Efficient Search Ranking in Social Networks

Monique V. Vieira[1,2]
monique@google.com

Bruno M. Fonseca[1]
brunomf@google.com

Rodrigo Damazio[1]
rdamazio@google.com

Paulo B. Golgher[1]
golgher@google.com

Davi de Castro Reis[1]
davi@google.com

Berthier Ribeiro-Neto[1,2]
berthier@google.com

[1]Google Engineering Belo Horizonte
Belo Horizonte, Brazil

[2]Computer Science Department
Federal University of Minas Gerais
Belo Horizonte, Brazil

ABSTRACT

In social networks such as Orkut, www.orkut.com, a large portion of the user queries refer to names of other people. Indeed, more than 50% of the queries in Orkut are about names of other users, with an average of 1.8 terms per query. Further, the users usually search for people with whom they maintain relationships in the network. These relationships can be modelled as edges in a *friendship graph*, a graph in which the nodes represent the users. In this context, search ranking can be modelled as a function that depends on the distances among users in the graph, more specifically, of *shortest paths* in the friendship graph. However, application of this idea to ranking is not straightforward because the large size of modern social networks (dozens of millions of users) prevents efficient computation of shortest paths at query time. We overcome this by designing a ranking formula that strikes a balance between producing good results and reducing query processing time. Using data from the Orkut social network, which includes over 40 million users, we show that our ranking, augmented by this new signal, produces high quality results, while maintaining query processing time small.

Categories and Subject Descriptors

H.3.1 [**Information Storage and Retrieval**]: Content Analysis and Indexing—*Indexing methods*;
H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Search process*

General Terms

Theory, algorithms, experimentation, performance.

CIKM'07, November 6–8, 2007, Lisboa, Portugal.

Keywords

Social networks, shortest path, graphs.

1. INTRODUCTION

Social networks are sites that allow people to interact and share social experiences through the exchange of multimedia objects (text, audio, video) associated with the people themselves and their actions. In a social network, each user creates a profile where he/she declares his/her personal information and preferences, and establishes friendship relationships with other users. These relationships define a large graph of interconnected nodes where the social interactions take place. The flexibility and convenience provided by this multimedia form of communication, magnified by the low access costs and the freedom that people now have to publish their ideas and opinions, have led to a very quick proliferation of large and powerful social network platforms. Illustrative examples of such are MySpace (www.myspace.com) with over 100 million registered users, Orkut (www.orkut.com) with over 40 million registered users, Facebook (www.facebook.com), and Friendster (www.friendster.com). Social networks are the new wave in the Web, they came to stay.

In a social network with a large number of users, a common operation is to search for people. In fact, one of the most common use cases of social networks is to reconnect to old friends and make new ones. To illustrate, in Orkut more than 50% of the searches are for other users, with an average of 1.8 terms per query. This makes searching for people a critically important part of a social network site and poses new challenges for the IR community, as the signals available for ranking documents are completely distinct from traditional Web search. Instead of links and text, there are friendship relationships, user information, and user preferences.

To demonstrate how ineffective text based ranking algorithms are in this context, we carried out a simple experiment using data from the Orkut [14] social network. We randomly selected 750 queries from the Orkut logs, with the restriction that all of them specify just user names. We then counted the average number of answers per query when the retrieval algorithm is based on an exact match between the query and the user name (i.e., for the query 'Maria', only the users who declared their names exactly as 'Maria' are

retrieved). The average number of answers is 48. In the absence of other signals, there is not much that ranking can do. As a consequence, the result page looks like a random sample of 48 answers. If we allow partial matches between the user name (in the Orkut profile) and the query (i.e., the query 'Maria' provides a match to the user named 'Maria A.'), the average number of results per query is an impressive 6,034. And again, text based ranking does not work well in this context. Thus, designing a ranking function whose results can be interpreted positively by the users is a must. Also, this function should be efficient enough to allow fast computation. This is the problem we face in our research.

We propose a ranking function for social networks based on friendship relationships. Our function takes as input distances in a *friendship graph*, a graph whose nodes represent the users and whose edges represent friendship relationships (as declared by the users of the social network). We discuss three alternatives for implementing our ranking function and show that they are too slow. We then modify our ranking function to take as input *approximate* distances in the graph of friendship relationships, based on a set of pre-selected landmark nodes (called *seeds*). To evaluate our ranking function, we run a series of experiments using real Orkut data. Our results show that effective ranking can be attained, while keeping query processing time small enough to be practical.

The paper is organized as follows. Section 2 presents a proposal for a simple (and we argue effective) ranking function for social networks. Section 3 discusses alternative algorithms for implementing this function, algorithms which are all too slow for the case of a large social network. Section 4 presents a modified ranking function, based on a set of seeds, that can be executed much faster. Section 5 discusses an algorithm for implementing this modified ranking function. Our experimental setup is discussed in Section 6, while our results are presented and analyzed in Section 7. Section 8 presents related work. Our conclusions follow in Section 9.

2. THE RANKING FUNCTION

A recent Facebook study [10] suggests that user interactions in a social network follow the same patterns as their interactions in real life, meaning that people tend to interact with people that they really know in person. That is, when a user searches for a person's name, he/she is more likely to be pleased when seeing people that are closer in the "social space" than when seeing people that are strangers. In other words, we trust the demographics of the social network as a good indicator of the users preferences. If the user John specified his query simply as 'Maria', we assume that it is more likely that he will be satisfied with the Maria who is a direct friend of him than with a Maria who he does not know and who is not known by his friends. That is, friendship relationships are an important evidence while searching for people in a social network.

To illustrate, consider the friendship graph in Figure 1. Assume that 'John' is the user logged into our social network and that he specified the query 'Maria'. There are three distinct users whose names match the label 'Maria'. Of these, 'Maria A' is a direct friend of John, 'Maria B' is friend of a friend of John, and 'Maria C' is a more distant friend. We argue that any search ranking function should favor first 'Maria A' and second 'Maria B'.

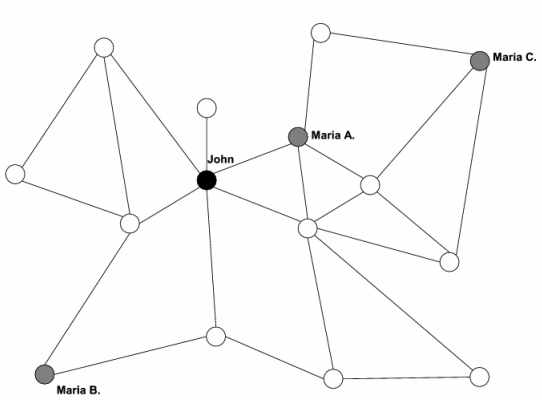

Figure 1: Friendship graph: John is at distance 1 of 'Maria A', at distance 2 of 'Maria B', and at distance 3 of 'Maria C'.

This reasoning leads us to propose the following simple ranking function:

$$rank_J(Maria) = \frac{1}{shortest\_path_J(Maria)} \qquad (1)$$

where $shortest\_path_J(Maria)$ is the shortest path between John and Maria in the friendship graph and $rank_J(Maria)$ is the rank of the user Maria, with regard to the query 'Maria' when posed by John. This rank is computed for the user names that match the query partially.

When displaying the answers to John, we can show him, for instance, six Maria's that are at a distance 1, four Maria's that are at a distance 2, and two Maria's that are at a distance 3. This way, he would have the option to expand each of these subsets for further inspection in case the person he is looking for has not been located yet. We played with this solution and found it to be much more effective and appealing than a trivial solution based simply on matching user names to the query (with no ranking). Our observation is in accordance with the trends in user behavior in large live social networks [10].

3. ALGORITHMS FOR COMPUTING THE RANKING

Consider again the friendship graph in Figure 1 and, as before, assume John is logged into the social network and has just entered the query 'Maria'. Our task is to find all Maria's with the *shortest paths* to John in the friendship graph, and use this information to rank the answers according to Equation 1. At first glance, this might look like a trivial problem. However, given the huge dimensions of live social networks (Orkut has 40 million nodes and 1,2 billion edges), simple solutions do not scale well in practice. In what follows, we present straightforward algorithms for implementing the ranking and argue that they are not practical for usage with the live social networks of today.

Algorithm 1:Pre-compute all distances

One trivial algorithm consists in pre-computing all distances between any pair of users and indexing them. This does not work because, in a graph with 40 million users, the number

of pairs of users that would need to be stored is in the order of $[40^2 \times 10^{12}]$, which is too large. One can counter argue that we do not need to store distances for all pairs. Even though, if each user has 100 friends on average[1], then the average number of friends at a distance 3 or smaller of a given user is 10^6, resulting in a still very large index size in the order of $[40 \times 10^{12}]$ pairs of friendship distances. Furthermore, pre-computing all friendship pairs presents no scalability. If the number of users is multiplied ten-fold, the number of friendship distances that needs to be stored becomes overwhelming.

Algorithm 2: On-the-fly Ranking

A second algorithm consists of calculating all the distances on-the-fly at scoring time. Consider again Figure 1. Assume John is logged in and has entered the query 'Maria'. One possible algorithm is to simply run a breadth first search for Maria starting from John. In a graph with 100 average friends per user, we would need to expand more than 10^6 users if we limit ourselves to people at most 3 hops from John. A somewhat superior alternative is to run a bi-directed breadth-first search [12], alternatively expanding the graph not only from John, but also from each Maria, and looking for intersections in the list of users reached in the new level. We then use the distances to rank the answers according to Equation 1. This algorithm, which we call *On-the-Fly Ranking*, is evaluated in Section 7.

Algorithm 3: Co-friends Ranking

A third possible algorithm is a mix of the first two: for each user, index their list of friends and at scoring time intersect both lists. This is fast and uses little space, but has a huge disadvantage: only captures friends or friends-of-friends relations.

A more promising approach is to pre-compute a friends-of-friends list for each user (which we call a *co-friends list*) and intersect it with the list of friends of the user submitting the query. For instance, if 'John' is searching for 'Maria', we should intersect John's friends with Maria's co-friends. This allows to capture friendship distances up to 3. As we discuss in our experiments, limiting friendship distances to 3 is not ideal because it hurts precision figures. Also, the space and time requirements of this approach are pretty high. This makes this approach unfeasible to be used cost effectively in a huge Web live service such as Orkut, as we later demonstrate. In our experiments, we call this solution *Co-friends Ranking*.

4. SEEDS-BASED RANKING

Since the algorithms for implementing our ranking function lack either precision of the answer or time efficiency, we propose a modified ranking function. Its key purpose is to produce results of high precision but that can be computed efficiently. This function is based on *estimates*, approximations of shortest paths and has a more complex composition than our ranking function in Equation 1, as we now discuss.

Seed Distances: Approximating Shortest Paths

For approximating shortest paths, we opt for a system of network landmarks, also called *seeds*, as proposed by various researchers [11, 6, 16, 15]. It consists of pre-determining a set of seed nodes that will serve as navigational beacons in the friendship graph. Starting from each seed, we run a breadth first search reaching out to all nodes in the network[2]. For each node reached, we annotate it with its distance to the seed we started the breadth first search from. This way, a vector of distances to seeds is associated with each node of the graph. These *vectors of seed distances* are computed offline.

The seeds have to be chosen somehow. In this paper, we use a pre-selected set of random nodes as seeds. We also explore how the precision of our ranking function varies with the number of seeds. As we later discuss, to improve ranking precision we use a much larger number of seeds than suggested in various works, exploiting properties of our graph and our specific problem. Let us proceed with an example.

Figure 2: Friendship graph with three pre-selected seeds: S1, S2, S3.

Figure 2 illustrates a modified version of our previous friendship graph in Figure 1. The difference is that now we have selected three nodes to act as navigational beacons, or seeds, indicated as S1, S2, and S3. In this case, seed distances are as follows.

Example 1 - Seed distances vectors for users in Figure 2.

$$D_{John} = [2, 1, 1] \quad (2)$$
$$D_{Maria\ A} = [1, 1, 2] \quad (3)$$
$$D_{Maria\ B} = [4, 3, 1] \quad (4)$$
$$D_{Maria\ C} = [1, 2, 4] \quad (5)$$

John is at distances 2, 1, 1 from seeds S1, S2, S3, respectively. For 'Maria A', the seed distances vector is given by $[1, 1, 2]$, and so on.

It is important to observe that, while the seeds provide information on the *relative* position of the nodes in the graph (thus, capturing structural information), they do not allow to directly compute all shortest paths. Indeed, using the distance vectors above we infer that the minimum distance (or shortest path length) between John and 'Maria B' is 2 and that the minimum distance between John and 'Maria C' is 3, which are both correct. However, we also find that the minimum distance between John and 'Maria A' is 2, instead of 1. In this case, the distance vectors allow to *approximate* the minimum distance. The higher the number of seeds, the better the approximation tends to be for a higher number of nodes.

Seeds-based Ranking

We can now use the seed distances vectors to design a ranking function. This comprises the online phase of our solution. After some experimentation and empirical evaluation, we came up with the following ranking function.

$$\begin{aligned} seeds_J(Maria) &= k_1 * number\_seeds\_dist\_1 + \\ &\quad k_2 * number\_seeds\_dist\_2 + \\ &\quad k_3 * number\_seeds\_dist\_3 + \\ &\quad k_4 * number\_seeds\_dist\_4 \end{aligned} \quad (6)$$

$$rank_J(Maria) = \frac{seeds_J(Maria)}{log(number\_seeds(Maria))}$$

[1]In Orkut, the average number of friends of a user is 90, but we use 100 to simplify our reasoning.

[2]The actual algorithm used to find the shortest path from the seeds to the graph nodes is a bit more sophisticated and is described later in this section.

where $rank_J(Maria)$ is the rank of the answer 'Maria' with regard to the query posed by John. The variables $number\_seeds\_dist\_i$ indicate the number of seeds whose sum of distances to both John and Maria is exactly i. For instance, in the case of 'Maria A', only seed S2 satisfies this condition, which means that $number\_seeds\_dist\_2$ is equal to 1 for 'Maria A'. The other variables of the type $number\_seeds\_dist$ are defined analogously. The variable $number\_seeds(Maria)$ indicates the number of seeds with a finite distance to Maria and is used as a normalization factor. This is our *Seeds-based Ranking* algorithm.

The constants k_1, k_2, k_3, and k_4 are weighting factors whose values were chosen empirically. For instance, in here the values for these constants are set to 10^6, 10^4, 10^2, and 1, respectively. These are the values used throughout this paper. Let us illustrate with an example.

Example 2 - Ranking of results based on the seed distances vectors in Example 1, for the query 'Maria'.

$$rank_J(Maria\ A) = \frac{10^6 * 0 + 10^4 * 1 + 10^2 * 2 + 1 * 0}{log 3} \quad (7)$$
$$= 9284.44$$
$$rank_J(Maria\ B) = \frac{10^6 * 0 + 10^4 * 1 + 10^2 * 0 + 1 * 0}{log 3}$$
$$= 9102.39$$
$$rank_J(Maria\ C) = \frac{10^6 * 0 + 10^4 * 0 + 10^2 * 2 + 1 * 0}{log 3}$$
$$= 182.05$$

Thus, our ranking function scores higher 'Maria A', then 'Maria B', and finally 'Maria C' which, in this case, is the right thing to do. Although more sophisticated ranking functions can be developed, we have empirically verified the effectiveness of the function described above, as discussed in Section 7.

5. ALGORITHM FOR COMPUTING SEEDS-BASED RANKS

In our experimentation, we observed that user distances higher than 4 are not of much use in a network like Orkut. This is because each user has a high number of friends on average, which implies that 3 hops later we are already reaching out to many people. That is, in practice we do not need to compute distances larger than 4. This is an important observation in the design of our algorithm for computing the seeds-based ranking.

Sparse Seed Distances Vectors

Let $d_J(Maria)$ be the distance between John and Maria. Let us restrict our reasoning to the cases in which the estimated distance is correct. Then, we have the following possible cases:

- $d_J(Maria) = 1 \Longrightarrow$ there is a seed S_i whose distances to John and Maria are either (0,1) or (1,0);
- $d_J(Maria) = 2 \Longrightarrow$ there is a seed S_i whose distances to John and Maria are either (0,2), (1,1), or (2,0);
- $d_J(Maria) = 3 \Longrightarrow$ there is a seed S_i whose distances to John and Maria are either (0,3), (1,2), (2,1), or (3,0);
- $d_J(Maria) = 4 \Longrightarrow$ there is a seed S_i whose distances to John and Maria are either (0,4), (1,3), (2,2), (3,1), or (4,0).

That is, if we consider only seed distances equal or smaller than 2, we capture most of the cases required to compute user distances up to 4. Also, it is important to notice that there are many seeds providing information about the distance between two nodes, so the fact that we discarded distances from one seed does not mean that there is no other seed providing similar information.

As a result of this reasoning, in here we consider only seed distances equal or smaller than 2. If a seed distance is higher than 3, we consider it to be infinite. With this approximation, our seed distances vectors illustrated above become as follows.

Example 2 - Approximate seed distances vectors for users in Figure 2.

$$D_{John} = [2, 1, 1] \quad (8)$$
$$D_{Maria\ A} = [1, 1, 2] \quad (9)$$
$$D_{Maria\ B} = [\infty, \infty, 1] \quad (10)$$
$$D_{Maria\ C} = [1, 2, \infty] \quad (11)$$

These approximate seed distances vectors are then applied to Equation 7 to compute the ranking.

Limiting seed distance to a maximum of 2 is a key design decision in our solution, derived from the particularities of our searching problem (basically, user distances larger than 4 have small impact in the quality of the results), and the fact that we have a very sparse graph. Because of this decision, a large number of seed distances is set to infinite and do not need to be stored in the seed distances vectors. As a result, these vectors become more sparse, which favors fast computation and reduces query processing time. More important, this is accomplished while producing high precision rankings, as discussed in Section 7. Further, by varying the number of seeds we can strike a direct balance between high precision and high query throughput.

Map-reduce Computation of Seed Distances

To generate distances from each user to all the seeds, instead of doing a standard graph search for every user, we used a Map-Reduce process [4] which iterates over all users several times, propagating distances from the seeds towards their friends. This algorithm is outlined in Figure 5, and assumes that the seeds have already received a distance tag with the distance set to zero.

In the map stage of this algorithm (lines 1-13), we take each seed distance already assigned to a user, and emit new distances to all his friends, but with the distances incremented by one. The friendship lists of all users are passed forward allowing the algorithm to run in a distributed fashion. Each cluster node keeps in memory only the friendship lists necessary for the current step.

In the reduce stage (lines 14-25), we simply combine the many seed distances generated for each user back into the main distances vector for that user.

We exploit the fact that the seed distance vectors are sparse, and store the distances as [seed id, distance] pairs, using 22 bits for the ids and 2 for the distances, resulting in a space consumption of just 3 bytes per seed distance.

```
1  Map(userid, friends[1..f], seed_distances[1..s])
2  begin
3    new_distances[1..s] ← −1
4    for i = 1 to s
5        if seed_distances[i] = phase_number
6          new_distances[i] ← seed_distances[i] + 1
7        fi
8    end
9    for i = 1 to f
10       emit (friends[i], ∅, new_distances)
11   end
12   emit (userid, friends, seed_distances)
13 end
14 Reduce(userid, friends[1..f], seed_distances[1..s][1..n])
15 begin
16   distances[1..s] ← ∞
17   for i = 1 to n
18       for j = 1 to s
19           if seed_distances[j][i] < distances[j]
20             distances[j] ← seed_distances[j][i]
21           fi
22       end
23   end
24   emit userid, friends, distances
25 end
```

Figure 3: Map-reduce algorithm for computing approximate seed distances.

Computing Seeds-based Ranks

Once the seed distances vectors have been computed, we can compute seeds-based ranks in straightforward fashion by applying Equation 7. First, we match user names to the query and retrieve those that partially match the query terms. Following, we generate the seeds-based ranks. This is what we call the *Seeds-based Ranking* algorithm.

6. EXPERIMENTAL SETUP

To compare our seeds based algorithm with the simpler solutions, we ran experiments in a cluster of 128 machines as illustrated in Figure 4. There is one additional machine, called the *mixer*, which is responsible for interfacing with the external world and coordinating the computations of the machines in the cluster. A new query is received by the mixer, which replicates it to each of the 128 machines in the cluster. Upon receiving a query, each machines searches its local repository of user profiles, retrieves those that match the query (even if partially), ranks them, and returns the top k answers to the mixer. Upon receiving all 128 partial answers, the mixer runs a merge-in-place and generates a final set of the top k ranked users.

Figure 4: Cluster of 128 machines: 2GHz processors, 1 Gbps Ethernet, 4GB of RAM, 40M/128 user profiles per machine.

We observe that the population of all users is equally partitioned among all machines. Thus, for a population of 40M (40 million) users, obtained from Orkut, each machine stores the profiles of 40M/128 users.

In terms of queries, we used 40,000 random queries to evaluate the performance of the solutions and 100 random queries to evaluate the precision (both the query terms and the user submitting the query were random). For the *Seeds-based Ranking* algorithm, we varied the number of seeds experimenting with the following values: a hundred thousand (100,000), five hundred thousand (500,000), one million (1,000,000), two million (2,000,000), and three million (3,000,000) seeds.

7. RESULTS

In this Section we present and analyze our experimental results. We evaluate the quality of the results, the performance of our ranking algorithms, and their space requirements.

7.1 Precision Results

Relevance measures could be conducted to evaluate the quality of our ranking function but, since most social networks users tend to interact with people they know personally, in this paper we argue that the social distance as reflected by Equation 1 brings a good approximation to what is perceived by the users as relevant. So we evaluate the quality of our ranking algorithms by comparing their results with those produced by the *On-the-fly Ranking* algorithm, which yields "perfect" results based on our notion of relevance. That is, we basically need to compare distinct rankings produced by our various algorithms.

Compare Rankings Precision (crP)

We argue that we cannot use traditional metrics for comparing rankings (such as *Kendall Tau*) simply because, given the high number of ties, very distinct rankings may be equally relevant according to our social distance metric.

An alternative is to use *Precision@10* figures (P@10), a very common metric used by IR systems. Although P@10 figures are very useful as they have a clear meaning within the IR community, they cannot be applied directly here because we do not have access to user relevance judgements (and trying to judge which particular friend a user is looking for when he types a query is difficult). That is, we need to define the set of documents to be considered as the *ideal answer set*. Since we want to compare the results produced by alternative ranking functions with those produced by the *On-the-fly Ranking* algorithm, what we call the ideal answer set is not a set of answers judged relevant by the users or by a set of specialists. Instead, the ideal answer set is the set of documents most likely to be relevant to the users, according to Equation 1.

One possible definition for the ideal answer set, in this context, is to adopt the first 10 documents returned by the *On-the-fly Ranking* algorithm. However, this approach has an important drawback, as we now discuss.

For a given user query, assume the results of the *On-the-fly Ranking* and of the *Seeds-based Ranking* algorithms are as shown in Table 1.

On-the-fly Ranking	Seeds-based Ranking
1. 'Maria A', dist=1	1. 'Maria A', dist=1
2. 'Maria B', dist=1	2. 'Maria B', dist=1
3. 'Maria C', dist=2	3. 'Maria C', dist=2
4. 'Maria D', dist=2	4. 'Maria D', dist=2
5. 'Maria E', dist=2	5. 'Maria E', dist=2
6. 'Maria F', dist=2	6. 'Maria F', dist=2
7. 'Maria G', dist=2	7. 'Maria G', dist=2
8. 'Maria H', dist=2	8. 'Maria H', dist=2
9. 'Maria I', dist=2	9. 'Maria I', dist=2
10. 'Maria J', dist=2	10. 'Maria K', dist=2

Table 1: Hypothetical rankings for two of our algorithms, with regard to a given user query.

We observe that the two rankings are exactly the same, including the distance values relative to the user who posed the query, except for the tenth answer which is 'Maria J' in one case and 'Maria K' in the other (even if both of them are at a distance 2 from the user who posed the query). If we take the top 10 results produced by the *On-the-fly Ranking* algorithm as the ideal answer set and compute the P@10 metric for the *Seeds-based Ranking* algorithm, we get a precision value of 90%. However, the rankings are equally likely to be relevant to the user (who posed the query) because 'Maria J' and 'Maria K' are both at distance 2. To remedy this, we change the definition of the *ideal answer set*, as follows.

Definition. *Let O_{top} be the set of top 10 users returned by the* On-the-fly Ranking *algorithm, with regard to a given query. Let $o_{10} \in O_{top}$ be the 10^{th} answer in the result set and let $D(o_{10})$ be the distance of o_{10} to the user who posed the query. Let A be the set of all answers returned by an alternative ranking function which we want to evaluate, with regard to the same query. Further, let $a_i \in A$ be the ith answer in the result set. For any a_i, let $D(a_i)$ be the distance between a_i and the user who posed the query. Define the subset $A_o \subset A$ as*

$$A_o = \{a_i \mid a_i \in A \wedge D(a_i) = D(o_{10})\}$$

Then, the ideal answer set *I is defined as*

$$I = O_{top} \cup A_o$$

This definition has the property that it accounts for answers not included in the set O_{top}, as long as these answers have friendship distances that match the distances of the answers in O_{top}.

We emphasize that the ideal answer set I, as defined above, does not include relevance judgements provided by humans. Instead, it is defined with the specific purpose of comparing the results of alternative ranking algorithms with those produced by the *On-the-fly Ranking* algorithm. Thus, when we compute precision figures relative to I, we are not computing precision as defined by the IR research community. To make this point clear, we say that we compute a *compare-rankings* precision, as follows.

Definition. *In the context of our social networks, we evaluate the results produced by a ranking algorithm by computing precision figures with regard to an ideal answer set given by $I = O_{top} \cup A_o$, as defined previously. Since the set I includes no relevance judgements, we call our metric* compare-rankings Precision *(also* cr-precision*). Further, crP@10 refers to the compare-rankings precision at the 10^{th} position of the ranking.*

Unless explicitly stated otherwise, all precision figures in this paper are really cr-precision figures. This is so, even when we use the terminology precision simply, which we do in a few occasions to facilitate the flow of the text.

A First Set of Precision Results

Using our definition of crP@10 above, we compared the results produced by the *Seeds-based Ranking* and the *Co-friends Ranking* algorithms with those produced by the *On-the-fly Ranking* algorithm (which yields a crP@10 equal to 100%, by definition). The results are shown in Table 2.

Algorithm	# seeds	Avg. crP@10 (%)
Seeds-based Ranking	100,000	71.48
	500,000	81.98
	1,000,000	86.53
	2,000,000	90.50
	3,000,000	92.08
Co-friends Ranking		87.02

Table 2: Average cr-precision @ 10 for the *Seeds-based Ranking* and the *Co-friends Ranking* algorithms.

Regarding the *Seeds-based Ranking* algorithm, we observe that average cr-precision starts at 71.48% when we use a hundred thousand seeds and goes up as we increase the number of seeds used. With two million seeds, average cr-precision reaches the 90% range, which is excelent. Regarding the *Co-friends Ranking* algorithm, we observe that it produces a pretty good average precision of 87.02%. These results clearly show that (a) high precision can be attained using ranking functions that require much less computational resources and (b) the Seeds-based Ranking is highly competitive in terms of precision.

Generalized cr-Precision

While the results we observed are quite good, they hide a fundamental issue that is frequently overlooked when using standard precision figures to evaluate IR systems. Suppose that instead of the ranking showed in Table 1, the *Seeds-based Ranking* algorithm produced the following results:

Seeds-based Ranking
1. 'Maria K', dist=2
2. 'Maria B', dist=1
3. 'Maria C', dist=2
4. 'Maria D', dist=2
5. 'Maria E', dist=2
6. 'Maria F', dist=2
7. 'Maria G', dist=2
8. 'Maria H', dist=2
9. 'Maria I', dist=2
10. 'Maria A', dist=1

Table 3: New hypothetical results produced by the Seeds-based Ranking algorithm, for a given user query.

From the point of view of the user, these results are worse than the results in Table 1 (at least according to our established social distance criteria), but present the same crP@10. To try to address this problem, in addition to standard crP@10 metric, we also present our results based on *generalized precision*, as proposed by Kekalainen et al [9]. In the context of generalized precision, the relevance decision

is not binary. Instead, weights are assigned to the answers (we refer the reader to the original paper for details), as follows.

Definition. *Generalized compare-rankings precision (gcrP) is given by*

$$gcrP(O_r, A_r) = \sum_i weight(a_i)/weight(o_i) \qquad (12)$$

where O_r is the On-the-fly Ranking, A_r is the alternative ranking we want to evaluate, $o_i \in O_r$ and $a_i \in A_r$ are the i^{th} answers in the respective rankings. Also let $D(o_i)$ and $D(a_i)$ be the friendship distances from o_i and a_i, respectively, to the user who posed the query. For the social networks of today, $D(o_i) \leq 5$ and $D(a_i) \leq 5$. We can then define

$$\begin{aligned} weight(a_i) &= 5 - D(a_i) \qquad (13) \\ weight(o_i) &= 5 - D(o_i) \end{aligned}$$

In the above, we assigned trivial weights 1, 2, 3, 4, 5 to the friendship distances 5, 4, 3, 2, 1, respectively. Fine tuning these weights is a important step, which is beyond the scope of this paper and is left as future work.

We present results in terms of generalized cr-precision gcrP@1, gcrP@5, and gcrP@10 figures to capture the semantics embodied in the relative ordering of answers in the result sets.

Generalized cr-Precision Results

Generalized cr-precision gcrP@1, gcrP@5, and gcrP@10 figures are shown in Table 4.

Algorithms	# seeds	gcrP@1	gcrP@5	gcrP@10
Seeds-based Ranking	100,000	60.03	57.55	63.37
	500,000	72.88	71.71	74.26
	1,000,000	78.87	76.65	78.75
	2,000,000	85.21	83.30	83.36
	3,000,000	87.44	84.12	85.07
Co-friends Ranking		81.02	82.23	80.96

Table 4: Average generalized cr-precision @ 1, 5 and 10 for the *Seeds-based Ranking* and the *Co-friends Ranking* algorithms.

For the *Seeds-based Ranking* algorithm, average cr-precision figures dropped now to the 60% when a hundred thousand seeds are used. When two million seeds are used, average cr-precision lies in the 83-85% range. For the *Co-friends Ranking* algorithm, average precision figures are now in the 80-82% range. Even if the quality of the results is now lower, it is still the case that (a) high precision is attained by ranking algorithms that require less computational resources and (b) the *Seeds-based Ranking* algorithm produces the best results, when two million seeds or more are used.

7.2 Performance results

We now compare the performance results of our proposed ranking algorithms.

Average Query Execution Time

To evaluate performance, we measured the average execution time per query for 40,000 random queries. Table 5 displays the results. Note that the times presented below are only related to the overhead generated by the ranking of the results and do not include the time to process the text matching algorithms and to retrieve the results to be ranked, which is equal for all solutions.

Algorithm	# seeds	Avg. query time (ms)
Seeds-based Ranking	100,000	1.15
	500,000	2.38
	1,000,000	2.84
	2,000,000	4.89
	3,000,000	6.77
Co-friends Ranking		202
On-the-fly Ranking		2,018

Table 5: Average query execution time over 40,000 random queries.

For the *Seeds-based Ranking* algorithm, we notice that average execution time per query, for the cluster we experimented with, is 1.15ms when we use 100,000 seeds. Increasing the number of seeds 5-fold (to 500,000 seeds) doubles the average execution time per query (2.38ms). Further increasing the number of seeds to 2,000,000 implies in quadrupling the average execution time per query (4.89ms).

For the *Co-friends Ranking* and the *On-the-fly Ranking* algorithms, average query execution times are more than two and three orders of magnitude higher, respectively, than those of the *Seeds-based Ranking* algorithm using a hundred thousand seeds. They are also more than 40 and 400 times higher, respectively, than the average query execution time for the *Seeds-based Ranking* algorithm using two million seeds (which always yields higher precision results).

Seed Distances Computation Time

Also of interest are the times related to the offline computation of the seeds distance vectors. While there is no need for an extremely fast procedure, as it is carried out offline, this still needs to be fast enough to be usable in a live Web system (i.e., it cannot take days). Table 6 presents the execution times for the offline computation of seed distances.

# seeds	Time (s)
100,000	205
500,000	367
1,000,000	517
2,000,000	725
3,000,000	1068

Table 6: Execution times for the offline calculation of seed distances.

We observe that, even in the case of two million seeds, all seed distances can be computed in basically 12 minutes (725 seconds), which is very fast for a graph of the size of Orkut.

7.3 Space Requirements

One of the biggest advantages of the *Seeds-based Ranking* algorithm is that it requires much less space, as we now discuss.

Seeds-based Ranking Space Requirements

The seed distance vectors are actually very sparse, due to the fact that we limited seed distances to the maximum value of 2 (which works for our application and still allows computing user distances that go up to 4). Each seed distance is stored as 3 bytes, the first 22 bits for storing the seed id, the last 2 bits for storing the seed distance. Also, the computation can be easily distributed as the seed distances vectors can be partitioned exactly as done for the user nodes, since there is exactly one seed distance vector associated with each user node.

Table 7 shows the total space needed by the seed distances vectors as well as the average number of seeds per user at a distance 2 or smaller.

# seeds	Avg. # seeds	Space (GBytes)
100,000	53.28	4.35
500,000	264.97	21.20
1,000,000	532.50	42.49
2,000,000	1,063.83	84.79
3,000,000	1,597.21	127.25

Table 7: Space requirements for the *Seeds-based Ranking* algorithm for different numbers of seeds.

On-the-fly Ranking Space Requirements

The *On-the-fly Ranking* algorithm requires space only for storing the friendship relations graph. This graph takes roughly 16GB of space. However, this solution demands that all machines in the cluster have their own copy of the graph, as the minimal path computation can not be easily distributed among the machines. As a result, in practice, the space requirements are much higher, since we will need N copies of the graph, where N is the number of machines in the cluster. In our case, $N = 128$. Thus, the space required by the *On-the-fly Ranking* algorithm is actually 128 * 16GB = 2,048GB.

Co-friends Ranking Space Requirements

Regarding the *Co-friends Ranking* algorithm, the rough space requirements are as follows. Each user in Orkut has 90 friends on average. Assume that a user id can be represented by a 4 bytes integer. We need to store the list of friend-of-friends for each of the 40 million users. Thus, the space requirements for this algorithm are roughly 40M * 90 * 90 * 4 = 1,260GB.

Analysis of the Space Requirements

As one can see, the space requirements of the *On-the-fly* and *Co-friends* ranking algorithms are quite high. While the auxiliary structures for the *Seeds-based Ranking* algorithm can easily fit in the memory available in our cluster (128 * 4GB = 952GB), this is not true for the case of the *On-the-fly* and *Co-friends* ranking algorithms. This is of course one key advantage of our proposed solution.

Even though, to make the performance evaluations fair, we used the following strategy to be able to store the auxiliary structures of all the solutions in memory. For both the *On-the-fly Ranking* and the *Co-friends Ranking* algorithms, we created an additional warm-up step prior to carrying out our experiments. This step executes all queries that will be evaluated and generate, for each algorithm, a new auxiliary structure that contains only the relevant parts of the original structure to each query in our evaluation set. This new structure, which provides distance information just for the queries in our evaluation set, is much smaller and fits in the available memory. For instance, if the query being evaluated contains the users 'Maria A', 'Maria B' and 'Maria N' as results and 'John' is the user submitting the query, in the *On-the-fly Ranking* algorithm, we will only store the parts of the graph that are required to perform the minimal path algorithm between 'John' and the three Maria's. Analogously, only three friend-of-friends lists would need to be stored for the *Co-friends Ranking* algorithm.

All time related figures reported here were produced with all the data stored in main memory. For this, we used the warm-up procedure above, which cannot be used in production because queries are not known in advance. The *Seeds-based Ranking* algorithm does not need the warm-up, since all its data fits in memory, as previouly discussed.

7.4 Analysis of Our Results

Algorithm	# seeds	cr-precision	time (ms)	speedup
Seeds-based Ranking	100,000	71.48	1.15	1,755
	500,000	81.98	2.38	848
	1,000,000	86.53	2.84	711
	2,000,000	90.50	4.89	413
	3,000,000	92.08	6.77	298
Co-friends Ranking		87.02	202	10
On-the-fly Ranking		100.00	2,018	1

Table 8: Average cr-precision figures and average query execution times, reproduced from Table 2 and Table 5. Speedup is computed as the ratio between the average query execution time for the On-the-fly Ranking algorithm and the average query execution time for the ranking algorithm under evaluation.

Table 8 combines the results of Table 2 and Table 5, for convenience. For the *Seeds-based Ranking* algorithm, we observe that the adoption of 100,000 seeds leads to a loss in precision of roughly 29% but yields a speedup of 1,755, which is 175 times larger than the speedup produced by the *Co-friends Ranking* algorithm. In this case, the number of seeds used is only 0.25% of the total number of user nodes.

By increasing the number of seeds to 500,000, we reduce the loss in precision to roughly 18% at the expense of doubling the average execution time and reducing the speedup to 848. It is still a very good speedup, and the precision is now above 80%. Further increasing the number of seeds to 2 million (5% of the population of users), reduces the loss in precision to 10% at the expense of reducing the speedup to a factor of 413, which is still 41 times larger than the speedup provided by the *Co-friends Ranking* algorithm.

Our results indicate that great speedup and space savings can be obtained, while maintaining precision in the range of 70-90%, using a number of seeds that varies from 0.25-5% of the total population of users. These results indicate that

our algorithm is useful for supporting search ranking in very large social networks.

8. RELATED WORK

In this section we discuss separatelly works related to the contributions of this paper. First, we revisit the literature about shortest path calculation. After that, we discuss search algorithms for the web in general and social networks in particular.

8.1 Shortest Path Computation

The single source shortest path problem in graphs, called SSSP, has been extensively studied in the last decade. The classical Dijkstra [5] algorithm is taught in undergraduate computer sciences courses. Nonetheless, researchers are still challenged by the problem, in the presence of increasing graph sizes and the need of fast exact or approximate solutions for the problem.

There are many speed-up techniques for (repeatedly) calculating single-source single-target shortest paths in large, sparse graphs. A description of these techniques, and proposals on how to combine them can be seen the work of Holzer et. al. [8]. The idea of using seeds, or *landmarks*, to improve shortest path algorithms is also not new. Recently, Goldberg and Harrelson combined the technique with the A* algorithm, creating a fast and elegant method for calculating shortest paths [7].

Algorithms for the all pairs shortest path or APSP problem, where you find shortest paths between all vertices in a graph, also have been improving steadily. The initial breakthroughs were due to the use of fast matrix multiplication techniques [13]. Others have also proposed approximate algorithms for the problem [6]. Recently, Chan improved the best time bounds known for APSP in sparse graphs [3].

A survey about the current algorithms for both the SSSP and APSP problems, in different scenarios was conducted by Zwick [16].

All the improvements for the SSSP proposed recently could be applied to the second algorithm we presented, called *On-the-fly Ranking*. There are a few challenges, such as storing the whole graph in memory or using on-disk versions of the algorithms. However, for the latency needs of web search, anything that can not be calculated in constant time is prohibitive.

If we used APSP algorithms, as proposed in our first algorithm, *Pre-compute all distances*, querying time would not be a problem, since all the work is done on the offline computation of the shortest paths. Unfortunatelly, storing the results of an APSP calculation is not feasible, as discussed in Section 3. Therefore none of the APSP algorithms available in the literature can be directly used in our problem.

A third similar problem, the creation of *approximate distance oracles*, was defined Thorup and Zwick in [15]. In his graph distance survey [16] Zwick states the problem as "given a graph $G = (V, E)$, we would like to *preproccess* it so that subsequent *shortest path queries* could be answered quickly". This is exactly the requirements for our application in Orkut search, and any distance oracles could be directly applied in the architecture we proposed. Actually, our seeds based algorithm is a variation of the landmark algorithm shown in [11].

However, we have exploited two properties of our problem to improve our own algorithm. The first unique property of our problem is that we have a sparse and we are not interested in distances beyond 4. This allow us to store seeds distances with a sparse structure, resulting in a much higher number of seeds than what we could use otherwise. The second property we exploit is that we are only interested in relative ordering, and not in the actual distance between nodes. Because of this, we can use the distance of the seeds directly in our ranking algorithm, without needing an intermediate normalization step, such as the use of the Manhanttan Distance in [11].

8.2 Social Networks Search Ranking

Search is one of the most active research areas nowadays and there is a large range of literature available on the topic [1]. In the more specific area of web search, recent breakthrough developments have been observed [2]. However, the authors of this work are unaware of any existing publications discussing social network specific search problems.

It is important to notice that existing social networks already have powerful search mechanisms. For example, in Orkut, users can add several restrictions to their queries, such as hair color, age or geographical locations. Traditional text search techniques such as the vectorial model are also already used on these search mechanisms. Other signals available on the web, such as pagerank for example, can easily be brought to social network search. However, they bring little help when it comes to users searching for their friends or people in their real life social network.

As we noticed before, searching for friends is the use case of 50% of the Orkut query stream. And our claim that users are less interested in other users far in the graph is corroborated by a recent study of the *Facebook* social network [10].

As far as the knowledge of the authors go, this is the first work that addresses the problem of how to add personalized graph distance as a signal to a search engine ranking architecture.

9. CONCLUSIONS

Social Networks are a new and important trend in the Web, they came to stay. Given their very particular characteristics, they present new and exciting challenges to CS researchers in a number of areas that range from scalability problems to user behavior detection.

In this work, we have focused on the problem of how to improve the search experience of the users. We noticed that 50% of the queries in the Orkut social network are about names of other users, with an average of 1.8 terms per query. We argue that using only standard search techniques, any

ranking of the results satisfying the query would look like a random permutation of unknown persons to the user doing the query.

Exploiting the social network graph, the shortest distance between the user doing the query and the users in the results, we could add a strong signal to our ranking function. We have shown how to add this new signal in a distributed architecture, and how the strategy we have developed for that outperforms naive approaches, both in terms of precision and performance. The naive solutions are 40 to 400 times slower than our solution, and actually not practical for production deployment.

The result is that we moved from a text based ranking, almost meaningless to users, to a ranking where one can easily recognize the people in the search results. This is the first work that makes use of the friendship graph in a big social network to improve the search experience.

There are still many improvements that can be done to our work. We are investigating better strategies for selecting the seeds and more sophisticated ranking functions.

10. ACKNOWLEDGEMENTS

The comments and suggestions by Nissan Hajaj set us on the right track early on, we are greatly in debt to him. Big thanks also to the Orkut team at Google, which has done a great job supporting over 40 million users.

11. REFERENCES

[1] R. A. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[2] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. In *Proceedings of the seventh international conference on World Wide Web*, pages 107–117, Amsterdam, The Netherlands, The Netherlands, 1998.

[3] T. M. Chan. All-pairs shortest paths for unweighted undirected graphs in o(mn) time. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithms*, pages 514–523, New York, NY, USA, 2006.

[4] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In *Sixth Symposium on Operating System Design and Implementation*, pages 137–150, 2004.

[5] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

[6] D. Dor, S. Halperin, and U. Zwick. All-pairs almost shortest paths. *SIAM Journal on Computing*, 29(5):1740–1759, 2000.

[7] A. V. Goldberg and C. Harrelson. Computing the shortest path: A search meets graph theory. In *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 156–165, Philadelphia, PA, USA, 2005.

[8] M. Holzer, F. Schulz, D. Wagner, and T. Willhalm. Combining speed-up techniques for shortest-path computations. *Journal of Experimental Algorithmics*, 10, 2005.

[9] J. Kekäläinen and K. Järvelin. Using graded relevance assessments in ir evaluation. *Journal of the American Society for Information Science and Technology*, 53(13):1120–1129, 2002.

[10] C. Lampe, N. Ellison, and C. Steinfield. A face(book) in the crowd: social searching vs. social browsing. In *Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work*, pages 167–170, New York, NY, USA, 2006.

[11] M. J. Rattigan, M. Maier, and D. Jensen. Using structure indices for efficient approximation of network properties. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 357–366, New York, NY, USA, 2006.

[12] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Englewood Cliffs, NJ, 2nd edition edition, 2003.

[13] R. Seidel. On the all-pairs-shortest-path problem. In *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, pages 745–749, 1992.

[14] E. Spertus, M. Sahami, and O. Buyukkokten. Evaluating similarity measures: a large-scale study in the orkut social network. In *Proceeding of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 678–684, New York, NY, USA, 2005.

[15] M. Thorup and U. Zwick. Approximate distance oracles. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 183–192, New York, NY, USA, 2001.

[16] U. Zwick. Exact and approximate distances in graphs - a survey. In *Proceedings of the 9th Annual European Symposium on Algorithms*, pages 33–48, London, UK, 2001.