The Anatomy of a Large-Scale *Social* Search Engine

Damon Horowitz
Aardvark
damon@aardvarkteam.com

Sepandar D. Kamvar
Stanford University
sdkamvar@stanford.edu

ABSTRACT

We present Aardvark, a social search engine. With Aardvark, users ask a question, either by instant message, email, web input, text message, or voice. Aardvark then routes the question to the person in the user's extended social network most likely to be able to answer that question. As compared to a traditional web search engine, where the challenge lies in finding the right *document* to satisfy a user's information need, the challenge in a social search engine like Aardvark lies in finding the right *person* to satisfy a user's information need. Further, while trust in a traditional search engine is based on authority, in a social search engine like Aardvark, trust is based on intimacy. We describe how these considerations inform the architecture, algorithms, and user interface of Aardvark, and how they are reflected in the behavior of Aardvark users.

1. INTRODUCTION

1.1 The Library and the Village

Traditionally, the basic paradigm in information retrieval has been the library. Indeed, the field of IR has roots in the library sciences, and Google itself came out of the Stanford Digital Library project [18]. While this paradigm has clearly worked well in several contexts, it ignores another age-old model for knowledge acquisition, which we shall call "the village paradigm". In a village, knowledge dissemination is achieved socially — information is passed from person to person, and the retrieval task consists of finding the right person, rather than the right document, to answer your question.

The differences how people find information in a library versus a village suggest some useful principles for designing a social search engine. In a library, people use keywords to search, the knowledge base is created by a small number of content publishers before the questions are asked, and trust is based on authority. In a village, by contrast, people use natural language to ask questions, answers are generated in real-time by anyone in the community, and trust is based on intimacy. These properties have cascading effects — for example, real-time responses from socially proximal responders tend to elicit (and work well for) highly contextualized and subjective queries. For example, the query "Do you have any good babysitter recommendations in Palo Alto for my 6-year-old twins? I'm looking for somebody that won't let them watch TV." is better answered by a friend than the library. These differences in information retrieval paradigm require that a social search engine have very different architecture, algorithms, and user interfaces than a search engine based on the library paradigm.

The fact that the library and the village paradigms of knowledge acquisition complement one another nicely in the offline world suggests a broad opportunity on the web for social information retrieval.

WWW2010, April 26-30, 2010, Raleigh, North Carolina.

1.2 Aardvark

In this paper, we present Aardvark, a social search engine based on the village paradigm. We describe in detail the architecture, ranking algorithms, and user interfaces in Aardvark, and the design considerations that motivated them. We believe this to be useful to the research community for two reasons. First, the argument made in the original Anatomy paper [4] still holds true — since most search engine development is done in industry rather than academia, the research literature describing end-to-end search engine architecture is sparse. Second, the shift in paradigm opens up a number of interesting research questions in information retrieval, for example around expertise classification, implicit network construction, and conversation design.

Following the architecture description, we present a statistical analysis of usage patterns in Aardvark. We find that, as compared to traditional search, Aardvark queries tend to be long, highly contextualized and subjective — in short, they tend to be the types of queries that are not well-serviced by traditional search engines. We also find that the vast majority of questions get answered promptly and satisfactorily, and that users are surprisingly active, both in asking and answering.

Finally, we present example results from the current Aardvark system, and a comparative evaluation experiment. What we find is that Aardvark performs very well on queries that deal with opinion, advice, experience, or recommendations, while traditional corpus-based search engines remain a good choice for queries that are factual or navigational.

2. OVERVIEW

2.1 Main Components

The main components of Aardvark are:

1. *Crawler and Indexer.* To find and label resources that contain information — in this case, users, not documents (Sections 3.2 and 3.3).

2. *Query Analyzer.* To understand the user's information need (Section 3.4).

3. *Ranking Function.* To select the best resources to provide the information (Section 3.5).

4. *UI.* To present the information to the user in an accessible and interactive form (Section 3.6).

Most corpus-based search engines have similar key components with similar aims [4], but the means of achieving those aims are quite different.

Before discussing the anatomy of Aardvark in depth, it is useful to describe what happens behind the scenes when a new user joins Aardvark and when a user asks a question.

2.2 The Initiation of a User

When a new user first joins Aardvark, the Aardvark system performs a number of indexing steps in order to be able to direct the appropriate questions to her for answering.

Because questions in Aardvark are routed to the user's extended network, the first step involves indexing friendship and affiliation information. The data structure responsible for this is the *Social Graph*. Aardvark's aim is not to build a social network, but rather to allow people to make use of their existing social networks. As such, in the sign-up process, a new user has the option of connecting to a social network such as Facebook or LinkedIn, importing their contact lists from a webmail program, or manually inviting friends to join. Additionally, anybody whom the user invites to join Aardvark is appended to their Social Graph – and such invitations are a major source of new users. Finally, Aardvark users are connected through common "groups" which reflect real-world affiliations they have, such as the schools they have attended and the companies they have worked at; these groups can be imported automatically from social networks, or manually created by users. Aardvark indexes this information and stores it in the Social Graph, which is a fixed width ISAM index sorted by userId.

Simultaneously, Aardvark indexes the topics about which the new user has some level of knowledge or experience. This topical expertise can be garnered from several sources: a user can indicate topics in which he believes himself to have expertise; a user's friends can indicate which topics they trust the user's opinions about; a user can specify an existing structured profile page from which the *Topic Parser* parses additional topics; a user can specify an account on which they regularly post status updates (e.g., Twitter or Facebook), from which the *Topic Extractor* extracts topics (from unstructured text) in an ongoing basis (see Section 3.3 for more discussion); and finally, Aardvark observes the user's behavior on Aardvark, in answering (or electing not to answer) questions about particular topics.

The set of topics associated with a user is recorded in the *Forward Index*, which stores each userId, a scored list of topics, and a series of further scores about a user's behavior (e.g., responsiveness or answer quality). From the Forward Index, Aardvark constructs an *Inverted Index*. The Inverted Index stores each topicId and a scored list of userIds that have expertise in that topic. In addition to topics, the Inverted Index stores scored lists of userIds for features like answer quality and response time.

Once the Inverted Index and Social Graph for a user are created, the user is now active on the system and ready to ask her first question.

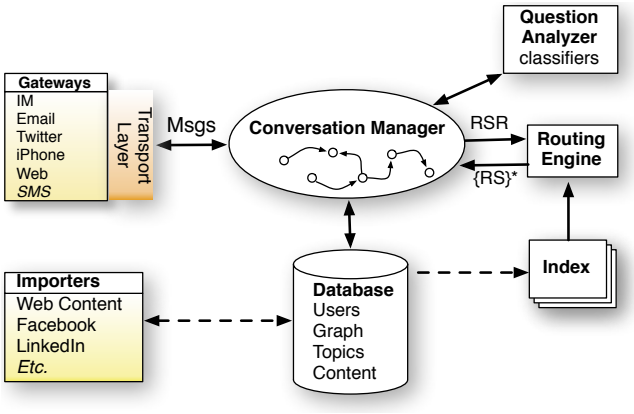

Figure 1: Schematic of the architecture of Aardvark

2.3 The Life of a Query

A user begins by asking a question, most commonly through instant message or text message The question gets sent from the input device to the *Transport Layer*, where it is normalized to a *Message* data structure, and sent to the *Conversation Manager*. Once the Conversation Manager determines that the message is a question, it sends the question to the *Question Analyzer* to determine the appropriate topics for the question. The Conversation Manager informs the asker which primary topic was determined for the question, and gives the asker the opportunity to edit it. It simultaneously issues a *Routing Suggestion Request* to the *Routing Engine*. The routing engine plays a role analogous to the ranking function in a corpus-based search engine. It accesses the *Inverted Index* and *Social Graph* for a list of candidate answerers, and ranks them to reflect how well it believes they can answer the question, and how good of a match they are for the asker. The Routing Engine returns a ranked list of *Routing Suggestions* to the Conversation Manager, which then contacts the potential answerers — one by one, or a few at a time, depending upon a *Routing Policy* — and asks them if they would like to answer the question, until a satisfactory answer is found. The Conversation Manager then forwards this answer along to the asker, and allows the asker and answerer to exchange followup messages.

3. ANATOMY

3.1 The Model

The core of Aardvark is a statistical model for routing questions to potential answerers. We use a network variant of what has been called an *aspect model* [12], that has two primary features. First, it associates an unobserved class variable $t \in T$ with each observation (i.e., the successful answer of question q by user u_i). In other words, the probability $p(u_i|q)$ that user i will successfully answer question q depends on whether q is about the topics t in which u_i has expertise[1]:

$$p(u_i|q) = \sum_{t \in T} p(u_i|t)p(t|q) \quad (1)$$

[1]Equation 1 is a simplification of what Aardvark actually uses to match queries to answerers, but we present it this way for clarity and conciseness.

The second main feature of the model is that it defines a query-independent probability of success for each potential asker/answerer pair (u_i, u_j), based upon their degree of social connectedness and profile similarity. In other words, we define a probability $p(u_i|u_j)$ that user u_i will deliver a satisfying answer to user u_j, regardless of the question.

We then define the scoring function $s(u_i, u_j, q)$ as the composition of the two probabilities.

$$s(u_i, u_j, q) = p(u_i|u_j) \cdot p(u_i|q) = p(u_i|u_j) \sum_{t \in T} p(u_i|t)p(t|q) \quad (2)$$

Our goal in the ranking problem is: given a question q from user u_j, return a ranked list of users $u_i \in U$ that maximizes $s(u_i, u_j, q)$.

Note that the scoring function is composed of a query-dependent *relevance score* $p(u_i|q)$ and a query-independent *quality score* $p(u_i|u_j)$. This bears similarity to the ranking functions of traditional corpus-based search engines such as Google [4]. The difference is that unlike quality scores like PageRank [18], Aardvark's quality score aims to measure intimacy rather than authority. And unlike the relevance scores in corpus-based search engines, Aardvark's relevance score aims to measure a user's *potential* to answer a query, rather than a document's existing capability to answer a query.

Computationally, this scoring function has a number of advantages. It allows real-time routing because it pushes much of the computation offline. The only component probability that needs to be computed at query time is $p(t|q)$. Computing $p(t|q)$ is equivalent to assigning topics to a question — in Aardvark we do this by running a probabilistic classifier on the question at query time (see Section 3.4). The distribution $p(u_i|t)$ assigns users to topics, and the distribution $p(u_i|u_j)$ defines the Aardvark Social Graph. Both of these are computed by the Indexer at signup time, and then updated continuously in the background as users answer questions and get feedback (see Section 3.3). The component multiplications and sorting are also done at query time, but these are easily parallelizable, as the index is sharded by user.

3.2 Social Crawling

A comprehensive knowledge base is important for search engines as query distributions tend to have a long tail [13]. In corpus-based search engines, this is achieved by large-scale crawlers and thoughtful crawl policies. In Aardvark, the knowledge base consists of people rather than documents, so the methods for acquiring and expanding a comprehensive knowledge base are quite different.

With Aardvark, the more active users there are, the more potential answerers there are, and therefore the more comprehensive the coverage. More importantly, because Aardvark looks for answerers primarily within a user's extended social network, the denser the network, the larger the effective knowledge base.

This suggests that the strategy for increasing the knowledge base of Aardvark crucially involves creating a good experience for users so that they remain active and are inclined to invite their friends. An extended discussion of this is outside of the scope of this paper; we mention it here only to emphasize the difference in the nature of "crawling" in social search versus traditional search.

Given a set of active users on Aardvark, the effective breadth of the Aardvark knowledge base depends upon designing interfaces and algorithms that can collect and learn an extended topic list for each user over time, as discussed in the next section.

3.3 Indexing People

The central technical challenge in Aardvark is selecting the right user to answer a given question from another user. In order to do this, the two main things Aardvark needs to learn about each user u_i are: (1) the topics t he might be able to answer questions about $p_{smoothed}(t|u_i)$; (2) the users u_j to whom he is connected $p(u_i|u_j)$.

Topics. Aardvark computes the distribution $p(t|u_i)$ of topics known by user u_i from the following sources of information:

- Users are prompted to provide at least three topics which they believe they have expertise about.
- Friends of a user (and the person who invited a user) are encouraged to provide a few topics that they trust the user's opinion about.
- Aardvark parses out topics from users' existing online profiles (e.g., Facebook profile pages, if provided). For such pages with a known structure, a simple Topic Parsing algorithm uses regular expressions which were manually devised for specific fields in the pages, based upon their performance on test data.
- Aardvark automatically extracts topics from unstructured text on users' existing online homepages or blogs if provided. For unstructured text, a linear SVM identifies the general subject area of the text, while an ad-hoc named entity extractor is run to extract more specific topics, scaled by a variant tf-idf score.
- Aardvark automatically extracts topics from users' status message updates (e.g., Twitter messages, Facebook news feed items, IM status messages, etc.) and from the messages they send to other users on Aardvark.

The motivation for using these latter sources of profile topic information is a simple one: if you want to be able to predict what kind of content a user will generate (i.e., $p(t|u_i)$), first examine the content they have generated in the past. In this spirit, Aardvark uses web content not as a source of existing answers about a topic, but rather, as an indicator of the topics about which a user is likely able to give new answers on demand.

In essence, this involves modeling a user as a content-generator, with probabilities indicating the likelihood she will likely respond to questions about given topics. Each topic in a user profile has an associated score, depending upon the confidence appropriate to the source of the topic. In addition, Aardvark learns over time which topics *not* to send a user questions about by keeping track of cases when the user: (1) explicitly "mutes" a topic; (2) declines to answer questions about a topic when given the opportunity; (3) receives negative feedback on his answer about the topic from another user.

Periodically, Aardvark will run a topic strengthening algorithm, the essential idea of which is: if a user has expertise in a topic and most of his friends also have some expertise in that topic, we have more confidence in that user's

level of expertise than if he were alone in his group with knowledge in that area. Mathematically, for some user u_i, his group of friends U, and some topic t, if $p(t|u_i) \neq 0$, then $s(t|u_i) = p(t|u_i) + \gamma \sum_{u \in U} p(t|u)$, where γ is a small constant. The s values are then renormalized to form probabilities.

Aardvark then runs two smoothing algorithms the purpose of which are to record the possibility that the user may be able to answer questions about additional topics not explicitly recorded in her profile. The first uses basic collaborative filtering techniques on topics (i.e., based on users with similar topics), the second uses semantic similarity[2].

Once all of these bootstrap, extraction, and smoothing methods are applied, we have a list of topics and scores for a given user. Normalizing these topic scores so that $\sum_{t \in T} p(t|u_i) = 1$, we have a probability distribution for topics known by user u_i. Using Bayes' Law, we compute for each topic and user:

$$p(u_i|t) = \frac{p(t|u_i)p(u_i)}{p(t)}, \tag{3}$$

using a uniform distribution for $p(u_i)$ and observed topic frequencies for $p(t)$. Aardvark collects these probabilities $p(u_i|t)$ indexed by topic into the Inverted Index, which allows for easy lookup when a question comes in.

Connections. Aardvark computes the connectedness between users $p(u_i|u_j)$ in a number of ways. While social proximity is very important here, we also take into account similarities in demographics and behavior. The factors considered here include:

- Social connection (common friends and affiliations)
- Demographic similarity
- Profile similarity (e.g., common favorite movies)
- Vocabulary match (e.g., IM shortcuts)
- Chattiness match (frequency of follow-up messages)
- Verbosity match (the average length of messages)
- Politeness match (e.g., use of "Thanks!")
- Speed match (responsiveness to other users)

Connection strengths between people are computed using a weighted cosine similarity over this feature set, normalized so that $\sum_{u_i \in U} p(u_i|u_j) = 1$, and stored in the Social Graph for quick access at query time.

Both the distributions $p(u_i|u_j)$ in the Social Graph and $p(t|u_i)$ in the Inverted Index are continuously updated as users interact with one another on Aardvark.

3.4 Analyzing Questions

The purpose of the Question Analyzer is to determine a scored list of topics $p(t|q)$ for each question q representing the semantic subject matter of the question. This is the only probability distribution in equation 2 that is computed at query time.

[2]In both the Person Indexing and the Question Analysis components, "semantic similarity" is computed by using an approximation of distributional similarity computed over Wikipedia and other corpora; this serves as a proxy measure of the topics' semantic relatedness.

It is important to note that in a social search system, the requirement for a Question Analyzer is only to be able to understand the query sufficiently for routing it to a likely answerer. This is a considerably simpler task than the challenge facing an ideal web search engine, which must attempt to determine exactly what piece of information the user is seeking (i.e., given that the searcher must translate her information need into search keywords), and to evaluate whether a given web page contains that piece of information. By contrast, in a social search system, it is the human answerer who has the responsibility for determining the relevance of an answer to a quesion — and that is a function which human intelligence is extremely well-suited to perform! The asker can express his information need in natural language, and the human answerer can simply use her natural understanding of the language of the question, of its tone of voice, sense of urgency, sophistication or formality, and so forth, to determine what information is suitable to include in a response. Thus, the role of the Question Analyzer in a social search system is simply to learn enough about the question that it may be sent to appropriately interested and knowledgeable human answerers.

As a first step, the following classifiers are run on each question: A *NonQuestionClassifier* determines if the input is not actually a question (e.g., is it a misdirected message, a sequence of keywords, etc.); if so, the user is asked to submit a new question. An *InappropriateQuestionClassifier* determines if the input is obscene, commercial spam, or otherwise inappropriate content for a public question-answering community; if so, the user is warned and asked to submit a new question. A *TrivialQuestionClassifier* determines if the input is a simple factual question which can be easily answered by existing common services (e.g., "What time is it now?", "What is the weather?", etc.); if so, the user is offered an automatically generated answer resulting from traditional web search. A *LocationSensitiveClassifier* determines if the input is a question which requires knowledge of a particular location, usually in addition to specific topical knowledge (e.g., "What's a great sushi restaurant in Austin, TX?"); if so, the relevant location is determined and passed along to the Routing Engine with the question.

Next, the list of topics relevant to a question is produced by merging the output of several distinct TopicMapper algorithms, each of which suggests its own scored list of topics:

- A *KeywordMatchTopicMapper* passes any terms in the question which are string matches with user profile topics through a classifier which is trained to determine whether a given match is likely to be semantically significant or misleading.[3]
- A *TaxonomyTopicMapper* classifies the question text into a taxonomy of roughly 3000 popular question topics, using an SVM trained on an annotated corpus of several millions questions.
- A *SalientTermTopicMapper* extracts salient phrases from the question — using a noun-phrase chunker and a tf-

[3]For example, if the string "camel wrestling" occurs in a question, it is likely to be semantically relevant to a user who has "camel wrestling" as a profile topic; whereas the string "running" is too ambiguous to use in this manner without further validation, since it might errantly route a question about "running a business" to a user who knows about fitness.

idf–based measure of importance — and finds semantically similar user topics.

- A *UserTagTopicMapper* takes any user "tags" provided by the asker (or by any would-be answerers), and maps these to semantically-similar user topics.[4]

At present, the output distributions of these classifiers are combined by weighted linear combination. It would be interesting future work to explore other means of combining heterogeneous classifiers, such as the maximum entropy model in [16].

The Aardvark TopicMapper algorithms are continuously evaluated by manual scoring on random samples of 1000 questions. The topics used for selecting candidate answerers, as well as a much larger list of possibly relevant topics, are assigned scores by two human judges, with a third judge adjudicating disagreements. For the current algorithms on the current sample of questions, this process yields overall scores of 89% precision and 84% recall of relevant topics. In other words, 9 out of 10 times, Aardvark will be able to route a question to someone with relevant topics in her profile; and Aardvark will identify 5 out of every 6 possibly relevant answerers for each question based upon their topics.

3.5 The Aardvark Ranking Algorithm

Ranking in Aardvark is done by the Routing Engine, which determines an ordered list of users (or "candidate answerers") who should be contacted to answer a question, given the asker of the question and the information about the question derived by the Question Analyzer. The core ranking function is described by equation 2; essentially, the Routing Engine can be seen as computing equation 2 for all candidate answerers, sorting, and doing some postprocessing.

The main factors that determine this ranking of users are Topic Expertise $p(u_i|q)$, Connectedness $p(u_i|u_j)$, and Availability:

Topic Expertise: First, the Routing Engine finds the subset of users who are semantic matches to the question: those users whose profile topics indicate expertise relevant to the topics which the question is about. Users whose profile topics are closer matches to the question's topics are given higher rank. For questions which are location-sensitive (as defined above), only users with matching locations in their profiles are considered.

Connectedness: Second, the Routing Engine scores each user according to the degree to which she herself — as a person, independently of her topical expertise — is a good "match" for the asker for this information query. The goal of this scoring is to optimize the degree to which the asker and the answerer feel kinship and trust, arising from their sense of connection and similarity, and meet each other's expectations for conversational behavior in the interaction.

Availability: Third, the Routing Engine prioritizes candidate answerers in such a way so as to optimize the chances that the present question will be answered, while also preserving the available set of answerers (i.e., the quantity of "answering resource" in the system) as much as possible by spreading out the answering load across the user base. This involves factors such as prioritizing users who are currently online (e.g., via IM presence data, iPhone usage, etc.), who are historically active at the present time-of-day, and who have not been contacted recently with a request to answer a question.

Given this ordered list of candidate answerers, the Routing Engine then filters out users who should not be contacted, according to Aardvark's guidelines for preserving a high-quality user experience. These filters operate largely as a set of rules: do not contact users who prefer to not be contacted at the present time of day; do not contact users who have recently been contacted as many times as their contact frequency settings permit; etc.

Since this is all done at query time, and the set of candidate answerers can potentially be very large, it is useful to note that this process is parallelizable. Each shard in the Index computes its own ranking for the users in that shard, and sends the top users to the Routing Engine. This is scalable as the user base grows, since as more users are added, more shards can be added.

The list of candidate answerers who survive this filtering process are returned to the Conversation Manager. The Conversation Manager then proceeds with opening channels to each of them, serially, inquiring whether they would like to answer the present question; and iterating until an answer is provided and returned to the asker.

3.6 User Interface

Since social search is modeled after the real-world process of asking questions to friends, the various user interfaces for Aardvark are built on top of the existing communication channels that people use to ask questions to their friends: IM, email, SMS, iPhone, Twitter, and Web-based messaging. Experiments were also done using actual voice input from phones, but this is not live in the current Aardvark production system.

In its simplest form, the user interface for asking a question on Aardvark is any kind of text input mechanism, along with a mechanism for displaying textual messages returned from Aardvark. (This kind of very lightweight interface is important for making the search service available anywhere, especially now that mobile device usage is ubiquitous across most of the globe.)

However, Aardvark is most powerful when used through

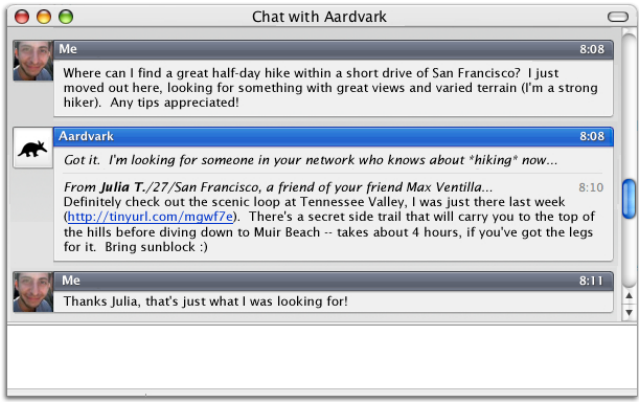

Figure 2: Screenshot of Aardvark over IM

[4]A general principle in the design of Aardvark is to use human intelligence wherever possible to improve the quality of the system. For the present task of Question Analysis, this involves giving both askers and answerers prompts and simple commands for telling Aardvark directly what the subject matter of a question is.

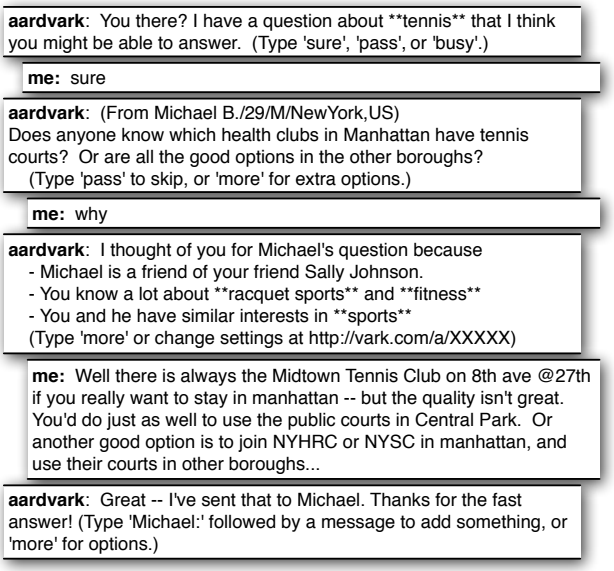

Figure 3: Example of Aardvark interacting with an answerer

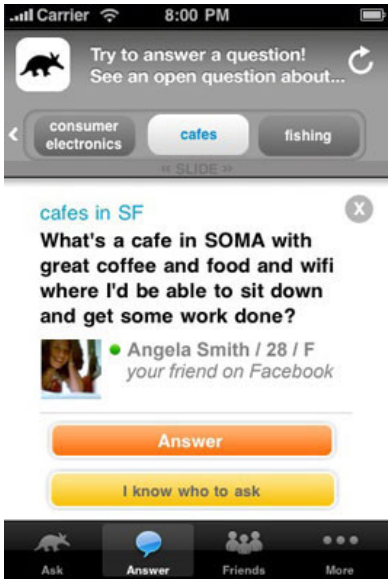

Figure 4: Screenshot of Aardvark Answering Tab on iPhone

a chat-like interface that enables ongoing conversational interaction. A private 1-to-1 conversation creates an intimacy which encourages both honesty and freedom within the constraints of real-world social norms. (By contrast, answering forums where there is a public audience can both inhibit potential answerers [17] or motivate public performance rather than authentic answering behavior [22].) Further, in a real-time conversation, it is possible for an answerer to request clarifying information from the asker about her question, or for the asker to follow-up with further reactions or inquiries to the answerer.

There are two main interaction flows available in Aardvark for answering a question. The primary flow involves Aardvark sending a user a message (over IM, email, etc.), asking if she would like to answer a question: for example, "You there? A friend from the Stanford group has a question about *search engine optimization* that I think you might be able to answer.". If the user responds affirmatively, Aardvark relays the question as well as the name of the questioner. The user may then simply type an answer the question, type in a friend's name or email address to refer it to someone else who might answer, or simply "pass" on this request.[5]

A key benefit of this interaction model is that the available set of potential answerers is not just whatever users happen to be visiting a bulletin board at the time a question is posted, but rather, the entire set of users that Aardvark has contact information for. Because this kind of "reaching out" to users has the potential to become an unwelcome interruption if it happens too frequently, Aardvark sends such requests for answers usually less than once a day to a given user (and users can easily change their contact settings, specifying prefered frequency and time-of-day for such requests). Further, users can ask Aardvark "why" they were selected for a particular question, and be given the option to easily change their profile if they do not want such questions in the future. This is very much like the real-world model of social information sharing: the person asking a question, or the intermediary in Aardvark's role, is careful not to impose too much upon a possible answerer. The ability to reach out to an extended network beyond a user's immediate friendships, without imposing too frequently on that network, provides a key differentiating experience from simply posting questions to one's Twitter or Facebook status message.

A secondary flow of answering questions is more similar to traditional bulletin-board style interactions: a user sends a message to Aardvark (e.g., "try") or visits the "Answering" tab of Aardvark website or iPhone application (Figure 4), and Aardvark shows the user a recent question from her network which has not yet been answered and is related to her profile topics. This mode involves the user initiating the exchange when she is in the mood to try to answer a question; as such, it has the benefit of an eager potential answerer – but as the only mode of answering it does not effectively tap into the full diversity of the user base (since most users do not initiate these episodes). This is an important point: while almost everyone is happy to answer questions (see Section 5) to help their friends or people they are connected to, not everyone goes out of their way to do so. This willingness to be helpful persists because when users do answer questions, they report that it is a very gratifying experience: they have been selected by Aardvark because of their expertise, they were able to help someone who had a need in the moment, and they are frequently thanked for their help by the asker.

In order to play the role of intermediary in an ongoing conversation, Aardvark must have some basic conversational intelligence in order to understand where to direct messages

[5]There is no shame in "passing" on a question, since nobody else knows that the question was sent to you. Similarly, there is no social cost to the user in asking a question, since you are not directly imposing on a friend or requesting a favor; rather, Aardvark plays the role of the intermediary who bears this social cost.

Figure 5: A random set of queries from Aardvark

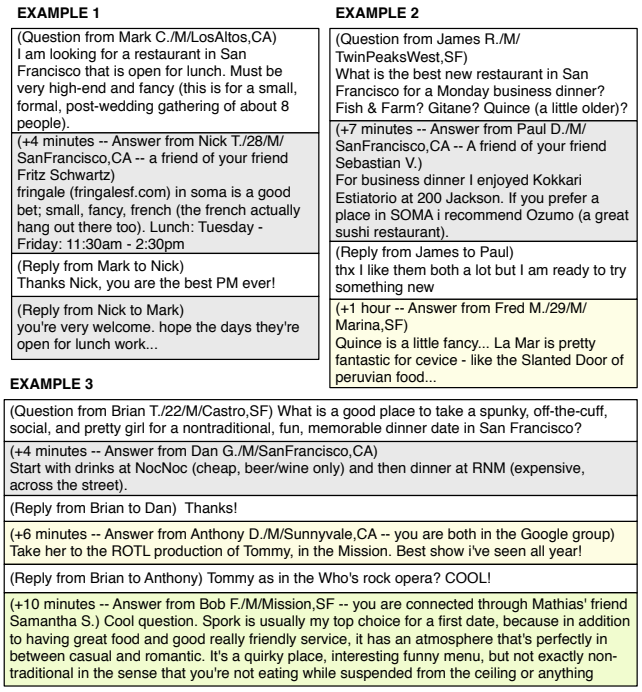

Figure 6: Three complete Aardvark interactions

from a user: is a given message a new question, a continuation of a previous question, an answer to an earlier question, or a command to Aardvark? The details of how the Conversation Manager manages these complications and disambiguates user messages are not essential so they are not elaborated here; but the basic approach is to use a state machine to model the discourse context.

In all of the interfaces, wrappers around the messages from another user include information about the user that can facilitate trust: the user's Real Name nametag, with their name, age, gender, and location; the social connection between you and the user (e.g., "Your friend on Facebook", "A friend of your friend Marshall Smith", "You are both in the Stanford group", etc.); a selection of topics the user has expertise in; and summary statistics of the user's activity on Aardvark (e.g., number of questions recently asked or answered).

Finally, it is important throughout all of the above interactions that Aardvark maintains a tone of voice which is friendly, polite, and appreciative. A social search engine depends upon the goodwill and interest of its users, so it is important to demonstrate the kind of (linguistic) behavior that can encourage these sentiments, in order to set a good example for users to adopt. Indeed, in user interviews, users often express their desire to have examples of how to speak or behave socially when using Aardvark; since it is a novel paradigm, users do not immediately realize that they can behave in the same ways they would in a comparable real-world situation of asking for help and offering assistance. All of the language that Aardvark uses is intended both to be a communication mechanism between Aardvark and the user and an example of how to interact with Aardvark.

Overall, a large body of research [8, 2, 7, 21] shows that when you provide a 1-1 communication channel, use real identities rather than pseudonyms, facilitate interactions between existing real-world relationships, and consistently provide examples of how to behave, users in an online community will behave in a manner that is far more authentic and helpful than pseudonymous multicasting environments with no moderators. The design of the Aardvark's UI has been carefully crafted around these principles.

4. EXAMPLES

In this section we take a qualitative look at user behavior on Aardvark. Figure 5 shows a random sample of questions asked on Aardvark in this period. Figure 6 takes a closer look at three questions sent to Aardvark during this period, all three of which were categorized by the Question Analyzer under the primary topic "restaurants in San Francisco".[6]

In Example 1, Aardvark opened 3 channels with candidate answerers, which yielded 1 answer. An interesting (and not uncommon) aspect of this example is that the asker and the answerer in fact were already acquaintances, though only listed as "friends-of-friends" in their online social graphs; and they had a quick back-and-forth chat through Aardvark.

In Example 2, Aardvark opened 4 channels with candidate answerers, which yielded two answers. For this question, the "referral" feature was useful: one of the candidate answerers was unable to answer, but referred the question along to a friend to answer; this enables Aardvark to tap into the knowledge of not just its current user base, but also the knowledge of everyone that the current users know. The asker wanted more choices after the first answer, and resubmitted the question to get another recommendation, which came from a user whose profile topics related to business and finance (in addition to dining).

In Example 3, Aardvark opened 10 channels with candidate answerers, yielding 3 answers. The first answer came from someone with only a distant social connection to the

[6]Names and affiliations have been changed to protect privacy.

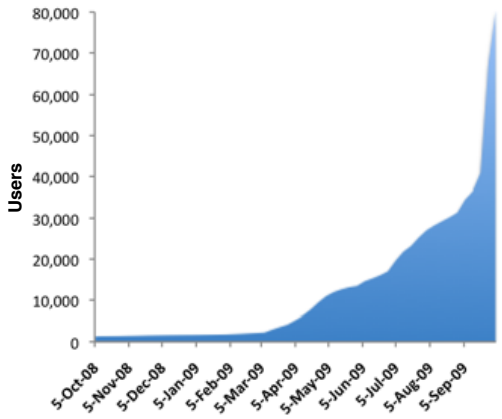

Figure 7: Aardvark user growth

asker; the second answer came from a coworker; and the third answer came from a friend-of-friend-of-friend. The third answer, which is the most detailed, came from a user who has topics in his profile related to both "restaurants" and "dating".

One of the most interesting features of Aardvark is that it allows askers to get answers that are hypercustomized to their information need. Very different restaurant recommendations are appropriate for a date with a spunky and spontaneous young woman, a post-wedding small formal family gathering, and a Monday evening business meeting — and human answerers are able to recognize these constraints. It is also interesting to note that in most of these examples (as in the majority of Aardvark questions), the asker took the time to thank the answerer for helping out.

5. ANALYSIS

The following statistics give a picture of the current usage and performance of Aardvark.

Aardvark was first made available semi-publicly in a beta release in March of 2009. From March 1, 2009 to October 20, 2009, the number of users grew to 90,361, having asked a total of 225,047 questions and given 386,702 answers. All of the statistics below are taken from the last month of this period (9/20/2009-10/20/2009).

Aardvark is actively used As of October, 2009, 90,361 users have created accounts on Aardvark, growing organically from 2,272 users since March 2009. In this period, 50,526 users (55.9% of the user base) generated content on Aardvark (i.e., asked or answered a question), while 66,658 users (73.8% of the user base) passively engaged (i.e., either referred or tagged other peoples questions). The average query volume was 3,167.2 questions per day in this period, and the median active user issued 3.1 queries per month. Figure 7 shows the number of users per month from early testing through October 2009.

Mobile users are particularly active Mobile users had an average of 3.6322 sessions per month, which is surprising on two levels. First, mobile users of Aardvark are more active than desktop users. (As a point of comparison, on Google, desktop users are almost 3 times as active as mobile users [14].) Second, mobile users of Aardvark are almost as active in absolute terms as mobile users of Google (who have on average 5.68 mobile sessions per month [14]). This is quite surprising for a service that has only been available for 6 months.

We believe this is for two reasons. First, browsing through traditional web search results on a phone is unwieldy. On a phone, it's more useful to get a single short answer that's crafted exactly to your query. Second, people are used to using natural language with phones, and so Aardvark's query model feels natural in that context. These considerations (and early experiments) also suggest that Aardvark mobile users will be similarly active with voice-based search.

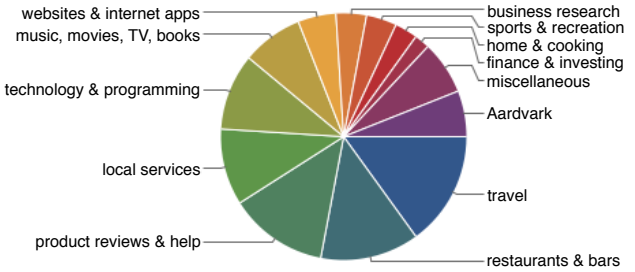

Figure 8: Categories of questions sent to Aardvark

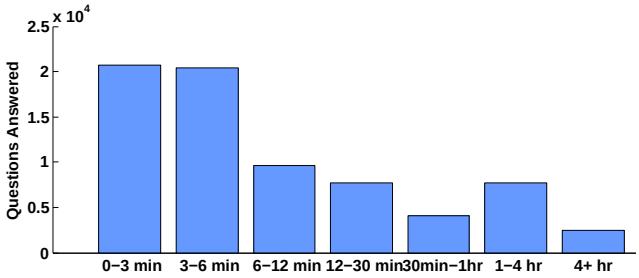

Figure 9: Distribution of questions and answering times.

Questions are highly contextualized As compared to web search, where the average query length is between 2.2 – 2.9 words [14, 19], with Aardvark, the average query length is 18.6 words (median=13). While some of this increased length is due to the increased usage of function words, 45.3% of these words are content words that give context to the query. In other words, as compared to traditional web search, Aardvark questions have 3–4 times as much context.

The addition of context results in a greater diversity of queries. While in Web search, between 57 and 63% of queries are unique [19, 20], in Aardvark 98.1% of questions are unique (and 98.2% of answers are unique).

Questions often have a subjective element A manual tally of 1000 random questions between March and October of 2009 shows that 64.7% of queries have a subjective element to them (for example, "Do you know of any great delis in Baltimore, MD?" or "What are the things/crafts/toys your children have made that made

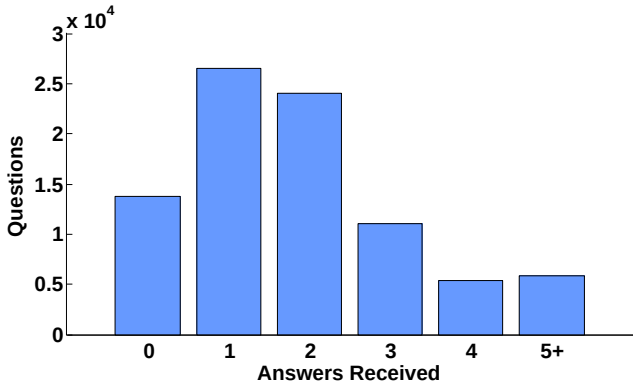

Figure 10: Distribution of questions and number of answers received.

them really proud of themselves?"). In particular, advice or recommendations queries regarding travel, restaurants, and products are very popular. A large number of queries are locally oriented. About 10% of questions related to local services, and 13% dealt with restaurants and bars. Figure 8 shows the top categories of questions sent to Aardvark. The distribution is not dissimilar to that found with traditional web search engines [3], but with a much smaller proportion of reference, factual, and navigational queries, and a much greater proportion of experience-oriented, recommendation, local, and advice queries.

Questions get answered quickly 87.7% of questions submitted to Aardvark received at least 1 answer, and 57.2% received their first answer in less than 10 minutes. On average, a question received 2.08 answers (Figure 10), [7] and the median answering time was 6 minutes and 37 seconds (Figure 9). By contrast, on public question and answer forums such as Yahoo! Answers [11] most questions are not answered within the first 10 minutes, and for questions asked on Facebook, only 15.7% of questions are answered within 15 minutes [17]. (Of course, corpus-based search engines such as Google return results in milliseconds, but many of the types of questions that are asked from Aardvark require extensive browsing and query refinement when asked on corpus-based search engines.)

Answers are high quality Aardvark answers are both comprehensive and concise. The median answer length was 22.2 words; 22.9% of answers were over 50 words (the length of a paragraph); and 9.1% of answers included hypertext links in them. 70.4% of inline feedback which askers provided on the answers they received rated the answers as 'good', 14.1% rated the answers as 'OK', and 15.5% rated the answers as 'bad'.

There are a broad range of answerers 78,343 users (86.7% of users) have been contacted by Aardvark with a request to answer a question, and of those, 70% have asked to look at the question, and 38.0% have been able to answer. Additionally, 15,301 users (16.9% of all users) have contacted Aardvark of their own initiative to try answering a question (see Section 3.6 for an explanation of these two modes of answering questions). Altogether, 45,160 users (50.0% of the total user base) have answered a question; this is 75% of all users who interacted with Aardvark at all in the period (66,658 users). As a comparison, only 27% of Yahoo! Answers users have ever answered a question [11]. While a smaller portion of the Aardvark user base is much more active in answering questions – approximately 20% of the user base is responsible for 85% of the total number of answers delivered to date – the distribution of answers across the user base is far broader than on a typical user-generated-content site [11].

[7]A question may receive more than one answer when the Routing Policy allows Aardvark to contact more than one candidate answerer in parallel for a given question, or when the asker resubmits their question to request a second opinion.

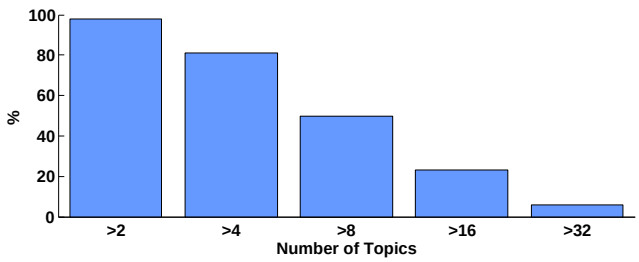

Figure 11: Distribution of percentage of users and number of topics

Social Proximity Matters Of questions that were routed to somebody in the asker's social network (most commonly a friend of a friend), 76% of the inline feedback rated the answer as 'good', whereas for those answers that came from outside the asker's social network, 68% of them were rated as 'good'.

People are indexable 97.7% of the user base has at least 3 topics in their profiles, and the median user has 9 topics in her profile. In sum, Aardvark users added 1,199,323 topics to their profiles; not counting overlapping topics, this yields a total of 174,605 distinct topics which the current Aardvark user base has expertise in. The currently most popular topics in user profiles in Aardvark are "music", "movies", "technology", and "cooking", but in general most topics are as specific as "logo design" and "San Francisco pickup soccer".

6. EVALUATION

To evaluate social search compared to web search, we ran a side-by-side experiment with Google on a random sample of Aardvark queries. We inserted a "Tip" into a random sample of active questions on Aardvark that read: "Do you want to help Aardvark run an experiment?" with a link to an instruction page that asked the user to reformulate their question as a keyword query and search on Google. We asked the users to time how long it took to find a satisfactory answer on both Aardvark and Google, and to rate the answers from both on a 1-5 scale. If it took longer than 10 minutes to find a satisfactory answer, we instructed the user to give up. Of the 200 responders in the experiment set, we

found that 71.5% of the queries were answered successfully on Aardvark, with a mean rating of 3.93 ($\sigma = 1.23$), while 70.5% of the queries were answered successfully on Google, with a mean rating of 3.07 ($\sigma = 1.46$). The median time-to-satisfactory-response for Aardvark was 5 minutes (of passive waiting), while the median time-to-satisfactory-response for Google was 2 minutes (of active searching).

Of course, since this evaluation involves reviewing questions which users actually sent to Aardvark, we should expect that Aardvark would perform well — after all, users chose these particular questions to send to Aardvark because of their belief that it would be helpful in these cases.[8] Thus we cannot conclude from this evaluation that social search will be equally successful for all kinds of questions. Further, we would assume that if the experiment were reversed, and we used as our test set a random sample from Google's query stream, the results of the experiment would be quite different. Indeed, for questions such as "What is the train schedule from Middletown, NJ?", traditional web search is a preferable option.

However, the questions asked of Aardvark do represent a large and important class of information need: they are typical of the kind of subjective questions for which it is difficult for traditional web search engines to provide satisfying results. The questions include background details and elements of context that specify exactly what the asker is looking for, and it is not obvious how to translate these information needs into keyword searches. Further, there are not always existing web pages that contain exactly the content that is being sought; and in any event, it is difficult for the asker to assess whether any content that is returned is trustworthy or right for them. In these cases, askers are looking for personal opinions, recommendations, or advice, from someone they feel a connection with and trust. The desire to have a fellow human being understand what you are looking for and respond in a personalized manner in real time is one of the main reasons why social search is an appealing mechanism for information retrieval.

7. RELATED WORK

There is an extensive literature on query routing algorithms, particularly in P2P Networks. In [5], queries are routed via a relationship-based overlay network. In [15], answerers of a multicast query are ranked via a decentralized authority score. In [6], queries are routed through a supernode that routes to answerers based on authority, responsiveness, and expertise, and in [10], supernodes maintain expertise tables for routing. Banerjee and Basu [1] introduce a routing model for decentralized search that has PageRank as a special case. Aspect models have been used to match queries to documents based on topic similarity in [12], and queries to users in P2P and social networks based on expertise in [6]. Evans and Chi [9] describe a social model of user activities before, during, and after search, and Morris et al. [17] present an analysis of questions asked on social networks that mirrors some of our findings on Aardvark.

[8]In many cases, users in the experiment noted that they sent their question to Aardvark specifically because a previous Google search was difficult to formulate or did not give a satisfactory result. For example: "Which golf courses in the San Francisco Bay Area have the best drainage / are the most playable during the winter (especially with all of the rain we've been getting)?"

8. ACKNOWLEDGMENTS

We'd like to thank Max Ventilla and the entire Aardvark team for their contributions to the work reported here. We'd especially like to thank Rob Spiro for his assistance on this paper. And we are grateful to the authors of the original "Anatomy" paper [4] from which we derived the title and the inspiration for this paper.

9. REFERENCES

[1] A. Banerjee and S. Basu. A Social Query Model for Decentralized Search. In *SNAKDD*, 2008.

[2] H. Bechar-Israeli. From <Bonehead> to <cLoNehEAd>: Nicknames, Play, and Identity on Internet Relay Chat. *Journal of Computer-Mediated Communication*, 1995.

[3] S. M. Bietzel, E. C. Jensen, A. Chowdhury, D. Grossman, and O. Frieder. Hourly analysis of a very large topically categorized web query log. In *SIGIR*, 2004.

[4] S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. In *WWW*, 1998.

[5] T. Condie, S. D. Kamvar, and H. Garcia-Molina. Adaptive peer-to-peer topologies. In *P2P Computing*, 2004.

[6] J. Davitz, J. Yu, S. Basu, D. Gutelius, and A. Harris. iLink: Search and Routing in Social Networks. In *KDD*, 2007.

[7] A. R. Dennis and S. T. Kinney. Testing media richness theory in the new media: The effects of cues, feedback, and task equivocality. *Information Systems Research*, 1998.

[8] J. Donath. Identity and deception in the virtual community. *Communities in Cyberspace*, 1998.

[9] B. M. Evans and E. H. Chi. Towards a Model of Understanding Social Search. In *CSCW*, 2008.

[10] D. Faye, G. Nachouki, and P. Valduriez. Semantic Query Routing in SenPeer, a P2P Data Management System. In *NBiS*, 2007.

[11] Z. Gyongyi, G. Koutrika, J. Pedersen, and H. Garcia-Molina. Questioning Yahoo! Answers. In *WWW Workshop on Question Answering on the Web*, 2008.

[12] T. Hofmann. Probabilistic latent semantic indexing. In *SIGIR*, 1999.

[13] B. J. Jansen, A. Spink, and T. Sarcevic. Real life, real users, and real needs: a study and analysis of user queries on the web. *Information Processing and Management*, 2000.

[14] M. Kamvar, M. Kellar, R. Patel, and Y. Xu. Computers and iPhones and Mobile Phones, Oh My!: a Logs-based Comparison of Search Users on Different Devices. In *WWW*, 2009.

[15] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina. The EigenTrust Algorithm for Reputation Management in P2P Networks. In *WWW*, 2003.

[16] D. Klein, K. Toutanova, H. T. Ilhan, S. D. Kamvar, and C. D. Manning. Combining heterogeneous classifiers for word-sense disambiguation. In *SENSEVAL*, 2002.

[17] M. R. Morris, J. Teevan, and K. Panovich. What do people ask their social networks, and why? A Survey study of status message Q&A behavior. In *CHI*, 2010.

[18] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank citation ranking: Bringing order to the Web. *Stanford University Technical Report*, 1998.

[19] C. Silverstein, M. Henziger, H. Marais, and M. Moricz. Analysis of a very large Web search engine query log. In *SIGIR Forum*, 1999.

[20] A. Spink, B. J. Jansen, D. Wolfram, and T. Saracevic. From e-sex to e-commerce: Web search changes. *IEEE Computer*, 2002.

[21] L. Sproull and S. Kiesler. Computers, networks, and work. In *Global Networks:Computers and International Communication*. MIT Press, 1993.

[22] M. Wesch. An anthropological introduction to YouTube. *Library of Congress*, 2008.