KudoRank

A Market Model for Mail Management

Rohit Khare, Ph.D

CommerceNet Labs Technical Report 05-06
November 2005

Abstract

This paper describes a novel approach to searching personal information archives: a market mechanism for 'pricing' the value of information from different authors. Traditional text indexing assumes all documents in a collection are equally important, so the ranking of search results is based on the frequency of term occurrence, age, or other endogenous properties of the result set itself. We investigated whether higher-quality results can be obtained by ranking documents based on analysis of the social network between correspondents. This approach appears particularly promising for personal information archives, where much of the data to be retrieved is private and little contextual metadata exists.

In particular, we posit that individual messages are like products in a marketplace: readers 'pay' for them with their attention, and the proceeds accrue to both the authors of messages and the messages it cites (as royalties for derivative work). The economic equilibrium that converges after several rounds of trading corresponds to a 'market-clearing price' for each message. That score can then serve to partially order the entire collection for further applications such as indexing, spam filtering, compliance management, or rerouting messages to mobile devices or other correspondents.

Disclaimer

Acknowledgments

The implementation in Appendix B was largely developed by Ben Sittler, and the entire project benefitted greatly from discussions with Adam Rifkin.

1. “Who Said What” Matters

The standard problem statement for text search is finding strings within other strings – searching or pattern-matching across a homogenous space of symbols. This is perfectly appropriate for most collections of documents, which can be considered the product of a single ‘voice’. Even in multiple-author collections such as a newspaper archive, the “reporter” is merely one more field users can explicitly query over, not an implicit factor for ranking.

When it comes to indexing a mailbox, most search tools treat this problem the same way. However, a mailbox is an archetypal example of a document with many authors – like a good postmodern literary theorist, one would have to deconstruct it line by line to accurately identify the voices contributing to it.

From this perspective, the text found on one’s hard drive is in fact the product of thousands of other people, all of whom are trusted to varying degrees. In real life, this distinction is critical when assessing the results of a query: we instinctively rank authorities.

The recent revolution in the quality of Web searching also stems from this insight, though due to a very different mechanism. Roughly speaking, each domain name presumably corresponds to a single organization or individual. Because pages within a website tend to be tightly interlinked, principal eigenvalue approaches such as Kleinberg’s HITS or PageRank that collect ‘votes’ for authoritative pages tend to also rank entire sites highly. Regardless, though, these approaches fail to consider multiple authorship within a single resource, either directly or indirectly.

Even so, our goal is not merely to detect that certain passages of text are attributable to a single author, and then offer users a direct way to query for, say, quotes from newspaper articles. Our goal is to leverage implicit authorship information to analyze networks of correspondents.

2. Why Mail Search is not Text Search

The first step is to establish merely that the challenge of searching email archives is not met by text indexing alone. Beyond the obvious mistake of indexing an entire mbox or .pst file as thought it were a single text document, there is still the challenge of coping with the unique structure within each email message.

Just as Web page searching requires parsing HTML to extract the user-visible sequence of words, formatted email has several layers of processing to be undone before the indexable content can be identified: paragraphs get reflowed around line breaks; various characters are used to prefix quoted lines; character sets have varied encodings into legacy ASCII; and MIME-formatted digests have to be burst out and attachments stripped.

Synonyms and stemming are also well-known aids for natural-language information retrieval. Adapting them to the decidedly non-natural space of email addresses is also not obvious. Establishing that "foo@yahoo.com" and "bar@khare.org" are the same person is even harder than comparing the various forms of canonical names attached those addresses.

Finally, headers must be considered separately from bodies. Not only must they be parsed properly and discarded (in the case of Received: or Message-Id:), but some headers are clearly more important than others (such as Subject: lines or Urgent: flags).

As a result, a simple text search for the string "mayfield" could lead to cognitive overload for the user because the results included:

- messages to or from anyone@mayfield.com
- to or from the person 'Jim Mayfield'
- typos ("The Yankees mayfield a new pitcher")
- company press releases from startups funded by Mayfield
- and many other sorts of information...

3. Why Mail Searching is not Web Searching

Arguably, the state-of-the-art for searching mail archives today is actually Web searching. Tools exist to convert mailboxes into hyperlinked web archives such as MHonarc, hypermail, and pipermail. These are typically used for collaborative applications (such as mailing lists) rather than for private email. Nevertheless, if we imagined crawling, indexing, and ranking such a web-mail corpus we suspect there would be several immediate objections:

1. There is no link between articles to the individuals that wrote them — mailto: links aren't included in ranking calculations the way http: URLs are.

2. "Thread," "Date," and "Author" index pages only serve to confuse matters by grouping together *all* messages on an equal basis, rather than grouping specific threads or individual authors as resources to be ranked independently.

3. There is no immediately obvious way to strip out the purely "navigational" links inserted before and after the original mail text, in order to emphasize thread-link relationships over mere adjacency in sequence order.

4. There is no obvious way to "backpropagate" scores from the sites that an email message refers to. It seems reasonable to rank a well-reasoned message that cites the New York Times more highly than one that only links to the New Zork Times.

5. There is little precedent for incorporating the frequency of access into ranking. It seems reasonable to increase the rank of a message the more often a user refers back to it in their own reading. In the terms of Web ranking algorithms, reading should also become a form of linking.

Furthermore, the more structured format of email messages encodes additional information not envisioned for hypertexts. Recipients may be represented as links, but what about the subtle social status information encoded in the choice between placing recipients in the To:, CC: and BCC: headers? In-Reply-To: headers may also be obvious links, but what about the myriad quoting formats used by other mail tools to indicate that the following paragraph(s) come from another source?

Yet other concerns focus on fields that are similar between Web pages and mail messages. How can we ensure that Subject: hits are ranked more highly than body text, just as HTML anchor text is used to label page contents? Citation links may seem similar to hypertext links, but they could mean the opposite: research into community-clustering shows that on USENET discussions, links should be interpreted as **dis**approval, rather than the Web model of links-as-endorsement.[1]

4. A Microeconomic Model of Mail

Our fundamental assumption is that what you want to read is (like) what you have read. As a result, we believe that search results should be presented in order of the probability you

[1] http://www.almaden.ibm.com/cs/people/srikant/papers/www03_social.pdf

would have liked to have read them, even if you hadn't already. The question is, how should that probability be estimated?

Our answer is an economic model inspired by the ancient institution of copyrights. We created a hypothetical currency, kudos, to signal approbation by readers for writers' works:

1. a person spends kudos for the messages they read (pay for content).
2. a person earns kudos from the messages they read (postage/attention bond).
3. a person earns kudos from the messages they write.
4. a person spends kudos to sponsor the messages they write (furthermore, anything a person writes must also have been read by that person, a so-called 'spam tax').
5. a replied-to or forwarded message transfers kudos to the message(s) it cites (derivative work).

Nonetheless, these principles are still not sufficient to determine a particular ranking algorithm. There are a few remaining broad issues:

- What do we do with To:'s and Cc:'s? Should a writer be penalized by having to paying readers to read his or her works?
- Does an In-reply-to: reference also improve the standing of the citing work? (should thread-heads be weighted more highly than thread-tails?)
- How is reading activity represented? Additional edges, or additional weight placed on existing edges? Should virtual "read receipt" messages be created to track responses in real-time?
- If it is indeed a typed, weighted graph, are there more complex models that happen to have more fidelity to the currency metaphor? Should a fixed 'tax' percentage be credited to the market as a whole, or to the owner of a mail corpus (see *Personalization*, §7.4).

The net effect of this model, though, is to value mail messages in the opposite way that, say, PageRank values Web pages. A web page is presumed to only be as good as the material that *cites it*; a mail message is only as good as the material/correspondents *it cites*.

As a mathematical aside, there is a problem with rank-sinks: people who write messages that are well-read, yet never read (contribute) back to the main graph. Such leaf cycles are handled by adding a 'tax' in this mode, which serves to damp down the whole system. At each turn, some amount of 'kudos' are credited from every Post and/or Person to a market-manager, who then either disperses the funds equally to all, or only back to the corpus-owner.

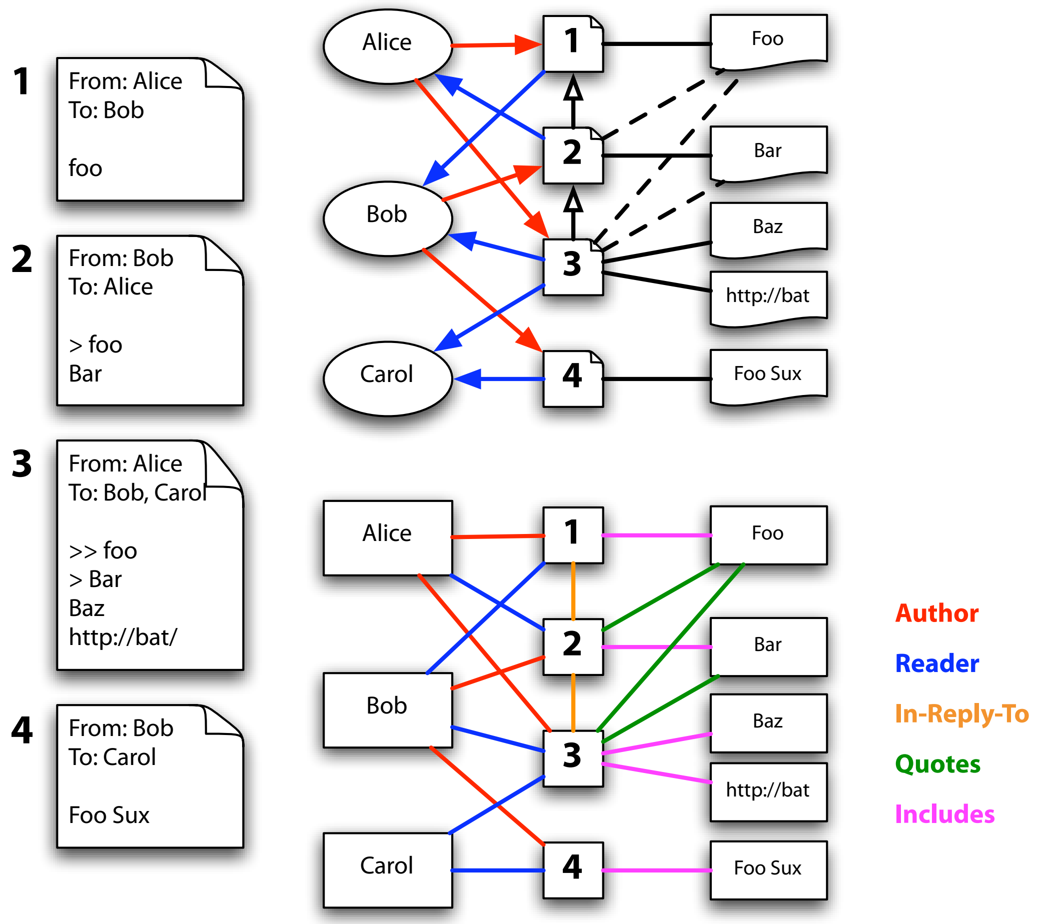

FIGURE 1: A sample email corpus and graph constructed from it (colored edges)

5. Preliminary Simulation Results

One of the major reasons we developed the experimental code in the Appendix was to establish whether or not this family of microeconomic models is subject to the "totem pole effect." That is, in building a graph of People and Post nodes, will value primarily accrue to people and tend to clump all of that person's postings together in the ranking? And indeed, running it over tens of thousands of messages from different mailboxes of mine and past archives reveals that it does not tend to merely count in- or out-degree (sending me lots of traffic doesn't raise your rank). In fact, it correctly identifies highly-connected components (correspondence chains) and thus ranks most highly, say, meeting-planning negotiations with several other close colleagues. It also correctly ranks spam messages and unread newsletters poorly.

It is that last point which has become most intriguing in our early runs: how to distinguish between spammers, celebrities, and long-lost friends?

- The friend: you might reply to, creating a shorter path to that person.
- The spammer: you may never read, leaving those messages disconnected from the main graph.
- The celebrity: you may always read, but never write back to directly… however, you might cut-and-paste his or her content in another mail to someone you do trust, and thus granting kudos back to the original work (see *Quotation Is Citation*, §7.3).

An open question for users to consider when tailoring the system to their preference, is whether thread-heads or –tails should be more highly ranked. Depending on the directionality of an in-reply-to: arc, kudos could either concentrate 'upwards' in the original message, or flow down. For example, in an email corpus where the dominant pattern is that person A asks a question and persons B, C, and D reply with their answers, one would probably want the most-trusted reply ranked first. On the other hand, if it were a social event invitation with dozens of one-line replies, the original may be most useful.

6. Ranking Search Results

The primary application we envision for this sort of ranking is a personal search engine. Ångströ is a service that indexes a user's own private information and uses Kudo pricing to rerank its search results. At minimum, it is an indexer that searches email headers, bodies, and attachments intelligently; ultimately, it should grow to encompass any sort of multi-agent discourse. There are many artifacts found on a modern PC that are not traditional hypertext documents, but do have implicit authorship links:

- Instant Messaging and chat logs are like one-line emails
- Buddy lists are trusted correspondents
- Telephone call records are trust links, whether SIP/VoIP/POTS
- Version control logs (multiple-authors changing a shared document)
- RSS/Atom article syndication feeds
- News article clippings by journalists

- WikiWikiWeb and other collaborative Web authoring systems.
- Saved web pages with "signature" data at the bottom or in HTML source comments
- Usenet news messages
- Internet Relay Chat logs and even connectivity graphs
- Calendar and scheduling items (calsched)
- Address book entries (vCard) – especially for all the contact info that 'flies by' without ever being explicitly entered into an address book
- Form submission data to websites – making an airline reservation or issuing a search engine query is tantamount to authoring a message sent to a remote web service agent.
- Even clipboard entries: the temporary data the users cuts-and-pastes from represents an indication of interest in those paragraphs

All of this discursive data, of course, is still interconnected with the "regular" hypertext corpus that represents the rest of the user's information archive (see *Notes on the Personal Web*, §A).

7. Additional Ranking Criteria

The current experimental program is aimed at analyzing "static" email archives; there is more work ongoing in tailoring this for "online" use of email. This section describes ways to collect and apply more refined knowledge about implied links between resources; ways to modify the market for personalization and to incentivize citation; and dynamic query-dependent re-ranking.

7.1 More Accurate Usage Information

By watching user access to email archives, it is possible to accumulate more reliable data on what the user has or has not read. It is not sufficient to simply use the read/unread flag written to disk by many mail user agents, since many users have the email habit of marking entire chunks "read" to get them off of their unread list; or, conversely, go back and re-flag a message as unread to remind themselves to take action on it later. Furthermore, static analysis cannot reveal how often a message is re-read over time.

On the other hand, simply wiretapping read accesses on IMAP or POP is not sufficient, either. Simply finishing reading one article can cause the mail client to request the next one; or it could download many messages all at once for offline reading. More sophisticated inference rules will be necessary to reliably estimate how likely a user is to have actually read the contents of the message (as well as which portions of long messages).

There are also additional degrees of "reading": when a user clicks through a link sent in email, that is further approbation for that correspondent. This can be noticed either by correlating hyperlink traffic by proxying HTTP or by dynamically re-writing URLs found in email messages to point to a redirector (hit-counter) running within the search engine itself.

All of these bits of more accurate usage information can be used to create additional links in the mail graph, or alternatively, to modify the weights of existing edges.

7.2 More Accurate Attribution

Rarely is an entire document authored by a single agent, and even more rarely is the author definitively identified within the document. The From: header is only a starting point for identifying the provenance of a text, especially for attachments that may have been forwarded from others.

Sometimes text is directly pasted in, preceded by a "on <date> <person> said:," or followed by a single URL. Other kinds of file formats store metadata of varying quality, such as Microsoft Word's Document Properties feature, or CVS' checkin stamps. Decoding this kind of "markup" is critical for accurate sub-document attribution. It involves inferring links between email addresses, IM handles, and "natural language" name variants.

Unifying all of a person or organization's identities may also be too strong. It is plausible to assert that an agent is only trusted in certain areas of expertise. Our model for this is subsidiary identities (pseudonyms). This way jbone+music can be distinguished from jbone+unix to establish the credibility of different classes of documents jbone has authored.

An extreme form of this might be to automatically create "hub" pages (in the sense of Kleinberg's HITS algorithm) based on the most-frequently-used rarest-terms in the corpus. These index pages would be a way to identify the keywords idiosyncratic to each user's interests.

7.3 Quotation Is Citation

Email, and for that matter, regular mail, is not a hypertext medium — it predates the notion of hyperlinking for citation and instead relies on quotation to indicate dependence on a source document. Identifying such overlaps is a more prosaic string-matching challenge, akin to compression with very large dictionaries.

Our goal is to refine the graph created above by breaking down Posts into Paragraphs, a third type of vertex. Paragraphs are *atomic* in that recurring uses of the "same" phrase are conserved. So each Post now has a Sequence of directed edges to each of its constituent Paragraphs. Furthermore, each Paragraph has a back-link to its own Author. Alternatively, this is equivalent to decomposing every paragraph of text in the corpus as an individual email message.

As an aside, we can also use this paragraph-level breakdown to recombine entirely new user interfaces for threads, such as a Talmudic commentary in the margins. Pretty-printing can effectively summarize entire threads at a glance.[2]

7.4 Personalization

Unlike public (anonymous) Web searching, email search is intensely personalized by default. The need to collect taxes to damp out the effect of "rank sinks" can be turned back into a way to "focus" the interest in the graph by rebating the rest to a specific user, group of users, or even a specific time window. The default would be to rebate taxes back to all nodes in the graph equally, but positive discrimination in this sense can increase the rank of pages related just to that subset.

This mechanism can also be used to add a measure of currency (real-time) to the system. As it is, all transactions in the Kudo market are presumed to clear simultaneously. One step in tailioring the system to user preferences is to apply an 'interest rate' for discounting past connections in favor of recent ones.

7.5 Kudos for Citing Trusted Sources

Web searching depends on interconnectivity between pages, but since there are so few links from the Web *to* email, mail archives are prone to becoming disconnected from the public

[2] Or GMail's still-unique 'cards' UI model for threads that de-duplicates quoted text.

Web. Only Kleinberg's HITS algorithm comes close to acknowledging that flaw. Consider that if an email message cites a newspaper article, the newspaper is still unlikely to link back to that email archive. This makes citation a futile exercise, since the rest of the Web becomes a rank sink. This is an undesirable economic disincentive.

It seems to us that Posts that tend to point to more "reliable" Web sites are themselves more useful. This sort of external score assignment for entire Web sites is already well-studied; it is a site's PageRank, for one. That score should be converted back into a certain "rebate" in kudos to the posts which cite it.

7.6 Clustering and Contextual Re-Ranking

So far, we have been focusing on Kudos as a query-independent ranking mechanism; the actual query term may also be useful to refine the ranking of results. Suppose that for a certain keyword, all the hits occur in messages sent during Q1 of 1998. It might be more useful to re-rank them according to the corpus as it *would have appeared* back then. This could be accomplished by only analyzing messages within the timespan of the text hits, or by only perturbing the ranks of people at different points in history. After all, trusted correspondents come and go. More insidiously, our trust in the same correspondent can vary over time.

8. Advantages of a Personal Proxy Server

One promising path for delivering this functionality is to package it as a personal ("desktop") proxy server. A piece of software installed on the user's PC that terminates all POP/IMAP/SMTP and even HTTP(S) traffic would allow Ångströ to inspect all content the user reads as well as collect information on access and usage of those resources. This allows the end-user to continue using his or her preferred email tools, web browsers, etc. without any visible reconfiguration. The only difference is, that as a local HTTP server, the Ångströ service appears to be a new website, running on localhost, that has a look-and-feel very similar to public Web search engines today.[3]

The user would invoke the query site using a keystroke, toolbar, widget, or other means; then enter a single query string; and choose to search or go directly to the most-likely hit. The reply page is a ranked, formatted list of resources with excerpts. For image searches, it could be a

[3] Google Desktop Search, released a few months after this paper was drafted, goes one better to integrate the results into the public site by proxying Windows Sockets connections.

grid of thumbnails, or for slide searches or other multi-page images, it could be a grid of individual slides mentioning the terms. Other output forms are clearly conceivable, such as graph views, bargraphs of trust levels or Kudo scores.

The search interface could also be more tightly integrated with existing mail clients, word processors, etc by either using client-side scripting to insert results into applications, or replacing the IMAP search protocol on the fly, so that ordinary mail search results come back sorted automatically.

8.7 Application Integration

By installing it as a proxy, the user need not change his or her daily application usage – we merely create a new "website" for issuing personal queries. A basic Web interface could still grow into a tightly-integrated application-specific user interface using client-side scripting, or by returning results as an IMAP search query (thus displaying ranked results within Outlook/Eudora/etc.).

The logical extreme of this approach is 0U – the "zero-unit" server, by analogy to the nU rack-mount server appliance metric. Because the entire interface to a search proxy is defined in terms of TCP/IP messages on certain ports, the application can be packaged together with an entire operating system image and run inside of a virtual machine. This way, if Ångströ were written in Linux, it could still be shipped on Windows by running inside of a VMWare session and intercepting IP traffic by explicitly addressing this "appliance" as a proxy for accessing SMTP/HTTP/IMAP/POP/FTP/etc.

8.8 Multiprotocol Integration

Furthermore, proxying multiple protocols doesn't simply provide more corpuses to search; they form a single *unified* corpus. Merging the streams enables determining whether an email is "interesting enough" that you actually bother clicking through to the URLs from it, or that you frequently revisit the blog of one of your correspondents. If the interaction effect between streams is persuasive enough, it might even motivate pre-caching web pages them upon receipt of an email (or, precaching news stories upon receipt of an RSS notification).

It also gives developers the ability to write scripts that "react" to incoming events on any protocol type in a single form. Call it an 'XGI' script, an extensible gateway interface by analogy to CGI. Now, an incoming email can trigger a script that fires off an outbound web service request. More prosaically, event handlers can drive incremental re-indexing of newly-arrived content.

Awareness of multiple protocols also enables us to recognize gatewaying of one protocol into another. A 'smart' HTTP proxy for Hotmail, say, could figure out that the actual unit of traffic going back-and-forth is an RFC822-formatted email, not the pretty HTML with all of its additional menus, advertising, &c. This sort of encapsulation also occurs with Web-browsing of Usenet news, for example.

8.9 Offline Integration

By maintaining a copy of all past traffic, not only can an Ångströ engine find past pages that may have been modified since last displayed, that content can also be made available when the user's network connection is interrupted. This can either be in the form of silent, automatic HTTP-compliant caching rules, or by explicitly warning users that the content is dated by inserting a toolbar or notice into the DHTML it returns.

Beyond merely caching, a personal search proxy can also become a more active terminator for certain transactions: it could accept and queue mail for delivery when back online, or make sure to refresh cache contents if accessed heavily. This is particularly potent for Web services invocations.

8.10 Network Effects

A network of personal Ångströ engines is another intriguing scenario. A personal search proxy should also be replicatable (to move the index off to another host that may be online 7x24, unlike a laptop), and hence remotely accessible. This includes access by the owner from other locations, but may also include access from the owners' closest colleagues, or, indeed, the public at large.

This raises privacy concerns that Ångströ must be sensitive to: content must be easily flaggable as personal/secure rather than public/shared, and hence not even displayed in the search results if the querier is not granted access to that result. Another level of safeguard might be to relay a query to some friends' Ångströ engines, but still have those engines ask their users whether to divulge the results back.

Finally, a personal search proxy can publish more than just a query interface. By hosting a blog or other content, it can power a personal web site. That much is well known; but in conjunction with decentralized trust management (see §8.6), it could automatically maintain a "semipermeable" archive – only visible to friends who are "close enough" in Kudo rank.

8.11 Versioning

By proxying the WebDAV protocol, it is possible for a personal search proxy to portably index ordinary files and folders, too. Initially, this is useful for tracing the use of files sent as email attachments or downloaded from websites. By watching the user's working folders, too, though, it can detect renaming of files, creation of new versions, and creation of derivative works (PDF from a Word file, say).

Beyond that, however, the proxy could store each version of a file, automatically inferring the version history of a resource. This would allow users to not only search the current version of their archive, but even past versions. This is a novel interface to version control systems: while programmers may reason in terms of "which point release do I want to roll this module back to?," ordinary writers may only ask "what version was I still discussing 'ethanol' in?"

8.12 Identity Management

A multiprotocol personal proxy is also in a position to opportunistically secure traffic, especially to/from other Ångströ engines. By assigning a public keypair to a proxy, it can at least encrypt sessions even if the users' desktop application is still talking to the proxy in the clear. By also responding to key lookup inquiries, it is in a position to automatically encrypt and digitally sign email traffic to other correspondents using such technology.

It can also more actively manage a user's identity on the Web. Like a personal web anonymizer, it can automatically generate per-use, per-site, per-correspondent-pair disposable email addresses, login names, and passwords which it inserts in-line into network transactions. Combined with the pre-caching facilities discussed earlier, it can even regularly "crawl" for your bank statements, frequent flier balances, etc. to maintain a single view of a user's confidential information.

9. Additional Applications

There are many additional applications of a microeconomic model for valuing messages in a multi-agent discourse, beyond searching personal information archives alone. It holds broader implications for trust management, collaborative filtering, inference engines, and exploratory archive analysis.

9.13 Decentralized Trust Management

Our Kudo market simulation over the graph of People, Posts, and Paragraphs assigns prices to all People too, not just Posts. This scoring is a form of decentralized trust management: it permits each user to quantify how much that user trusts any of the others. We can also compute distances between agents, in a form useful for summarizing the state of massively-multiplayer online role-playing games.

Further refinement of this aspect may call into question whether correspondents listed in To:. CC:, BCC:, and mailing lists headers are to be equally trusted, or differentially weighted.

9.14 Recommendation Systems

Such trust metrics are useful for creating online collaboration systems with subtler access control distinctions than merely "buddy vs. stranger." Mailing lists are another binary in/out group coordination mechanism. Instead, email with trust ranking allows a graduated way for recommending information to other users. On that basis, a client can compute not just the top-10-most-popular news stories, but the top-10-most-popular news stories *according to your friends*.

The key is collecting read/access data from other users. Again, though, this raises tough new privacy concerns. Our invention is to send "read receipts," but 1) only include the MD5 hash of the URL visited; 2) drop some random percentage of them; 3) insert some random percentage of made-up ones; 4) delay delivery of them randomly; and 5) digitally sign them with a widely publicized key. All of these defenses increase "plausible deniability," so that no one friend can reconstruct a user's reading habits with any certainty; and also ensure that friends only discover what site it refers to if they, too, visit the same location.

Making it easy to recommend information is like making mailing lists (such as FoRK) into a videogame. A quick bookmarklet lets users flag a page (or paragraph) as of interest to "self"/ "friends"/"all". Then, just as the Kudo ranking algorithm gives credit to those authors whose emails' contain links you visit most often, the correspondents whom you share interests with will automatically cluster together.

9.15 Decentralized Databases

Part of the core infrastructure for building this systems is what we term a "factoid database" – a way of storing opinions, rather than facts. Traditional DBMSes are fine for storing absolute statements such as "X = 5", but inadequate for reasoning about "Adam claimed X=5 an hour ago." A decentralized database (DZMS) must cope with ambiguity and hence notify its clients when beliefs change. That's how we can store the assertion "Rohit believes that khare @ xent is identical to Rohit @ KnowNow with 99% certainty" (until, of course, the passwords to one of those two accounts are compromised).

One user interface to that is a FactSheet, a table with subject/relationship/object and speaker/timestamp/confidence triples in each row. Then one can construct formulas for inferences. Some of the fundamental logical elements are $\forall$, $\exists$, $\cup$, $\cap$, pick the most recent, multiply confidence scores when chaining, and so on.

This sort of conditional inference occurs throughout our problem domain:

- Who is a member of a mailing list (at a given point in time)?
- Who is the owner of a mailing lists?
- When was a forwarded email likely to have been written? (since you didn't receive it directly)
- Who else is a member of the same organization? (domain names are only a start)
- How likely is a correspondent to still be awake/logged in?
- Where in the world are you? (according to your calendar schedule, timezone, cellphone records, etc…)
- How soon are you likely to respond to this person?

9.16 Data Mining

Kudo ranking is also a guide for exploring archives of correspondence. Consider the case of a lawyer coping with gigabytes of email resulting from a subpoena: how can he or she determine patterns of conspiratorial communication, or at the very least, be guided to the relevant messages rather than the vast majority of routine junk from Accounting, HR, and daily operations?

In this case, we can apply the technology to multiple corpuses simultaneously – we can calculate ranks according to a "god's-eye" view of the discourse, or compare and contrast the importance of messages from multiple participants' perspectives.

It may also be useful to feed paragraphs of text back into public search engines to determine the putative source of certain quotes – automatically recognizing where a PDF attachment may have come from originally. Certain aspects of internal document structure come back to the fore in these cases. A research librarian considering a corpus of technical reports and standards-group mailing lists needs to focus on detecting bibliographic citation data, or at least separating text into chunks based on perceived outline/hierarchy (based on separators, headings, advertising boxes, etc).

Appendix A: Notes on the PersonalWeb

What is a personal web?

- My bits. No one else's.
- Totally Private. <---> Stuff That's Cached. <---> Public and Googlable.
- Not private out of paranoia, just private out of laziness.
- No two personal webs are the same.
- A graph of the bits that represent your interaction with the network and things that you are personally interested in.
- The useful version of My Computer.

Why would you want a personal web?

- More for searching than for browsing?
- Would we benefit from something that is "my browsable, personal web"? From a research point?
- Just for search, not for offline. Text-only not images.

Inventory of artifacts in a personal web:

- Bookmarks file
- Access logs
- Error logs
- Past query terms
- Inbound email and attachments (email lists, google news alerts, plaxo message, vacation messages, etc)
- Outbound email and attachments
- Playlists
- Archived IM sessions
- RSS feeds

- SSL transactions
- ftp in and out
- cvs/svn transactions
- Appointments
- Deadlines
- To do items = appointments without a defined deadlines)
- Recommendations -- ?What are they? ("On date X Adam recommended URL Y")
- Local files
- Rules for profiling of files on disk—should be able to filter out (common) OS files
- SOAP—way too science-fiction today
- Image search
- Powerpoint slide view
- A spider that pours things in that you're likely to like
- Attachments

An example personal web homepage:

- I should always want to go to my personal web home before the public web.
- Most important emails I have yet to read.
- Most important news items I have yet to read ("top 25 according to your friends").
- Up to date calendar and address book. (tell me when highly-trusted friends are in town by watching their calsched RSS feeds)
- Prioritized to do list for filling in any available time. (in fact, the proxy server could block access to the internet while you're supposed to be focused on one task)
- GossipSense – what if the "ads" inserted into your browsing experience were actually contextual hits from your parade of unread email (i.e. an Infoworld piece on a conference should advertise your friends who are speaking there)

Appendix B: Source Code for Kudos.py

```
#!/usr/bin/python --

import time
import math
import os
import sys
import mailbox
import rfc822

addrheaders = ( 'from', 'to', 'cc', 'message-id', 'in-reply-to' )
metaheaders = ( 'subject', 'from', 'date' )

metabyid = {}
metaidx = {}

def addOrInc(hash, key, idx):
  if not hash.has_key(key):
    hash[key] = map(lambda hdr: 0, addrheaders)
  hash[key][idx] += 1

def canonid(idpair, isid = False):
  idpair = map(lambda x: x.lower(), idpair)
  ret = (idpair[1].split('@'))[0]
  if isid:
    ret = rfc822.dump_address_pair(idpair)
  return ret

linktypes = [
#  ('in-reply-to', 'message-id'),
  ('message-id', 'to'),
  ('message-id', 'cc'),
  ('from', 'message-id'), # authorship
  ('message-id', 'from'), # author reads
  ('message-id', 'in-reply-to'), # transclusion
  ('to', 'message-id'), # presumed readership
  ('cc', 'message-id') # presumed readership
  ]

graph = {}

allnodes = None

god = None

count = {}

class Node:
  def __init__(self, label, score = 0.0):
    self.links = []
    self.label = label
    self.score = score
    self.oldScore = score
  def propagate(self):
    global allnodes
    if len(self.links) == 0: self.links = allnodes
    for link in self.links:
      link.score += self.oldScore / len(self.links)
  def balance(self):
    self.score = 0.85 * self.score + 0.15 * self.oldScore
  def start(self):
    self.oldScore = self.score
    self.score = 0.0
  def link(self, othername):
    global graph
    if not graph.has_key(othername):
      graph[othername] = Node(othername)
      graph[othername].links.append(god)
    self.links.append(graph[othername])

def iterate():
  for node in allnodes:
    node.start()
  for node in allnodes:
    node.propagate()
  for node in allnodes:
    node.balance()

def addlink(src, dst):
  try:
    if not graph.has_key(src):
      graph[src] = Node(src)
      graph[src].links.append(god)
    graph[src].link(dst)
#    graph[src].link(dst)
#    graph[src].link(dst)
#    graph[src].link(dst)
#    graph[src].link(dst)
#    graph[src].link(god.label)
  except:
    pass

god = Node(canonid(('', 'god@heaven.org')), 100000000.0)
graph[god.label] = god

def printgraph():
  for key in graph:
    node = graph[key]
    first = "%5.f" % (1000 * math.log(node.score + 1))
    #first = "%9.f" % node.score
    try:
      print first + '     ' + " ".join((metabyid[node.label][metaidx['subject']] + ' [' +
canonid(rfc822.parseaddr(metabyid[node.label][metaidx['from']])) + ']').split("\n"))
    except:
      try:
        print first + ' ' + node.label + ' ' + str(count[node.label][:3])
      except:
        print first + ' ' + node.label

def main(args):
  # usage: matritsa.py [ mbox [ god-loves-addr [ nr-iters ] ] ]
  iters = 10
  addridx = {}
  for header in addrheaders:
    addridx[header] = len(addridx.keys())
  for header in metaheaders:
    metaidx[header] = len(metaidx.keys())

  mboxfilename = args[1]
  del args[1]
  if len(args) > 1:
    addlink(canonid(('','god@heaven.org')), canonid(('',args[1])))
    del args[1]
  if len(args) > 1:
    iters = int(args[1])
    del args[1]
  fd = open(mboxfilename, 'rb')
  mbox = mailbox.UnixMailbox(fd)
  if 1:
    table = []
    while 1:
      msg = mbox.next()
      if not msg: break
      # FIXME: can't handle RMail's broken Message-ID with nested <>'s
      for msgid in msg.getaddrlist('message-id'):
        metabyid[canonid(msgid, True)] = map(lambda header: msg.getheader(header), meta-
headers)
      table.append(map(lambda header: msg.getaddrlist(header), addrheaders))
      if not (len(table) % 100):
        sys.stderr.write(str(len(table)) + ' \r')
        sys.stderr.flush()
    fd.close()
    def isidhdr(hdr):
      return hdr == 'message-id' or hdr == 'in-reply-to'
    if 1:
      linktypesidx_isid = map(lambda pair: map(lambda hdr: (addridx[hdr], isidhdr(hdr)),
pair), linktypes)
      for row in table:
        map(lambda (idx, isid): map(lambda addr: addOrInc(count,
                    canonid(addr, isid), idx),
                row[idx]),
            map(lambda hdr: (addridx[hdr],
                 isidhdr(hdr)),
          addrheaders))
        for linktypeidx_isid in linktypesidx_isid:
          map(lambda src: map(lambda dst: addlink(canonid(src, linktypeidx_isid[0][1]),
                    canonid(dst, linktypeidx_isid[1][1])),
                  row[linktypeidx_isid[1][0]]),
              row[linktypeidx_isid[0][0]])
  global allnodes
  allnodes = map(lambda label: graph[label], graph.keys())
  sys.stderr.write(str(len(allnodes)) + ' nodes\n')
  sys.stderr.flush()
  startiterat = time.time()
  for iter in range(iters):
    iterate()
    sum = 0.0
    for key in graph:
      sum += graph[key].score
  sys.stderr.write("iteration took %f s\n" % (time.time() - startiterat))
  sys.stderr.flush()
  printgraph()

if __name__ == "__main__":
  main(sys.argv)
```