A Multi-Agent Referral System for Matchmaking

Leonard N. Foner
MIT Media Lab
20 Ames St, E15-305
Cambridge, MA 02139

617/253-9601

ABSTRACT

Many important and useful applications for software agents require multiple agents on a network that communicate with each other. Such agents must find each other and perform a useful joint computation without having to know about every other such agent on the network. This paper describes a *matchmaker* system, designed to find people with similar interests and introduce them to each other. The matchmaker is designed to introduce *everyone*, unlike conventional Internet media which only allow those who take the time to *speak* in public to be known.

The paper details how the agents that make it up the matchmaking system can function in a decentralized fashion, yet can group themselves into clusters which reflect their users' interests; these clusters are then used to make introductions or allow users to send messages to others who share their interests. The algorithm uses *referrals* from one agent to another in the same fashion that word-of-mouth is used when people are looking for an expert. A prototype of the system has been implemented, and results of its use are presented.

KEYWORDS: agents, collaborative filtering, CSCW, joint computation, ecology of computation, user modeling, intelligent systems, information retrieval, distributed AI, Internet.

INTRODUCTION

Software agents are computer programs which attempt to perform some set of tasks autonomously for their users, in a trustworthy, personalized fashion. They can be either manually programmed by the user, or use techniques from machine learning to discover how the user does some task and gradually automate it. Examples include mail filtering programs, which learn or are told whose mail is valued and whose is not [9][10]; meeting scheduling programs, which learn or are told when and with whom to schedule meetings and how flexible to be in negotiating (with other agents) for times depending on who else is in the meeting [7]; and so forth. Many software agents are even designed to be primarily entertaining, perhaps with ancillary practical or informative goals [3][11].

Other agents take more initiative; they actively inform the user when they find items that match the user's known interests. Often, such agents may not understand the domain of interest directly, but are instead facilitators that can find *other people* who understand the domain better who can advise. *Automated collaborative filtering*, in which users with similar tastes are matched up, is used in systems such as Webhound[9] or HOMR/Ringo [14].

While the two agents above match up users' tastes to make recommendations, their focus is not explicitly to matchmaking users and introducing them to each other. The research described in this paper is focussed on introducing users who are interested in similar topics. There are a number of reasons why one might want to do this:

- People are often working on similar projects without realizing it—be it two people down the hall from each other reinventing the same wheel, or two doctors both doing research on similar cases but having no idea that both of them are studying the same literature.
- It is often the case that people need to find an expert in some field, but finding such an expert can be difficult and time-consuming. Those who are not well "plugged-in" via word of mouth can find this even more difficult.
- There is potential for a great deal of social collaboration on the Internet, but it is often underutilized. "Lurkers" who read but do not post to mailing lists or newsgroups, for example, are an undiscovered resource to the community, invisible because they do not contribute to public discussion.

Current communications systems on the Internet are not well-designed for this sort of matchmaking. In almost all media on the Internet, only people who take the time to write a piece of prose and transmit it somewhere, whether by mail, news, or making a Web page, are ever seen by anyone else. Two people who are both working on the same problem, or who share an interest, may never know if they themselves are not actually writing about it. The matchmaking system described here is designed to aid these "lurkers" who are not part of the public discussion nonetheless find each other and establish a community.

Why having multi-agent systems helps

Many currently-implemented agents use a *centralized* architecture, in which one agent serves either one or many users. A centralized architecture has its advantages: for example, if there is no effective way for peers to find each other, a centralized solution may be the only workable solution. Unfortunately, there are problems with a centralized architecture:

- Scaling such an architecture to large numbers of users is difficult; in systems which must correlate user interests, for example [14], straightforward approaches to this problem generally require a quadratic-order matching step somewhere.
- If the system requires either high availability (due to constant demand for its services) or high trustability (because it handles potentially sensitive information, such as personal data), a centralized server provides a single point where either accidental failure or deliberate compromise can have catastrophic consequences.

For these reasons and others, many foreseeable future applications for software agents involve large numbers of agents interacting with each other. Users may have a number of agents operating on their behalf, and agents of any particular user may have to communicate with other agents elsewhere on the network in order to share information.

Why multi-agent systems are hard to build

While decentralized, multi-agent systems have several important advantages, one of the largest problems with them is *how agents are supposed to find each other.* Each agent should not have to know about (and, indeed, probably cannot know about) every other agent, user, or resource on the network. Instead, some mechanism by which agents may locate only the useful agents on the network must be arranged.

There are several relatively straightforward approaches that have been used in other networked systems. For example, hierarchical organization of the entities, as is done with resource records in the Internet domain name system [12] or with newsgroup topics in the Usenet [4], can help to reduce the inherently quadratic problem into a logarithmic one. However, such approaches depend on some inherent organizational principle that is established in advance, which is neither always optimal nor always convenient; for example, consider the number of crossposted Usenet articles, a clear indication that single-inheritance hierarchies are not necessarily a good match to the underlying topic space.

This research focuses on the problems of a *matchmaking* service, one designed to find groups of people with similar interests and bring them together to form coalitions and interest groups. We are *not* explicitly interested here in romantic matchmaking between users, for many reasons—the most obvious being that shared interests do *not* necessarily mean that two people are romantically compatible. The intended scale of the matchmaking is that of the entire Internet, an environment in which there are potentially millions of users and millions of agents corresponding to them. The domain and the large number of agents presents difficult coordination problems, such as:

- there is no obvious a priori hierarchy by which to organize the agents (why would any one person's interests be at the top of any hierarchy? how would we know whom to pick, anyway?);
- asking other agents *at random* resembles diffusion in a gas and is extremely slow—it means each agent could be required to ask every agent on the network, guaranteeing a solution that scales poorly; and
- a centralized approach runs into the problems mentioned above of quadratic scaling, and also is subject to single-point-of-failure problems if the central system either fails or is compromised—an important point for an application handling potentially sensitive data.

Finding the right cluster of peer agents: the core idea

To address these problems, this research considers an overall organization which borrows ideas from *computational ecology* [5], in which agents have only local knowledge, but self-organize into larger units. The *core ideas* in the approach taken here are to

- compare the agents' information in a *peer-to-peer, decentralized* fashion,
- use *referrals* from one agent to another and an algorithm resembling *hill-climbing* to find other, more appropriate agents when searching for relevant peers, in order to
- build *clusters* or *clumps* of like-minded agents, and to
- *use these clusters* of similar or like-minded agents (whose users therefore share similar interests) to *introduce* users to each other and enable cluster-wide *messaging* between users whose interests match.
- use a *persistent* agent that runs most of the time, for long periods; the user does not start up the agent, get an immediate result, and shut it down, but instead runs it in the background for hours or weeks, while it uses "word of mouth" to find and join appropriate groups of agents whose users share the same interests.

How the resulting clusters can be used

Once agents have formed clusters—an ongoing and continuous process for real agents on the Internet, due to the scale and constantly-changing environment involved—how can we use these clusters? There are many applications; this is a short summary:

- *Messaging into the group.* A user whose agent is in some particular group can send a message into the group—either those other agents known directly by the user's agent to be in the same cluster, or transitively through all other agents in the cluster by following cluster cache information in a flooding algorithm. Thus, given some particular granule on the user's local agent, the user could ask his agent to send a message to all other agents in the clump of which this granule is a member.
- *Introductions.* The chain of referrals themselves can be useful information, and can be exposed to the user under certain circumstances. Not only can the user send message to particular individuals (whether pseudonymously or not), but the agent itself can facilitate a "flirtatious" sort of introduction in which information can be symmetrically and gradually revealed, via cryptographic protocols. Users could ask for an explicit introduction to particular members of the cluster, or could instruct their agent to accept or solicit introductions when it looked like there was a particularly good match available.

- *Finding an expert.* By using a combination of messaging into the group and introductions, the clusters that a user's agent finds itself in can potentially be used to find experts on the subject, since presumably such experts (if they, too, are running the agent) will have their interests reflected in the clustering. Here, a user could prepare a small piece of prose, or find some existing message, which talks about the subject for which the user wants an expert; the clustering algorithm could then generate a granule for this grain and attempt to find a suitable cluster. Once found, it could start the introduction process to acquaint the questioner and the expert.

What is described in this paper

The following sections describe the algorithm used in a prototype of the clustering system, the testing used to evaluate its performance, and how this work is integrated into the larger goal of automatically building interest groups and coalitions on the Internet.

Note that the algorithms described below are but a small piece of the overall task. In particular, since the system handles sensitive information such as people's interests, fielding the system on the Internet requires cryptographic privacy safeguards briefly described elsewhere [1][2] and which are the subject of current research. Furthermore, as an initial prototype for testing the efficacy of clustering, no user interface is described. The entire system, including such cryptographic safeguards, a user interface, and other necessary elements, is called *Yenta*; to avoid confusion, the prototype piece described here is called *Yenta-Lite* or *YL* for short.

THE APPROACH

The overall goal is to form clusters of agents whose users share similar interests. In order to do this, we must answer the following questions:

- What does it mean to have an interest, and how do agents know about these interests?
- How do we determine similarity of interests?
- How does a particular agent know which other agents to contact?
- How can we form clusters of similar agents?

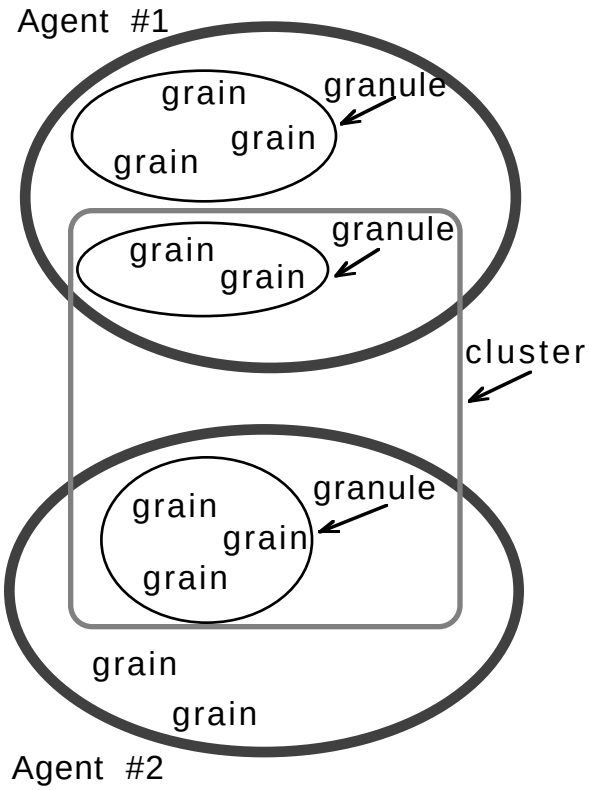
What does it mean for a user to have an interest, and how do we capture that computationally?

For the purposes of matching people by their interests, we assume that these interests are *capturable* in some computer-based form. At the moment, Yenta only deals with text, such as electronic mail messages, the contents of various newsgroup articles, the contents of the user's files in a filesystem, and so forth. The architecture of Yenta supports somewhat different sources of information as well (such as World Wide Web hotlists and homepages)—the crucial requirements for any interest are *a)* they are represented in some electronic form, hence captured by the computer, and *b)* there is some way of comparing two potential interests and assigning a *degree of similarity* between them.

As currently implemented, Yenta-Lite can examine the contents of email messages, newsgroup articles, and user files that the user has received, read, or written. The tests described in this paper used *newsgroup articles* and *email messages* only, as discussed in the section on evaluating Yenta-Lite's performance. Each individual message, article, or file being compared is considered a *document*; however, since Yenta might eventually be comparing nontextual documents, we use the term *grain* to refer to any individual chunk of bits associated with a user.

A user is deemed to *have* an interest if several grains are *similar* to each other. Such a collection of similar grains is called a *granule*. A user may own many granules, each corresponding to some separate interest; for example, a user who regularly reads newsgroup articles on dogs and cars would presumably have two granules reflecting these disparate interests.

Two users, A and B, are deemed to *share* an interest if A has at least one granule that is similar to at least one of B's granule. Two or more users who share an interest are *conceptually* in a *cluster* at the instant that they both possess similar granules; they are *actually* in clump at the instant their two agents discover this similarity. A diagram illustrating this is below.



Suppose we have three users, A, B, and C. Suppose that A and B are in a clump, and B and C are in a clump. Are A, B, and C all in a clump together? Not necessarily. If A is interested in dogs and cars, his associated granules are A_{dogs} and A_{cars}. If the other granules are B_{dogs}, B_{zebras}, C_{dogs}, and $C_{guitars}$, then A, B, and C are all in a clump, because they all share an interest in dogs. However, if C_{dogs} was instead C_{zebras}, then we have two clumps, one reflecting A and B's interests in

dogs, and one reflecting B and C's interest in zebras. B in this case is in two clumps, while A and C are each in one clump.

How do we determine similarity of interests?

The fundamental assumption behind Yenta's assessment user similar of user interests is this: If two users both have several documents which are similar to each other, then the users are assumed to share an interest themselves.

In order to function at all, Yenta demands that any two grains can be compared to yield some measure of similarity. It is also required that this measure be (at least) partially-ordered; a floating-point number, for example, which reflects how similar two grains are is an acceptable representation. The Yenta architecture allows more sophisticated similarities than scalar numbers, but Yenta-Lite, and the results reported here, use only scalars. At the moment, it is also assumed that this comparison operator is reflexive, e.g., that if A's similarity to B is 0.74, then B's similarity to A is likewise 0.74. Future work may explore the stability of the clustering algorithm in the face of nonreflexive comparison operators.

Since Yenta-Lite's grains are all exclusively textual, we use the SMART [15] document system to compare them. SMART is designed primarily to index and retrieve documents from large collections. It has many possible modes of operation; in our use, SMART first stems all words in any given document (e.g., removes prefixes and suffixes and otherwise canonicalizes the text), computes an inverse-frequency metric for each word in the document (so that rare words with greater power to discriminate two documents from each other have greater weight than common words which appear in most documents), and computes a vector which describes the document based on these.

When used to index into a large collection of documents, SMART normally takes a query, computes the vector associated with the query, and dots the resulting query vector with the vectors corresponding to each document. Dot products which have high scores are reported. In Yenta's case, the query is itself a document; therefore, Yenta essentially takes pairs of documents, dots them together, and assumes that high scores indicate similarity.

This is not the only way to do this, of course. For example, consider WordNet [13], which is a semantic net that allows comparing words based on how many links away one word is from another, and in what direction (e.g., synonym, antonym, superset, etc). Future implementations of Yenta may combine SMART and WordNet if the advantages (e.g., possibly more resilience in the face of synonyms that rarely co-occur in a single document) outweigh the disadvantages (e.g., greater semantic "fuzz" in the comparison due to the greater number of words investigated in any given document).

Forming clusters via referrals

We now come to the heart of the clustering algorithm. Given that we have a multiplicity of agents with no central node and no hierarchy, how can we reasonably form clusters which reflect the interests of the users?

The major steps (described in more detail) are:

- Intra-agent initialization, known as *preclustering*: Combine grains into granules within a single agent.
- Inter-agent initialization, known as *bootstrapping*: Find at least one other agent with which to communicate.
- Walk referrals and cluster: Form clusters of like-minded agents.

Preclustering

When an agent first starts running, it must determine what interests its user possesses. It does this by collecting some subset of the user's email, newsgroup articles, and files; each such item is known as a *grain*. Each separate grain is considered for membership in a growing collection of granules.

First, each grain is converted into a SMART vector. Next, a complete cross-product table is created in which each grain's SMART vector is dotted with each other grain's SMART vector; each resulting dot-product p is an entry in the table. This is an $O(n^2)$ operation, given that there are n grains in the user's collection. The result is a table in upper-triangular form, with the main diagonal suppressed (since the main diagonal corresponds to comparing each grain to itself). We then compute the mean, $\bar{p}$, and the standard deviation σ of all of the *nonzero* entries in this table of p values. Typically, 60% of the entries in the table are zero.

Next, a grain is picked at random to start the process of preclustering into granules. It is assigned to the first granule, G_0. To grow G_0, we pick a grain g not already in G_0 and compare it—by dotting SMART vectors together—to each grain already in G_0; we compute the mean $\bar{g}$ of these dot products. We repeat this process for all the other grains not in G_0 and remember $\bar{g}_{best}$, which is the best mean. Then, we see if

$$\bar{g}_{best} > (W\sigma) + \bar{g}$$

is true, where W is a weight described in the next paragraph. If the relation is true, then the grain corresponding to $\bar{g}_{best}$ is added to G_0. When we have made a complete pass through all documents not in G_0, we take a document at random in the leftovers and start trying to make granule G_1.

The weight W is essentially a user-tunable variable. $W = 1$ implies that roughly 17% of the grains will pass this test when compared to a randomly selected granule, since a weight of 1 corresponds to everything on the high side of one standard deviation from the mean; that is:

$$\frac{100\% - 67\%}{2} = 17\%.$$

This process of producing granules is relatively time-consuming (it has several $O(n^2)$ steps in it), but must be done only once for any given collection of the user's grains, and, as shown later, it appears to produce acceptable results.

In true Yenta, it is assumed that the user will constantly be adding grains to his collection as new messages come in or new files are created; however, incrementalizing the algorithm to cope with each added grain is relatively easy: we compare each new grain with existing granules for membership, adding it if it matches well. Otherwise, it is put aside with the rest of the unmatched grains, and this pile of unmatched grains is occasionally scanned to see if a large

enough number of grains are similar that they can form a new granule.

Boostrapping

The next phase requires finding at least one other agent with which to communicate; finding more after that is easier—due to other agents' rumor caches—in that it is less likely that we will require either ad-hoc heuristics or user intervention. In Yenta-Lite, we finess this problem and assume that we can always find another agent. Several heuristics are availabie for true Yenta, including broadcasts and directed multicasts on local network segments to find other agents in the same organization, asking a central registry which contains a *partial* list of other known agents, and asking the user for suggestions. All of these heuristics have various advantages and disadvantages, but we shall not pursue them here.

Data structures used in finding referrals and clusters

We now come to the step in which the various granules in agents form clusters with other granules. For concreteness, assume that we have two agents, named A and B, which each have a few granules in them, e.g., G_{A0}, G_{A1}, etc. Each agent also contains several other data structures:

- A *cluster cache, CC*, which contains the names of all other agents currently known by some particular agent as being in the same cluster. Thus, if agent A knows that its granule 1 is similar to granule 3 of agent B, then CC_A contains a notation linking G_{A1} to G_{B3}. There are two important limits to the storage consumed by such caches: g_l ("local granules"), the number of separate granules that any given agent is willing to remember about itself; and g_r ("remote granules"), the number of granules this agent is willing to remember about other agents. The total size of *CC* is hence g_l times g_r. In Yenta-Lite, these are essentially unbounded; in an implementation that wishes to save space, limiting g_r before limiting g_l would seem to make the most sense, as this limits the total number of other agents that will be remembered by the local agent, while not limiting the total number of disparate interests belonging to the user that may be remembered by the local agent.
- A *rumor cache, RC*, which contains the names and other information (described below) from the last *r* agents that this agent has communicated with. In Yenta-Lite, *r* is arbitrarily set to 5, and it should definitely be bounded in true Yenta as well, since otherwise any given agent will remember *all* of the agents it has ever encountered on the net and its storage consumption will grow without bound. Reasonable values for bounds in real-life operation with large numbers of agents are currently unknown, but are suspected to be on the order of 20 to 100.
- A *pending-contact* list, *PC*, which is a priority-ordered list of other agents that have been discovered but which the local agent has not yet contacted.

The rumor cache contains more than just the names of other agents encountered on the network. It also contains some subset, perhaps complete, of the text of each granule corresponding to those agents.

The stored granules themselves are essential for the referral process. Having the complete text of each granule, or even most of it, could represent a large amount of storage (e.g., 100K or more per granule, depending on exactly what is in any given granule). We do not just store the SMART vectors because:

- The Yenta architecture does not *require* that the comparison operator be able to handle a "reduced information" representation of the two grains to be compared. SMART happens to compare two documents by reducing them to a pair of vectors before dotting them together, but other comparison operators might not produce such a compact representation as part of their operation.
- The Yenta architecture does not enforce a requirement that each agent be running identical software, and indeed expects that any given pair of agents may be running slightly different versions, including different comparison functions. There is no telling a priori whether some reduced-information representation of a particular grain will be correct for two different comparison operators.

  Note that one might allow the user to choose a reduced-information version of each granule, accepting the reduced performance that would result when other agents give up on interoperating with the local agent when they discover that their comparison operators differ.
- Having the complete text of each granule represents more than a space penalty—it also represents a serious privacy problem if some particular agent were to be maliciously modified to disgorge both the contents and identity of some remote agent. In true Yenta (but not Yenta-Lite), this is ameliorated using cryptographic protocols to hide information, even in the cache, and also to hide identities of the remote agents.

Getting referrals and doing clustering

Now that we have all this mechanism in place, performing referrals and clustering is relatively uncomplicated.

The process starts when some agent (call it A) has finished preclustering and has found at least one other agent (call it B) via bootstrapping. Agent A then performs a comparison of its local granules with those of agent B, using a process reminiscent of the preclustering phase but simplified. A builds an upper-triangular matrix describing the similarities between each of its local granules and those locally held by B. Then, rather than taking averages and standard deviations, it simply finds the highest score (e.g., closest similarity) between any given granule (say, G_{A1}) and B's granules. If there is no such value above a particular threshold, then the local granule under consideration does not match any of B's granules, although some other local granule, e.g., G_{A2}, might match.

The comparison process is simpler in the clustering (inter-agent) phase than in the preclustering (intra-agent) phase in part because two agents talking to each other cannot assume that they have complete information about either each other or the space of all possible other granules on the network. Thus, we do not bother trying to calculate averages and standard deviations; as observed in the prototype, a simpler,

threshold-based match appears to work well enough.

When we are done comparing granules from A with granules from B, agent A may have found some acceptably close matches. Such matches are entered, one pair of granules at a time, in A's cluster cache. B is likewise doing a comparison of its granules with A and is entering items in its own cluster cache.

Whether or not any matches were found that were good enough to justify entering them in a cluster cache, the next step is to acquire *referrals* to agents that might be better matches. In the example here, agent A asks agent B for the entire contents of its rumor cache, and runs the same sort of comparison on those contents that it did on agent B's own local granules. Good matches are added to A's cluster cache, the rest of the data is added to A's rumor cache, and A's namelist is updated by adding to it those other agents which showed good matches to A, that is, those agents which had granules that went into A's cluster cache. These agents will be contacted next, after A finishes with B and any other entries in its namelist. The various caches belonging to B that A has been consulting were gathered by B in a similar way; every agent participating in this protocol is thus building up a collection of data for its own use and for the use of other agents.

This procedure acts somewhat like human word of mouth. If Sally asks Joe, "What should I look for in a new stereo?" Joe may respond, "I have no idea, but Alyson was talking to me recently about stereos and may know better." In effect, this has put Alyson into Sally's rumor cache (and, if Joe could quote something Alyson said that Sally found appropriate, perhaps into Sally's cluster cache as well). Sally now repeats the process with Alyson, essentially hill-climbing her way towards someone with the expertise to answer her question.

EXPERIMENTAL EVALUATION OF THE ALGORITHM

To test the algorithm presented above, the Yenta-Lite prototype was implemented. This prototype contains simulates 20 agents by running them all on a single machine.

A randomly-chosen mix of newsgroups and mailing list archives, comprising 13 megabytes total from 7 sources, were used as the grain data for the agents. In particular, the sources were comp.ai.philosophy, rec.pyrotechnics, and sci.math (Usenet newsgroups), and alive-archive, macmoose-archive, physics-archive, and subgenius-archive (two mailing lists about programming projects, an announcement list for events of interest to physicists, and an aggressively eclectic mailing list for members of the Church of the SubGenius).

Each of the 7 sources was subdivided into smaller files, each no more than 150-200K, yielding 64 smaller files total. Thus, comp.ai.philosophy was divided into 20 small files, alive-archive into 3, and so forth. These smaller files were then randomly distributed amongst the 20 agents, such that each agent received either 3 or 4 of them.

Preclustering was run for each of the 20 agents, and the resulting clusters were hand-analyzed to get an idea of what the results were. While preclustering, a grain was deemed interesting enough to create a granule if at least 5% of the other grains available in the given agent also participated in the granule. Thus, grains which formed granules consisting only of themselves or a tiny number of other grains were inhibited.

By way of illustration, consider the two example agents below, which were selected randomly from the 20 total. Agent 1 got two small files from comp.ai.philosophy (the first and second of them, ai.1 and ai.2) and one from sci.math; SMART converted the resulting grains into 180 vectors. Preclustering yielded 12937 nonzero matrix entries, which were 39% of the total entries, and formed 5 different granules (named 1.1 through 1.5). Human analysis of the resulting granules indicated that there was some overlap between the subject areas of the two newsgroups (two granules contained messages from both newsgroups, for example). Agent 6, on the other hand, completely partitioned the SubGenius mailing list from the physics mailing list, and further segmented each of those into two different subject areas.

Agent 1: ai.1, ai.2, sm.1,
180 vectors, $\bar{p}$=.072, σ =.085, NZ=12937 (39%)
1.1 ai/sm. Limits of computing power/theoret. comput.
1.2 ai. Long discourses on fuzzy logic/psychology.
1.3 ai/sm. Philosophy of N,Z, Q, and R construction.
1.4 ai. Books about small towns.
1.5 sm. Division by zero tricks.

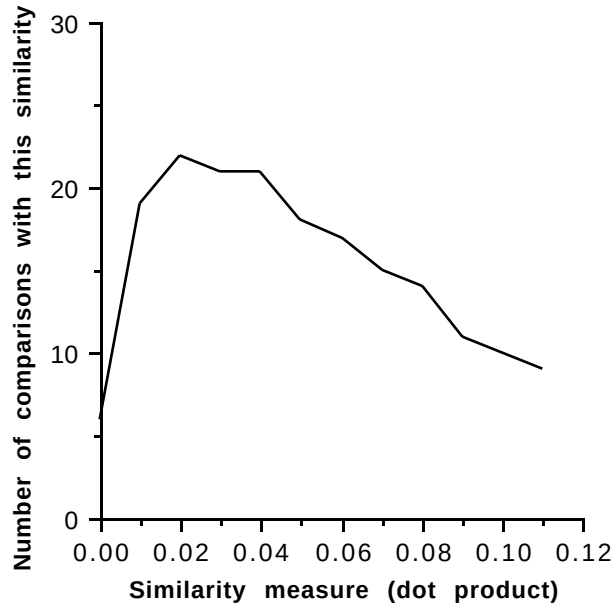
Agent 6: sg.1, rp.1, ph.2
68 vectors, $\bar{p}$=.112, σ =.128, NZ=2131 (46%)
6.1 sg. SubGenius random flaming.
6.2 sg. More SubGenius random stuff; new topic.
6.3 ph. Drivel from the American Physical Society.
6.4 ph. Boston area physics calendar; bad physics poetry.

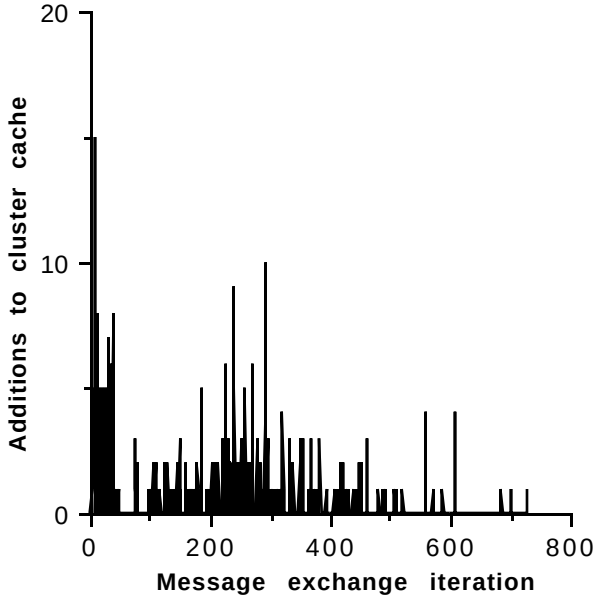
The distribution of similarity scores was somewhat surprising; instead of an expected Gaussian, the curve looked more like a blackbody curve. For example, a randomly-selected result from comparing one particular grain to a set of others, while trying to decide whether to place it into a granule, yielded a curve with a mean $\bar{p}$=.097, σ =.096, and the shape below.

Once preclustering was completed, the agents were run in random order and allowed to exchange messages. The simulation was run "to convergence," meaning that agents were allowed to continue exchanging messages until no additions were made to any agent's cluster cache for hundreds of exchanges—and hence all clusters that were *going* to form *did* form. This is *not* the situation that would obtain with true Yenta on the Internet, both because the sheer number of agents would require a very large number of message exchanges, and because the grains and granules making up each individual agent would be under constant change as their users received or sent additional messages—hence the system could never converge.

A plot of the number of additions made to the Yenta-Lite running at any particular instant vs the message exchange number in the entire simulation appears below.

Convergence was achieved before 800 messages were exchanged between the agents. There was an initial burst in which several agents added a large number of granules to





their cluster caches, followed by a relative lull, followed by a gradual rise and fall in cluster-cache additions. It is not entirely clear what accounts for the lull around the 100th message exchange; it is possible that all the "easy" clustering happened early and each agent then had to build up enough of a rumor cache and do sufficient hill-climbing using it for progress to continue.

Since this is a static simulation, it is possible to ask how the number of messages exchanged during the clustering phase compares to a brute-force solution, in which each agent's granules are methodically compared to every other granule in every other agent. Since the 64 original files turned into 68 total granules, such a crossbar would require 68^2=4624 comparisons if done naively, and 2248 comparisons if one realizes that the upper triangular part of the crossbar matrix, minus the main diagonal, is all that need be computed given a reflexive comparison function. On average, each message exchange by each Yenta-Lite compared 3.4 granules (68/20) at each end of the exchange, so the approximately 750 message exchanges performed 2550 comparisons. This is not much more work than the brute-force solution would have taken, yet it possesses desirable properties that the brute-force crossbar would not:

- The clusters are grown incrementally for each agent, so at any given time, each agent sees at least some of many clusters.
- No agent need retain knowledge of all other granules in the system at any time.
- If a agent were to disappear from the system, the only lasting effect would be for other agents to "forget" it; the rest of the clusters would still form.

Manual inspection of the clusters that resulted from this run show that the brute-force crossbar solution and the referral solution are essentially identical.

RELATED WORK

There are many efforts in distributed AI and multi-agent systems which could be considered relevant; here we consider only other matchmaking systems and related approaches.

A common technique in systems that support computation amongst a group of users is to centralize a server and have its users act like clients. Systems that match user interests to each other, and have such a centralized structure, include Webhound [14] and HOMR/Ringo[9].

Kuokka and Harada [8] describe a system that matches advertisements and requests from users and hence serves as a brokering service. Their system certainly is a matchmaker, but it assumes a centralized matchmaker and a highly-structured representation of user interests.

Others have taken a more distributed approach. For example, Kautz, Milewski, and Selman [6] report work on a prototype system for expertise location in a large company. Their prototype assumes that users can identify who else might be a suitable contact, and use agents to automate the referral-chaining process; they include simulated results showing how the length and accuracy of the resulting referral chains are affected by the number of simulated users and the accuracy and helpfulness of their recommendations. Yenta-Lite differs from this approach in using ubiquitous user data to infer interests, rather than explicitly asking about expertise.

CONCLUSIONS AND FUTURE WORK

Yenta-Lite demonstrates that referral-based matchmaking can provide acceptable results without requiring any one agent to know about all other agents, and without requiring unreasonable messaging traffic or local computation.

Work is currently proceeding on several aspects of the final Yenta design:

- Implementing the requisite privacy safeguards and user interface to permit a networked implementation with real user data.

- Evaluating the suitability and stability of the clustering algorithms in the face of hundreds or thousands of instantiations of the agent in a real environment.
- Experimenting with different comparison metrics to enhance Yenta's ability to accurately determine a match in user interests.

ACKNOWLEDGMENTS

I would like to thank undergraduates Jon Litt for figuring out the intricacies of a large system like SMART and for his sysadmin expertise, and Bayard Wenzel for implementing the initial prototype of these ideas. I would also like to thank my advisor, Dr. Pattie Maes, for her advice and her support of this research. This research has been supported in part by British Telecom.

REFERENCES

[1] Foner, Leonard, "Clustering and Information Sharing in an Ecology of Cooperating Agents, or How to Gossip without Spilling the Beans," *Proceedings of the Conference on Computers, Freedom, and Privacy '95 Student Paper Winner*, Burlingame, CA, 1995.
{http://foner.www.media.mit.edu/people/foner/Reports/CFP-95/paper.ps}

[2] Foner, Leonard, "Clustering and Information Sharing in an Ecology of Cooperating Agents," *AAAI '95 Spring Symposium Workshop Notes on Information Gathering in Distributed, Heterogeneous Environments*, Stanford, CA, 1995.
{http://www.isi.edu/sims/knoblock/sss95/foner.ps}

[3] Foner, Leonard, "What's an Agent, Anyway? A Sociological Case Study," *Agents Memo 93-01*, MIT Media Lab, Cambridge, MA, 1993.
{http://foner.www.media.mit.edu/people/foner/Julia/}

[4] Horton, Mark and Adams, Rick, "Standard for interchange of USENET messages," Internet RFC1036, 1987.
{ftp://ds.internic.net/rfc/rfc1036.txt}

[5] Huberman, B.A., editor, *The Ecology of Computation*, Elsevier Science Publishers B.V., 1988.

[6] Kautz, Henry, Milewski, Al, and Selman Bart, "Agent Amplified Communication," *AAAI '95 Spring Symposium Workshop Notes on Information Gathering in Distributed, Heterogeneous Environments*, Stanford, CA.

[7] Kozierok, Robin, and Maes, Pattie, "A Learning Interface Agent for Meeting Scheduling," *Proceedings of the 1993 International Workshop on Intelligent user Interfaces*, ACM Press, New York, 1993.

[8] Kuokka, Daniel, and Harada, Larry, "Matchmaking for Information Agents," *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI) '95*, 1995.

[9] Lashkari, Yezdi, Metral, Max, and Maes Pattie, "Collaborative Interface Agents," *Proceedings of the Twelfth National Conference on Artificial Intelligence*, MIT Press, Cambridge, MA, 1994.

[10] Maes, Pattie, "Agents that Reduce Work and Information Overload", *Communications of the ACM*, Vol 37 #7, July 1994.

[11] Mauldin, Michael, "ChatterBots, TinyMuds, and the Turing Test: Entering the Loebner Prize Competition," *Proceedings of the Twelfth National Conference on Artificial Intelligence*, MIT Press, Cambridge, MA, 1994.
{http://fuzine.mc.cs.cmu.edu/mlm/aaai94.html}

[12] Mockapetris, Paul, "Domain names - implementation and specification," Internet RFC1034, 1987.
{ftp://ds.internic.net/rfc/rfc1034.txt}

[13] Miller, George, Beckwith, Richard, Fellbaum, Christiane, Gross, Derek, and Miller, Katherine, "Introduction to WordNet: An On-line Lexical Database," Princeton University Technical Report, 1993.

[14] Shardanand, Upendra, and Maes Pattie, "Social Information Filtering: Algorithms for Automating 'Word of Mouth'", *Proceedings of the CHI '95 Conference, 1995.*

[15] Zumoff, Joel, "Users Manual for the SMART Information Retrieval System," Cornell Technical Report 71-95.