search on U with edges in E_k accessed as needed.

To be useful later, we show the following more general result. In this case we specify a subset $S \subseteq U$ and the goal is to obtain the connected components of $G_{k,S}$. So, the question boils down to checking if the intersection of the neighborhood of a node $u \in S$ and another node $u' \in U$ is at least k. We show how to do this efficiently. For each node $u \in S$, we store $\Gamma(u)$ as a hash table (or a Bloom filter). Now for another node $u \neq u' \in U$, we need to check if the sets $\Gamma(u)$ and $\Gamma(u')$ intersect in at least k places.

LEMMA 4. *Given $G = (U, V, E)$, $S \subseteq U$, and any $k > 1$, the algorithm* ALG-INTERSECT *computes the connected components of $G_{k,S}$ in time $O(|S| \cdot |E|)$ and using space $O(|U| + |E|)$.*

PROOF. The correctness of the algorithm is immediate. We need to argue about the running time. Since $\Gamma(u)$ is stored as a hash table, checking if $|\Gamma(u) \cap \Gamma(u')| \geq k$ can be done in time $|\Gamma(u')| = \deg(u')$.

Over the entire run of the breadth-first search, the total time spent per node u is at most $\sum_{u' \in U} \deg(u') = |E|$. Therefore, this algorithm runs in time $O(|S| \cdot (|E| + |U|))$. Note that this is actually $O(|S| \cdot |E|)$ as long as there are no isolated nodes in G. It is also easy to see that the space used by the algorithm is $O(|U| + |E|)$. □

Observe that the running time of the above algorithm is independent of k. Also this simple algorithm is especially appealing if the graph G has few nodes of very high degree since the running time depends only on the number of nodes of high degree and the sum of degrees. We will exploit this fact later.

4.3 Second naive algorithm: ALG-TUPLE

The second algorithm we consider, called ALG-TUPLE, works in the following manner.

The main idea is to perform a breadth-first search on an implicitly defined bipartite graph whose right-hand side consists of all possible k-tuples of V and whose edges represent the adjacency of u to *all* the nodes of the k-tuple. Formally, let $G'_k = (U, \mathcal{V}_k, E'_k)$ where $\mathcal{V}_k = \{V' \subseteq V \mid |V'| = k\}$ and $(u, V') \in E'_k$ for $V' = \{v_1, \ldots, v_k\}$ if and only if $(u, v_i) \in E$ for $i = 1, \ldots, k$. Clearly, $|\mathcal{V}_k| = \binom{|V|}{k}$. It is easy to see that

u and v are in the same connected component in G_k if and only if they are in the same connected component in G'_k.

An efficient way to implement this scheme would be to construct a k-tuple of neighbors for every $u \in U$. Now, by either sorting or hashing the tuples, we may determine whether two nodes in U share a tuple and if so, identify them in a single component.

LEMMA 5. *Given $G = (U, V, E)$ and any $k > 1$, the algorithm* ALG-TUPLE *computes the connected components of G_k in time $\tilde{O}(\sum_{u\in U} \deg(u)^k)$ and in space $O(\sum_{u\in U} \deg(u)^k)$).*

PROOF. As before, the correctness is obvious and we need to argue about the running time and space. Constructing the k-tuple of neighbors for each $u \in U$ takes time (and space) $\sum_u \binom{\deg(u)}{k} = O(\sum_u \deg(u)^k)$. And identifying if two nodes in U share a tuple or not can be done in time $\tilde{O}(\sum_u \deg(u)^k)$ time. □

The above algorithm is especially valuable if all the degrees in the graph are small. In particular, if the maximum degree is at most c, then the running time of the algorithm is roughly $\tilde{O}(c^k|U|)$; this can be substantially smaller than $O(|U| \cdot |E|)$ if c is small. Again, we will exploit this factor later.

Note that the naive implementation of ALG-TUPLE given in the proof above uses space $O(\sum_{u\in U} \deg(u)^k)$. It is possible to improve this space to $O(km)$ by using the following simple observation: the edges of G_k are a subset of edges of G_{k-1}. Hence, for a given k, we can first pick a node in V, and consider only the subgraph induced by nodes in U that are adjacent to v. We recursively solve the problem on this induced subgraph for $k-1$. Using the proper data structures, the running time is $O(\sum_u \deg(u)^k)$, while the space drops to $O(km)$.

4.4 Proof of the main result

In this section we establish the main result by providing an algorithm ALG-HYBRID. Our algorithm will be a careful combination of ALG-INTERSECT and ALG-TUPLE. For simplicity of exposition, we treat the running time of ALG-TUPLE to be $O(\sum_{u\in U} \deg(u)^k)$, i.e., we ignore the logarithmic terms. Let $u_1, \ldots, u_n$ be the nodes in U sorted in decreasing order of the degrees.

The algorithm ALG-HYBRID first determines a break-point $b \in [0, n+1)$ such that

$$\sum_{i=1}^{b} \deg(u_i)^k \geq b \cdot m$$

and

$$\sum_{i=b+1}^{n} \deg(u_i)^k \leq (b+1) \cdot m.$$

Note that such a break-point always exists since u_i's are sorted and the left-hand side term of the first inequality is an increasing function of b while the left-hand side of the second inequality is a decreasing function of b. For sake of simplicity, we assume that b in fact satisfies the equality

$$\sum_{i=b}^{n} \deg(u_i)^k = b \cdot m, \tag{1}$$

and let $S = \{u_1, u_2, \ldots, u_b\}$. By construction, S picks out the highest degree nodes in U and the above equation will help us balance the cost of ALG-INTERSECT and ALG-TUPLE.

Now, we run ALG-INTERSECT on $G_{k,S}$ and ALG-TUPLE on $G_{k,\overline{S}}$. The break-point b achieves a trade-off between the number of edges in the graph supplied to ALG-INTERSECT, which is its bottleneck, and the maximum degree in the graph supplied to ALG-TUPLE, which is its bottleneck. As mentioned earlier, the connected components in $G_{k,S}$ are found by ALG-INTERSECT and the connected components in $G_{k,\overline{S}}$ are found by ALG-TUPLE. It is then straightforward to merge these components. The total running time is

$$O(bm) + \tilde{O}(\sum_{u\in\overline{S}} \deg(u)^k), \tag{2}$$

which follows from Lemma 4 and Lemma 5.

The correctness of this algorithm is immediate. We need to argue about the running time. To do this, we first make the following simple yet crucial observation.

LEMMA 6. *For all $i = 1, \ldots, n$, $\deg(u_i) \leq |E|/i$.*

PROOF. This follows by a counting argument. Suppose for some i we have $\deg(u_i) > |E|/i$. Then, we have the following sequence of inequalities leading to a contradiction.

$$|E| = \sum_{j=1}^{n} \deg(u_j) \geq \sum_{j=1}^{i} \deg(u_j) \overset{(a)}{>} \sum_{j=1}^{i} \frac{|E|}{i} = |E|.$$

Here, (a) follows since the nodes are sorted by the degrees and by our assumption on $\deg(u_i)$. □

With this observation, the proof of the running time is easy. We may now state the proof of Theorem 3.

PROOF. Recall our choice of b in (1). From Lemma 6, we know that $\deg(u_i) \leq m/i$.

Suppose the equality holds, i.e.,

$$\deg(u_i) = m/i. \tag{3}$$

In this case (1) leads to

$$m^k \sum_{i=b}^{n} \frac{1}{i^k} = b \cdot m.$$

Therefore,

$$b = O\left(\frac{m^{1-1/k}}{(k-1)^{1/k}}\right). \tag{4}$$

From (2), we now obtain the final running time to be $\tilde{O}(m^{2-1/k})$.

In case when (3) does not hold, it is easy to verify that (4) still holds and the running time is still $\tilde{O}(m^{2-1/k})$. In other words, the worst case for the running time is attained when $\deg(u_i) = m/i$. □

4.5 Special case: Power law degrees

In this section we consider an interesting special case in which the degrees of nodes in U are distributed according to a power law with exponent $\alpha \geq 1$, i.e, the number of nodes of degree i is roughly m/i^α. (For ease of exposition, we omit the normalizing constants.) We show the running time of the algorithm as a function of the exponent α of this power law. Specifically, we show the following corollary of our main theorem.

COROLLARY 7. *Suppose the degrees of U are distributed according to a power law with exponent $\alpha \geq 1$. Then, there is an algorithm to compute the connected components of $G_k, k > 1$ that runs in time $\tilde{O}(m^{1+(1-1/k)/\alpha})$.*

PROOF. The proof mirrors that of Theorem 3. We compute the appropriate break-point b using the expression

$$\sum_{i=b}^{n} \left(\frac{m}{i^\alpha}\right)^k = bm.$$

From this, we get

$$b = \frac{m^{(1-1/k)/\alpha}}{(\alpha k - 1)^{1/(\alpha k)}}.$$

As before, using (2), the running time is $\tilde{O}(m^{1+(1-1/k)/\alpha})$. □

Thus, if the degrees are skewed with $\alpha > 1$, then the running time of our algorithm is bounded away from being quadratic.

4.6 Special case: Small right-hand side

In this section we consider another interesting special case when $|V| \leq \log |U| = \log n$. We demonstrate a completely different and almost linear time algorithm for the problem. The running time of this algorithm is independent of the value of k; in fact, it computes the components of G_k for every k.

LEMMA 8. *Suppose $|V| \leq \log n$. Then, there is an algorithm to compute the connected components of $G_k, k > 1$ that runs in time $\tilde{O}(n)$.*

PROOF SKETCH. The algorithm works on the lattice L of bit vectors defined by all possible subsets of nodes of V; the lattice operations are defined by the bit-wise boolean AND and OR. Since $|V| \leq \log n$, the size of this lattice is at most n. Let $|x|$ denote the Hamming weight of $x \in L$.

Now, each node $u \in U$ maps to exactly one element of the lattice depending on the characteristic vector of $\Gamma(u)$. For any lattice element $x \in L$, let $W(x)$ denote the (possibly empty) set of nodes in u that map to x. For any node $x \in L$, let $\text{anc}(x)$ denote the set of ancestors of x in L, i.e., $\text{anc}(x) = \{y \in L \mid |y| = |x| + 1 \wedge \forall i, y_i \geq x_i\}$.

Now, we scan this lattice in level order, starting from the all ones element, which is the supremum. Let $B_i = \{x \in \{0,1\}^n \mid |x| = i\}$ be the set of vectors at level i. For each element $x \in B_i$, we update $W(x) \leftarrow W(x) \cup \bigcup_{y \in \text{anc}(x)} W(y)$. Now, the connected components in G_k are given by the set $\{W(x) \mid |x| = k\}$, with possible duplicates removed.

We claim that for every pair nodes u and v, we have $u, v \in W(x)$ if and only if u and v are in the same component in $G_{|x|}$. This can be proved by induction; we omit the proof in this version. It is also easy to see that this algorithm can be implemented in essentially $\tilde{O}(n)$ time, the overhead above linear time being the time necessary to perform union–find. □

5. EXPERIMENTS AND RESULTS

In this section we generate the KNC-plot for four large real bipartite graphs. We implemented ALG-HYBRID using under 1000 lines of Java code. For each of the graphs, we run our algorithm and compute the connected components. We extract the size of the largest component and the number of connected components in the graph G_k, as a function of k, and plots the results in the KNC-plot. The shape of these curves will be an indication of the bipartite structure of the graph. Typically, we run our algorithms on bipartite graphs with $|U|$ roughly equal to 10^5 or 10^6. In some cases, the original graphs may be larger than this, and so we employ a form of downsampling, as follows. To generate a subgraph of size s, we randomly select s nodes from U without replacement, and then select all edges incident to the selected nodes, and all the associated elements of V.

Dataset	U	V	E
User–interests	897K	287K	13M
User–queries	95K	42K	3.9M
Page–ads	358K	313K	8.6M
Photo–tags	42K	19K	189K

Table 1: Statistics of data sets.

There are two primary application domains for our algorithm. The first is to provide a natural and easily-described hierarchical clustering of the nodes of one bipartition. This comes about because the components of G_{k+1} are guaranteed to be subsets of the components of G_k, so a hierarchical clustering is induced. From this hierarchical clustering, standard techniques may then be applied to derive affinity measures between nodes in the graph, and so forth. The second application is to produce the KNC-plot as a macroscopic summary of the graph, to be used for data forensics, anomaly detection, and generating a high-level understanding of the data set. In this section we will explore the second application only — this application must introspect on the properties of the various neighborhood graphs, and thus provides a novel perspective.

We first describe the datasets we use, then give some information about the downsampled graphs we study.

User–interests in LiveJournal: Our first bipartite graph is derived from the user–interest graph obtained from the LiveJournal blogspace. LiveJournal is an online blogging community where each user in the social has a profile that may contain a self-reported list of interests. We use a crawl of the users of LiveJournal from May 2006.

User–queries in Yahoo! query log: Our second graph is derived from the query logs inside Yahoo! We obtained query logs consisting of anonymized user ids and the queries issued by the user over a 75 day period.

Page–ads graph in content match: Our third graph is derived from the content match program at Yahoo! Here, the data corresponds to which ads were shown on which pages.

Photo–tags in Flickr: Our fourth graph is derived from the Flickr data set. Flickr is an online photo sharing site where each photo can be tagged with free text.

Table 1 shows the statistics for the downsampled graphs studied in each of the four cases.

We now turn to some aggregated statistics that are very helpful in understanding the nature of each of the bipartite graphs. Recall that $n = |U|$. Table 2 shows for each data set the value of k for which the largest component in G_k is $n/2$,

Dataset	50%	10%	1%
User–interests	5	14	36
User–queries	16	> 50	> 50
Page–ads	9	27	> 50
Photo–tags	3	4	6

Table 2: Values of k at which largest component in each dataset represents 50%, 10%, and 1% of the original graph.

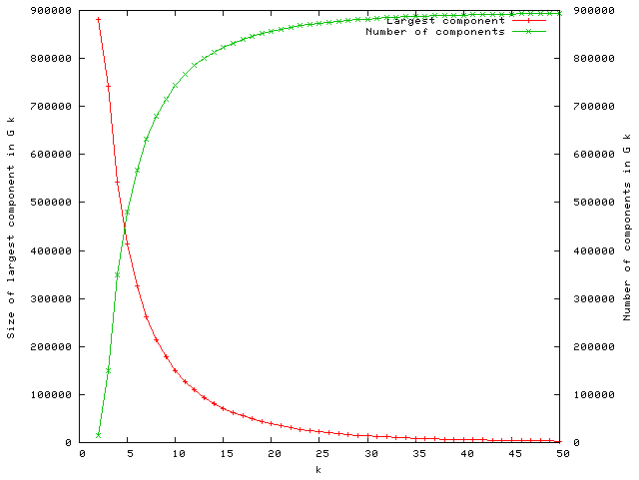

Figure 2: KNC-plot of user-interests graph.

$n/10$ and $n/100$. We will give a more detailed discussion of each dataset, with pointers back to this table.

Figure 2 shows the KNC-plot of the user–interests graph. As Table 2 shows, the largest component of G_{14} has size $n/10$, and that of G_{36} has size $n/100$. Thus, one should imagine that the 1% of users who are connected together by dint of sharing thirty-six distinct interests with others in this group are in fact a highly unusual collection, with strong affinity compared to others in the graph. Approaches that connect users based on, say, six common interests should be viewed with suspicion, as this level of connectivity is sufficient to (transitively) interconnect half the population. Finally, the dynamic range of the graph is such that sharing of interests ranging from a handful all the way to thirty or more all represent meaningful levels of connectivity that can be responsible for inducing nontrivial structure.

Figure 3 shows the KNC-plot of the user–queries graph. Here, the story is very different. As Table 2 shows, 50% of the users are connected in G_{16}, and more than 10% (around 20%) are connected in G_{50}. This bipartite graph shows a very different structure from the user–interests graph. In a sense, it should be viewed as "heavy-tailed," meaning that the level of connectivity between users based on shared queries may meaningfully be expanded all the way from just a few queries to in excess of fifty queries, with only a slow diminution of the number of connected users. This is surprising, given that the average user issued only forty queries overall, and the queries are highly diverse.

Figure 4 shows the KNC-plot of the page–ads graph. This graph displays a third distinct pattern, in which five shared

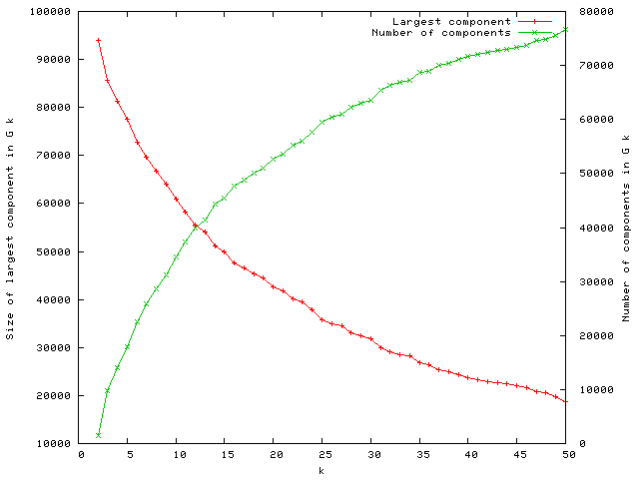

Figure 3: KNC-plot of user-queries graph.

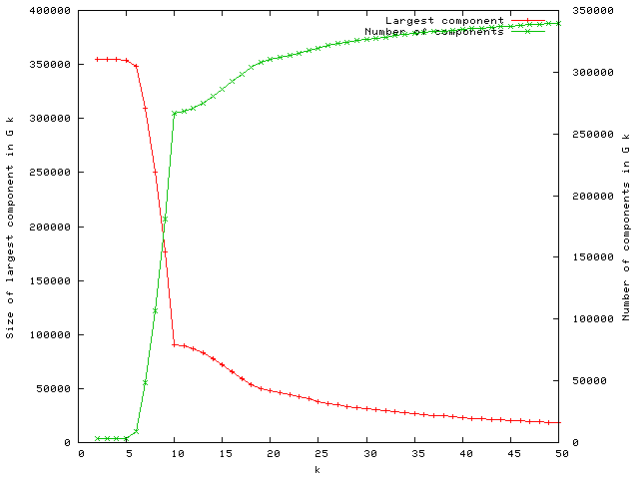

Figure 4: KNC-plot of page-ads graph.

ads says almost nothing about the level of connection between two pages, while six shared ads makes a much stronger statement. In this example, unlike the previous two graphs, there is a sharper threshold behavior, and then a heavy tail in which 20% of the pages may be connected by six common ads, and half of this set remains connected even when requiring twenty-seven common ads.

Finally, Figure 5 shows the KNC-plot of the photo–tags graph. Once again, this graph shows yet a fourth pattern of connectivity. As we construct G_3, G_4, and G_5, the size of the largest connected component drops aggressively, to the point that requiring six common tags to connect two photos will result in fewer than 1% of photos being connected. For this graph, we may view the higher-level bipartite connectivity as essentially non-existent.

5.1 Running times: $k = 2$, increasing n

To clearly illustrate the benefits of the algorithm in terms of its improvement over a naive algorithm, we compare our algorithm Alg-Hybrid with Alg-Intersect for $k = 2$.

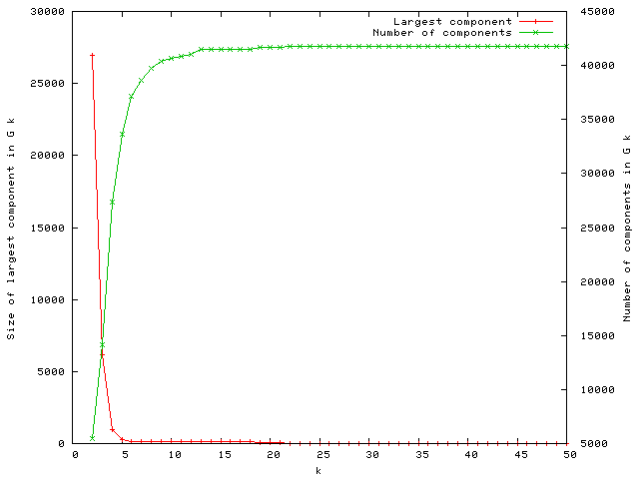

Figure 5: KNC-plot of photo–tags graph.

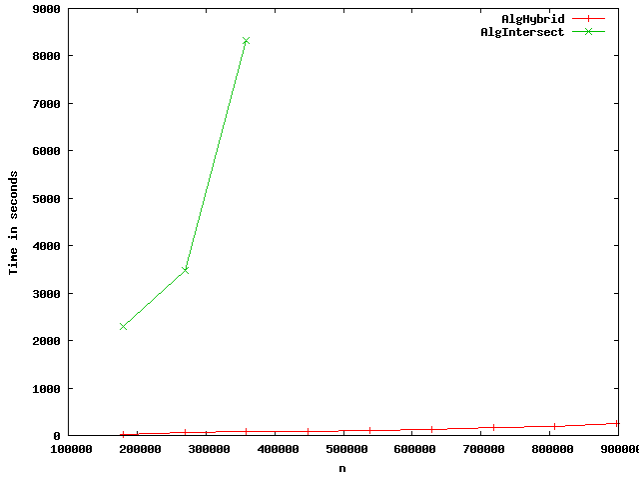

Figure 6: Running time of ALG-HYBRID and ALG-INTERSECT for $k = 2$ and different values of n.

Figure 6 shows the running times for various values of n; the underlying subgraphs were sampled from the LiveJournal graph. It is clear from the figure that even for moderate values of n, the naive algorithm becomes infeasible whereas ALG-HYBRID remains under five minutes even for n close to 10^6.

5.2 Running times: fixed n, increasing k

We next compare the performance of ALG-HYBRID with ALG-INTERSECT for larger values of k. Table 3 shows the running times of our experiments for $k = 2, \ldots, 50$. From the table, we see that even for this high value of k, ALG-HYBRID does (in some cases, substantially) better than the naive ALG-INTERSECT. This demonstrates the effectiveness and efficiency of our algorithm, even as a function of k.

6. CONCLUSIONS

In this paper we studied the connectivity structure of massive bipartite graphs. We developed a notion two nodes being k-neighbors if they have at least k disjoint length two paths among them and used this notion to define the k-neighborhood of a given bipartite graph. From this definition, we introduced the KNC-plot, showing falloff in connectivity as a function of k. The KNC-plot sheds light on the connectivity aspects of the bipartite graph and we believe might be a valuable tool in understanding the macroscopic properties of such graphs.

Dataset	n	m	ALG-HYBRID	naive
photo–tags	41,811	189,159	7.4m	39.6m
page–ads	35,920	845,112	1m34s	1m30s
user–interests	269,591	3,918,820	36m33s	62m51s
user-queries	95,399	3,874,271	20m23s	24m2s

Table 3: Comparative running times of the ALG-HYBRID and ALG-INTERSECT for $k = 2, \ldots, 50$.

We developed efficient algorithms to compute the connected components of the k-neighborhood graph, and hence the KNC-plot. Our algorithms run in sub-quadratic time and are extremely practical. We also study improved algorithms and improved analysis for some commonly occurring special cases. We apply our algorithms to four large bipartite graphs, namely, the user–interest graphs from the LiveJournal blogging community, the user–queries graph from Yahoo! search, the page–ads graph from Yahoo! content match, and the photo–tags in Flickr. Experiments show that our algorithm is extremely practical, and its efficient running time makes it possible to study the KNC-plot of these large graphs.

Future work include improving the running time of our current algorithms. Another interesting direction to pursue is to develop a data stream version of our algorithms, where we make one or more passes over the graph and output some function of the connected components (say, the number of connected components). The paper of Charikar et al. [7] on counting distinct elements in a data stream might be relevant in this case.

7. REFERENCES

[1] P. K. Agarwal, N. Alon, B. Aronov, and S. Suri. Can visibility graphs be represented compactly? *Discrete and Computational Geometry*, 12:347–365, 1994.

[2] R. Andersen and K. Lang. Communities from seed sets. In *Proc. 15th International Conference on the World Wide Web*, pages 223–232, 2006.

[3] A.-L. Barabasi and R. Albert. Emergence of scaling in random networks. *Science*, 286:509–512, 1999.

[4] I. Bezakova. Compact representation of graphs and adjacency testing. Master's thesis, Comenius University, Bratislava, 2000.

[5] A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. Wiener. Graph structure in the web. *Computer Networks*, 33(1–6):309–320, 2000.

[6] M. Charikar. Greedy approximation algorithms for finding dense components in a graph. In *Proc. 3rd International Workshop on Approximation Algorithms for Combinatorial Optimization*, pages 84–95, 2000.

[7] M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. In *Proc. 29th*

International Colloquium on Automata, Languages, and Programming, pages 693–703, 2002.

[8] S. Dill, R. Kumar, K. S. McCurley, S. Rajagopalan, D. Sivakumar, and A. Tomkins. Self-similarity in the web. In *Proc. 27th International Conference on Very Large Databases*, pages 69–78, 2001.

[9] Y. Dourisboure, F. Geraci, and M. Pellegrini. Extraction and classification of dense communities in the web. In *Proc. 16th International Conference on the World Wide Web*, pages 461–470, 2007.

[10] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the internet topology. In *Proc. of the ACM SIGCOMM Conference on Applications, Technology, Architectures, and Protocols for Computer Communication*, pages 251–262, 1999.

[11] T. Feder and R. Motwani. Clique partitions, graph compression, and speeding-up algorithms. *Journal of Computing and System Sciences*, 51(2):261–272, 1995.

[12] G. Flake, S. Lawrence, and C. L. Giles. Efficient identification of web communities. In *Proc. 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 150–160, 2000.

[13] D. Gibson, J. Kleinberg, and P. Raghavan. Inferring web communities from link topology. In *Proc 9th ACM Conference on Hypertext and Hypermedia*, pages 225–234, 1998.

[14] D. Gibson, R. Kumar, and A. Tomkins. Extracting large dense subgraphs in massive graphs. In *Proc. 31st International Conference on Very Large Data Bases*, pages 721–732, 2005.

[15] M.-Y. Kao, N. Occhiogrosso, and S.-H. Teng. Simple and efficient graph compression schemes for dense and complement graphs. *Journal of Combinatorial Optimization*, 2(4):351–359, 1998.

[16] R. M. Karp, O. Waarts, and G. Zweig. The bit vector intersection problem. In *Proc. 36th IEEE Symposium on Foundations of Computer Science*, pages 621–630, 1995.

[17] R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins. Trawling the web for emerging cyber-communities. *Computer Networks*, 31(11-16):1481–1493, 1999.

[18] J. A. Tomlin. A new paradigm for ranking pages on the world wide web. In *Proc. the 12th International Conference on the World Wide Web*, pages 350–355, 2003.

[19] D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393(6684):440–442, June 1998.