TIMESTAMP-BASED CORRELATION MEASURES FOR FINDING HIDDEN GROUPS IN CHAT ROOMS

By

Christopher P. Willmore

A Thesis Submitted to the Graduate

Faculty of Rensselaer Polytechnic Institute

in Partial Fulfillment of the

Requirements for the Degree of

MASTER OF SCIENCE

Major Subject: COMPUTER SCIENCE

Approved:

_____

Mark K. Goldberg, Thesis Adviser

Rensselaer Polytechnic Institute
Troy, New York

April 2008
(For Graduation May 2008)

CONTENTS

LIST OF FIGURES

Acknowledgments

I would like to thank foremost Dr. Mark Goldberg for his direction and guidance as my advisor during my stay at RPI. I would also like to thank Dr. Malik Magdon-Israel, Stephen Kelley, Konstantin Mertsalov, Sean Barnes, and Andrey Sarayev for their general feedback regarding my research, telling me whether what I was looking at had been done before, and generally giving me new ideas; Suzanne Matthews, Akintayo Holder, Ed Levie, and Jamey Lewis for their psychological support; and the faculty and staff of the RPI CS department for maintaining an environment in which research can thrive.

Abstract

This thesis describes a new two-step algorithm for finding hidden groups from chat transcripts, that is, transcripts of communication where the recipient of a message is not known. The algorithm is presented in two steps: calculating a correlation value between every pair of users in the chat transcript, and finding clusters in the weighted undirected graph that results. The inter-user correlation can be calculated in a number of different ways, some of which are accomplished by projecting individual user transcripts into an inner product space. The clustering step uses the existing iterative-scan algorithm, with some new modifications. This approach is found to work under limited conditions.

1. Introduction

Several online communication media have the property that messages emitted in it are not annotated with an explicit recipient. For example, email, instant messaging, and most forms of private correspondence have each message annotated with a recipient. This thesis does not deal with those; since the graph of relationships resulting from this type of message is made explicit, analysis of the resulting groups is made much easier. Here we deal with messsaget ath have no explicit recipient. Examples of this type of communication include most forms of online chat, including IRC; interception of private encrypted correspondence, where the intended recipient cannot be recovered due to the encryption; and stock market activity, wehre individual stock trades may be interpreted as messages emitted into the market.

The problem we wish to solve can, in most cases, be simplified to the following. In a single *observation* there are n *users* $i = 1, \ldots, n$. Each user, over the course of the observation period $[0, T]$, emits k^i messages, the contents of which are ignored, at times $t^i = \{t^i_j\}$, $j = 1, \ldots, k^i$; these messages are assumed to be sorted in chronological order, so that $t^i_1, t^i_2, \ldots$ is an increasing sequence. We call the record of all messages emitted during the observation period the *transcript* of this period.

Two examples of the above type of data might make the purpose of defining it as such clearer. The foremost example, and the primary focus of this thesis, is the sending of messages between users in an IRC channel. Each user corresponds (ignoring nickname changes and other such complicating factors) to one user in the transcript, and each message sent in the channel corresponds to one message in the transcript, with the value of the timestamp in the IRC channel. An example of an IRC client with a conversation in it may be seen in Figure 1.1.

It may make sense to look at an IRC transcript in the form shown in Figure 1.2; here, every user's individual transcript is represented by a line which is given a mark for every message that that user posts, with the horizontal offset of the mark corresponding to the time at which the message was posted. This sort of overview plot is very useful in terms of determining the qualitative properties of a chat. We

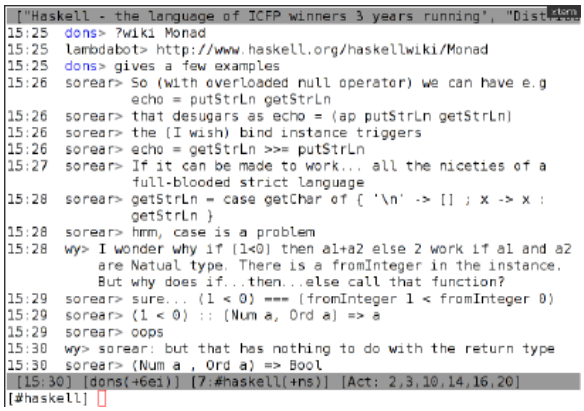

Figure 1.1: An example IRC window. Each line contains a timestamp, username, and message. Messages are input in the blank line on the bottom.

can use it to apply the "eyeball test": if we can't pick out groups of users just by looking at the plot of the transcript, then it's unlikely that a computer program can as well.

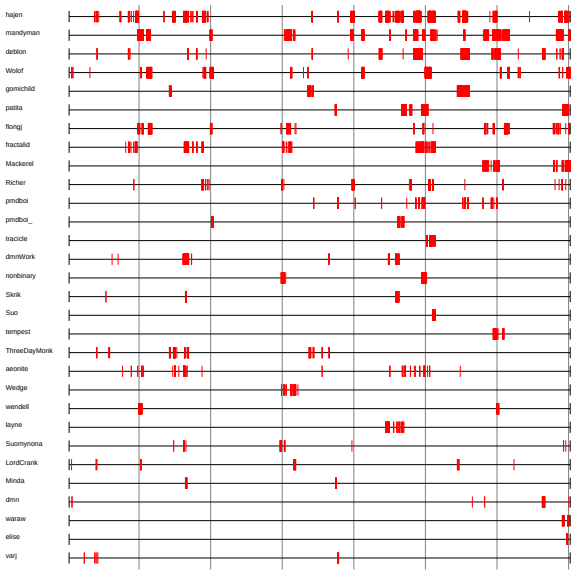

Figure 1.2: A chat transcript, plotted with respect to time. Usernames are on the left.

What we wish to gain is a deeper understanding of the structure that links the users that produce these messages: we want to find hidden groups caused by relationships that, although they cannot be made explicit through the mere emission of messages, can be inferred through the coincident occurrence of messages.

This goal is somewhat uncomfortably vague, so part of this thesis will be devoted to defining precisely what constitutes a hidden group, and how one knows how one has found one (or, at least, how much confidence one can have in the meaning of one's findings).

The method proposed to solve the above problem can be effectively broken into two parts:

1. The transcript is analyzed, and a matrix C_{ij}, measuring the correlation between users a_i and a_j, is produced. This step is detailed in Chapter 3

2. The correlation matrix is given as input to a clustering algorithm, which detects the hidden groups in the original transcript. This step is detailed in Chapter 4.

2. Previous Work

Clustering high-dimensional data in general has a long history, and several different techniques may be employed, including k-means clustering and hierarchical clustering (see [6]). However, most treatments of clustering deal with finding disjoint clusters, and thus categorizing the graph in question by splitting. The hierarchical clustering algorithm ameliorates this somewhat by allowing for selective interpretation of the resulting clustering tree, but it still does not allow for two clusters with both non-trivial union and intersection, as the algorithm to follow does.

The problem of finding hidden groups in online interaction is one that has been dealt with before in this research group; see [2] and [1] for previous uses of the following notion of "cluster density" and the iterative scan algorithm. The addition of the edge probability term to the density measure (see section 4.4) was inspired by conversations with Mark Goldberg ad Stephen Kelley, who had looked at the same measure before.

3. Timestamp Correlation Measures

There are a number of ways to aproach the first part of the chat analysis problem, that is, deriving a correlation matrix C_{ij} from the chat transcripts. Two related but fairly distinct approaches are discussed here: the *inner-product approach*, which calculates the correlation by converting each user's individual transcript into an element of an inner product space, and taking that space's inner product as a basis for the correlation; and the *attention-based approach*, which calculates the correlation between two users' transcripts directly by analyzing the space between individual messages.

3.1 Inner-product-based methods

The inner-product-based approach uses the fact that the inner product has been used as a correlation measure for other problems in the past. The inner product of two unit-length vectors in the inner product space has a natural geometric interpretation, that of the cosine of the angle between the two vectors in that space. As it turns out, the inner product still provides a fairly good correlation metric, in that the clusters it generates (according to the algorithm in the next chapter) are meaningful. This will be discussed more in Chapter 6.

3.1.1 Histogram method

One intuitive approach to a correlation measure is to calculate a histogram by dividing the observation interval $[0, T]$ into several sub-intervals or *bins* $[j\Delta t, (j+1)\Delta t]$ where $\Delta t = T/q$, q is the number of bins, and $j = 0, \ldots, k-1$. Thus from a given user i we arrive at a histogram $h^i = \{h^i_j\}_{j=0}^{k-1}$, where

$$h^i_\ell = |\{t \in t^i \mid \ell\Delta t \leq t < (\ell+1)\Delta t\}|,$$

that is, h^i_j is the number of messages that user i emits during the time interval corresponding to bin j.

In this manner user i's transcript is represented as a q-vector of nonnegative real numbers. We can then naturally apply the Euclidean inner product to get our correlation measures

$$C_{ij} = \langle h^i, h^j \rangle = \sum_{\ell=0}^{q-1} h_\ell^i h_\ell^j.$$

This formulation can be justified intuitively as follows: we only want a given user's activity in the channel to count toward that user's correlation with another user if the other user is also active at the same time. Simply put, if two people aren't talking at the same time, then there's no indication that they're talking to each other. Thus, in the above, if either h_ℓ^i or h_ℓ^j are 0, then we conclude that those users were not talking to each other during the period corresponding to bin ℓ, and nothing during that period counts toward their correlation.

One disadvantage of this method is that the correlation between two users can depend on the manner in which the observation interval is divided, or, in fact, on the starting time of the observation interval itself. For example, consider the conversation in Figure 3.1. The bucket size is the same for both correlation calculations, but the starting time T is offset in the second measurement from the first one by about $\Delta t/2$. This results in the conversations between users 1 and 2 falling in different buckets instead of the same one, and thus results in the correlation between those two users falling to 0. Whether this is justified is debatable, but the dependence on a variable unrelated to the conversation itself (the starting point T) is not.

3.1.2 Kernel method

One approach to remedy this problem is to discard the idea of a discrete histogram of activity, and instead use a more continuous variant. In particular, instead of constructing a q-vector and using the L^2 norm to calculate correlations, we want to construct an ℓ^2-integrable function over time for each user which roughly describes the activity of that user "around" a given point in time. This allows for steady fall-off from the point at which the user actually posts the message, to account for the diminishing relevance of that message to the conversation in the period immediately following.

Specifically, we take a kernel $K(t)$, which has the following properties:

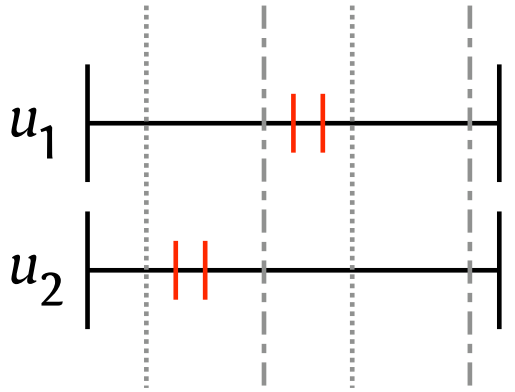

Figure 3.1: An illustration of the difference on the correlation measure that the position of the buckets can make. If the bucket boundary is at the dotted line; these two users have (non-normalized) correlation 4; if it is at the dashed line, they have correlation 0.

- $K(t) = K(-t)$ (the kernel is symmetric)
- $K(t)$ is non-negative for all t.

(In fact, this last restriction might not even be necessary: it might be possible to engineer the kernel to discourage counting message that appear too close together.)

There are several possible choices for a kernel, representing different concepts of attention surrounding a message in the chat, but we will focus on two of them: the square kernel

$$K_{\text{square}}(t) = \frac{1}{2\sigma}\chi_{[-\sigma,\sigma]} = \begin{cases} 1/2\sigma & |t| < \sigma \\ 0 & \text{otherwise} \end{cases}$$

(where $\chi_A(t)$ is the characteristic function equal to 1 when $t \in A$ and 0 elsewhere) and the Gaussian kernel

$$K_{\text{Gaussian}}(t) = \frac{1}{\sqrt{2\pi}\sigma} e^{-t^2/2\sigma^2}.$$

Both of these kernels have a parameter σ, which may be interpreted as the *bandwidth* of the kernel; a higher value of σ leads to messages which appear further apart in

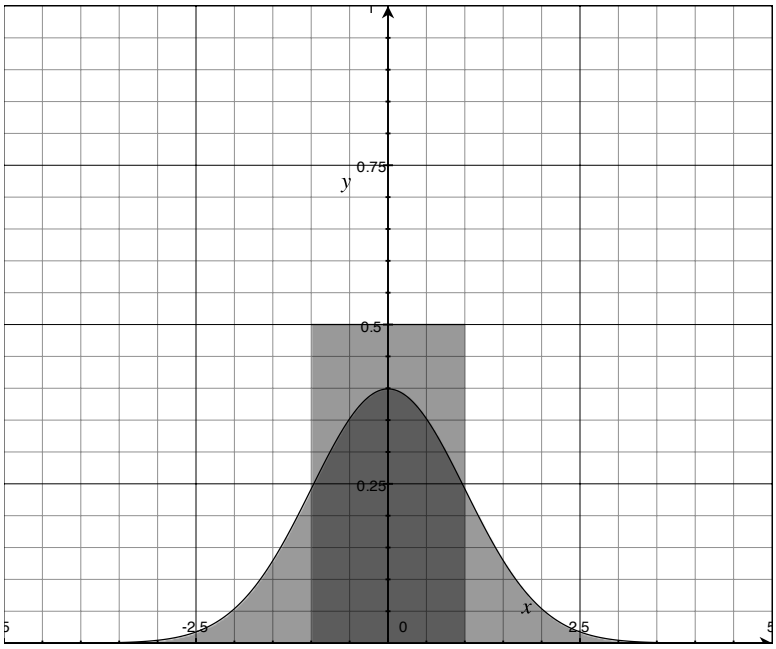

Figure 3.2: The square and Gaussian kernels, with $\sigma = 1$ for each.

the transcript to exert more influence on the correlation between the users involved. Both of these kernels are illustrated in Figure 3.2.

Once we have chosen a kernel, we may construct a function $f_i : \mathbb{R} \to \mathbb{R}$ representing user i as the sum of several displaced kernels, one centered on each message in user i's transcript. In particular,

$$f^i(t) = \sum_{j=1}^{k^i} K(t - t^i_j),$$

for a chosen kernel K. An example of this sort of function can be found in Figure 3.3, where the function f^i has been constructed by centering a Gaussian kernel on the timestamps of messages occurring at times $t = 0, 2, 5$.

Then we may define the correlation between two users as the ℓ^2 inner product between their respective functions:

$$C_{ij} = \langle f^i, f^j \rangle = \int_{-\infty}^{\infty} f^i(t) f^j(t)\, dt.$$

It is interesting to note that the above integral may be calculated exactly in

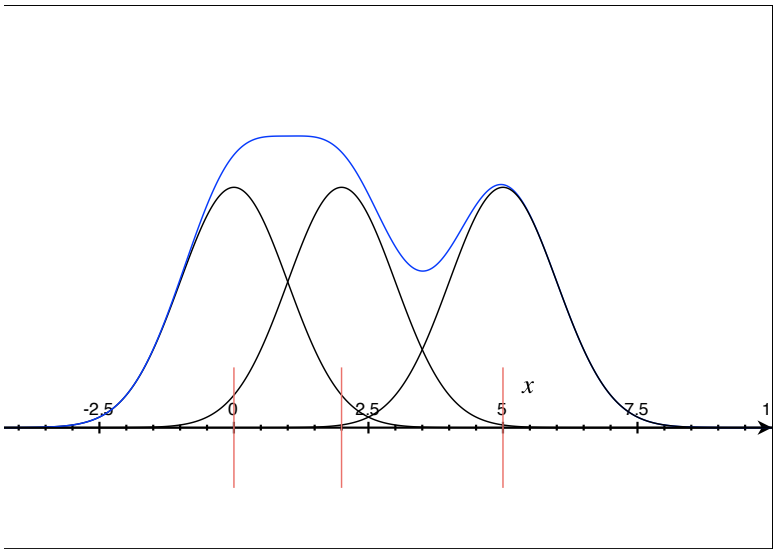

Figure 3.3: An example of a function formed by adding together several Gaussian kernels.

finite time by observing that

$$\begin{aligned}
C_{ij} &= \int_{-\infty}^{\infty} f^i(t) f^j(t)\ dt \\
&= \int_{-\infty}^{\infty} \left(\sum_{a=1}^{k^i} K(t - t_a^i) \right) \left(\sum_{b=1}^{k^j} K(t - t_b^j) \right)\ dt \\
&= \sum_{a=1}^{k^i} \sum_{b=1}^{k^j} \int_{-\infty}^{\infty} K(t - t_a^i) K(t - t_b^j)\ dt \\
&= \sum_{a=1}^{k^i} \sum_{b=1}^{k^j} \int_{-\infty}^{\infty} K(\tau) K(\tau + (t_a^i - t_b^j))\ d\tau \\
&= \sum_{a=1}^{k^i} \sum_{b=1}^{k^j} \kappa(t_a^i - t_b^j)
\end{aligned}$$

where we define

$$\kappa(\Delta t) \equiv \int_{-\infty}^{\infty} K(\tau) K(\tau + \Delta t)\ d\tau.$$

As long as we can compute $\kappa(\Delta t)$ exactly, we can compute C_{ij} exactly as well. Fortunately, for the kernels that we've looked at, this integral can be evaluated exactly:

Kernel	$K(t)$	$\kappa(\Delta t)$
Square	$\chi_{[-\sigma,\sigma]}(t)/2\sigma$	$\begin{cases} (2\sigma - \Delta t)/4\sigma^2 & \lvert\Delta t\rvert < 2\sigma \\ 0 & \text{otherwise} \end{cases}$
Gaussian	$\frac{1}{\sqrt{2\pi}\sigma} e^{-t^2/2\sigma^2}$	$\frac{1}{2\sqrt{\pi}\sigma} e^{-\Delta t^2/4\sigma^2}$

In any case, clearly these methods no longer suffer from the arbitrary discretization that afflicts the histogram method. For this reason (and also for the sake of elegance), most of the experiments run in this paper use the Gaussian kernel method to calculate correlations between users.

3.1.3 Normalization in inner product methods

Both inner product methods described above compute a number which may be meaningfully interpreted as the amount of shared activity between two given uers in a chat. However, the values returned here may not match what the experimenter is seeking in the chat. In particular, the methods described above "reward" users who simply have a lot of activity in the chatroom, since the correlation that a given user has with another user is directly proportional to the amount of messages that the first user posts at the same time the second user is active. Depending on what one wants, this could weight the correlation values unfairly to users who simply tend to post more messages, regardless of the recipient, for any number of reasons (likes to break sentences up among multiple lines, is generally talkative, etc.).

In order to remedy this, we can introduce the concept of *normalization*; that is, we interpret the correlation values we get in the previous step as a sort of tentative correlation $\tilde{C}_{ij}$, and then we normalize these values in such a manner that the true correlation value between any two users with identical transcripts (i.e., the true correlation value of a user with himself) is 1. We do this by defining

$$C_{ij} = \frac{\tilde{C}_{ij}}{\sqrt{\tilde{C}_{ii}\tilde{C}_{jj}}}.$$

Note the resemblance to the classic equation derived from the Law of Cosines,

$$\cos\theta = \frac{\mathbf{u}\cdot\mathbf{v}}{\|\mathbf{u}\|\|\mathbf{v}\|}.$$

Going back to the inner product interpretation of the correlation, C_{ij} is now simply the cosine of the angle between the representations of i and j in the corresponding inner product space ($\mathbb{R}^q$ for the histogram method, $C^2(\mathbb{R})$ for the kernel method). This makes sense, from the perspective of measuring the sameness between two users' transcripts: we're simply projecting those transcripts in a continuous manner into high-dimensional space and measuring the angle between them in that space. The smaller the angle (and higher the cosine), the more similar those two users are.

3.1.4 Adaptation of distance metric

Given all the above talk about inner products and inner product spaces, one might wonder if it wouldn't be more straightforward to look at the distance between these transcripts after being projected into the higher-dimensional space rather than the inner product; after all, it would seem to be a more straightforward measure of similarity between users. In fact, since we're dealing with an inner product space, distance can be expressed in terms of vectors:

$$d(\mathbf{u},\mathbf{v})^2 = \|\mathbf{v}-\mathbf{u}\|^2 = \|\mathbf{v}\|^2 + \|\mathbf{u}\|^2 - 2\mathbf{u}\cdot\mathbf{v}.$$

If we first normalize $\mathbf{u}$ and $\mathbf{v}$, this becomes $2(1-\cos\theta)$, and so maximizing the normalized inner product measure (i.e., the cosine measure discussed above) is equivalent to minimizing the normalized distance.

3.2 Attention-based methods

An alternate way to measure the correlation between users is to take the following approach. Let us define the *attention span* of a user to be the period of length Δt following every message that that user posts, where Δt is a programmer-definable parameter (like the previous parameter σ, a larger Δt results in further-apart mes-

sages contributing to the correlation between two users). Call this attention span

$$A^i = \left(\bigcup_{j=1}^{k^i} [t_j^i, t_j^i + \Delta t] \right) \cap [0, T].$$

Then the correlation between two users is simply the measure of the intersection of their attention spans,

$$C_{ij} = \mu(A^i \cap A^j)$$

(where the set measure μ is defined in the usual manner with $\mu([a, b]) = b - a$ and $\mu(U \cup V) = \mu(U) + \mu(V)$ if U and V are disjoint closed subsets of $\mathbb{R}$). This can be intuitively justified as the amount of time that both users can be expected to spend looking at the activity in the chat room at the same time. This model can be further refined by first preprocessing each A^i by removing the connected intervals it contains which have measure less than a given critical length $\lambda > \Delta t$. This can be justified by asserting that, when a user posts an isolated message in the channel, he may just be visiting the channel momentarily without actually paying attention to the conversation.

This measure is fairly easy to implement with a scanning algorithm which merges intervals and then figures out the magnitude of two users' attention spans' intersections.

3.3 Incorporating message contents

The above correlation measures rely solely on the time at which the message was posted and the identity of the user who posted it. However, in real-life chats, there are very often conversational cues that give hints as to the intended recipient of a message. These include:

- Directly addressing the recipient in the beginning of a message
- Immediately following a message in the form of a question with a message in the form of a short answer ("yes" or "no")

These conversational cues can be incorporated by adding a "confidence factor" to the relevant correlation measure; for example, in the Gaussian correlation, if we are comparing users i and j and we conclude that the message sent at timestamp t_ℓ^i was defintely addressed to user j, we can elect to use $\kappa(0)$ instead of $\kappa(t_\ell^i - t_m^j)$, where t_m^j is the time of the closest message that user j posts to the message that user i posts at t_ℓ^i (i.e., we treat this pair of messages as if they actually happeend at the same instance.) Alternatively, we can selectively boost the amplitudes of the kernels when constructing the function for the kernel-based approach; i.e., if we are calculating the (non-normalized) correlation between users i and j, and we conclude that the message posted by user i at time t_ℓ^i was meant to reach user j, then we temporarily assign some confidence $c_\ell^i > 1$ to that message (with the default confidence being 1). The function for user i (at least, while calculating the correlation with user j) becomes

$$f^i(t) = \sum_{\ell=1}^{k^i} c_\ell^i K(t - t_\ell^i).$$

3.4 Which correlation measure should I use?

Given the sheer number of options one has for calculating the correlation between two users, it can be daunting to choose the right correlation measure for a given application. Unfortunately, the author has found that the only reliable way to determine the correct method is through experiment, in particular, combining the correlation measure with the clustering which is to be described in the next chapter. The optimal correlation measure can be found by measuring its effectiveness using the algorithm correctness measure in Chapter 5 and iteratively refining the parameters to the measure. Chapter 6 details the author's findings in this area.

4. Clustering Methods

One of the requirements set forth for finding hidden groups from transcripts is that the groups that are found must be allowed to overlap. This is because groups that are formed in real-life contexts, especially social ones, are not mutually exclusive, and in fact often overlap. Consider, for example, the set of people in an online programming community who program in C, versus the set of people who program in Perl. While there are bound to be members of one community who do not belong to the other, and vice versa, the groups will have overlap between them. In addition, in a given transcript there may be users who don't belong to any group of interest; that is, in the case of IRC, they were people who appeared in the channel but didn't have any meaningful interaction with other people online.

Due to these restrictions, most clustering algorithms available at the moment (for example, hierarchical clustering, Bayesian clustering, and k-means clustering) are unfit to solve this problem, due to the fact that the clusters they produce are (1) disjoint, and (2) comprehensive, that is, they include all samples in the data set in exactly one of the resulting clusters. It is perhaps possible to modify the algorithms mentioned above to allow for overlapping data structures.

4.1 Basic Overlapping Cluster Algorithm

We have chosen to use an iterative clustering method originally developed by Goldberg and Magdon-Israel for the purpose of analyzing relationships formed in email databases and LiveJournal communities, where the recipient of a given communication (email/blog post, respectively) is known, unlike in IRC. The algorithm is an iterative one, and unfortunately inexact, but in practice it has proven itself to find meaningful groups in both the LiveJournal groups and in the IRC groups that are reported in this paper.

This algorithm iteratively refines a candidate cluster S so that the *density* of the cluster, defined as

$$d(S) = \frac{E_{\text{int}}(S)}{E_{\text{int}}(S) + E_{\text{ext}}(S)}$$

is maximized, where

$$E_{\text{int}}(S) = \sum_{\substack{i,j \in S \\ i<j}} C_{ij} \quad \text{and} \quad E_{\text{ext}}(S) = \sum_{\substack{i \in S \\ j \notin S}} C_{ij}$$

Here we refer to $E_{\text{int}}(S)$ as the *internal energy* of the cluster S and $E_{\text{ext}}(S)$ as the *external energy* of S; roughly, if we interpret the correlation measure held in C_{ij} as the amount of activity that occurs between users i and j, then the internal energy corresponds to the amount of activity that takes place exclusively within S, and the external energy E_{ext} corresponds to the amount of energy that takes place between S and users outside S. Then the objective becomes clear: $d(S)$ is simply the fraction of activity that S is involved with which occurs exclusively within S, and we wish to maximize this fraction — that is, we wish to find a subset of users which is involved, as much as possible, only with itself.

The algorithm follows naturally from this concept. Essentially, we take some initial set S and repeatedly examine different users $i \in U$ — the order of examination may be taken randomly, in increasing order of associated activity, etc. If $i \notin S$ and $d(S \cup \{i\}) > d(S)$, then i is added to S; otherwise, if $i \in S$ and $d(S - \{i\}) > d(S)$, then i is removed from S. In this manner users are added and removed from S, with each modification of S only increasing its density.

The initial clusters for consideration may be generated in a number of ways, but the manner used in this work is known as *link aggregage*. The users in the transcript are sorted in order of increasing sum of weight of adjacent edges (i.e., increasing graph connectivity). The two users on either end of the heaviest edge in the graph are considered to be a potential cluster by themselves, and then each user, in sorted order, is checked against each potential vertex to see whether adding it to that vertex improves its density (in this sense, it's similar to the main iterative algorithm). If it is found that that user cannot improve the density of any candidate clusters, it itself is added as a candidate cluster.

4.2 Necessary Conditions for Cluster Merging

The algorithm described above works fairly well for clusters that are well-delimited, that is, clusters for which the correlation between members of distinct clusters is small. We wish to know at what level of correlation between two groups we will see merging of those two groups as perceived by the above algorithm. In other words, how strong must the relationship between two groups of users U and V be before the algorithm sees them as a single group?

Let $m = |U|$ and $n = |V|$, and, for the sake of simplicity, let $C_{ij} = 1$ for $i, j \in U$ or $i, j \in V$ (i and j belong to the same group) and $C_{ij} = \alpha$ if $i \in U$ and $j \in V$ or vice-versa, where $0 < \alpha < 1$. (This sort of relationship comes about when two disjoint groups in a chat have overlapping activity and a normalized correlation measure is used.) Suppose also that the current state of the above clustering algorithm has $S = U$ and is currently examining the possibility of adding a user from V, say i, to S. Note that, since there are $m(m-1)/2$ internal edges and mn external edges,

$$E_{\text{int}}(S) = m(m-1)/2 \quad \text{and} \quad E_{\text{ext}}(S) = \alpha mn,$$

so

$$d(S) = \frac{m(m-1)/2}{m(m-1)/2 + \alpha mn}.$$

If we were to add i to S, resulting in the cluster $S' = S \cup \{i\}$, we would have

$$E_{\text{int}}(S') = m(m-1)/2 + \alpha m \quad \text{and} \quad E_{\text{ext}}(S') = \alpha m(n-1) + n - 1,$$

so

$$d(S') = \frac{m(m-1)/2 + \alpha m}{m(m-1)/2 + \alpha mn + n - 1}.$$

We want to know under what circumstances $d(S')$ is greater than $d(S)$. Using Maxima [4], we find that the critical value of α above which this is true is

$$\alpha^* = \frac{\sqrt{9m^2 - 10m + 1} - m + 1}{4m}.$$

Note that, surprisingly, this is independent of n. Also, $\lim_{m\to\infty} \alpha^* = 1/2$, and it

approaches this limit fairly quickly: $\alpha^* \approx 0.46$ when $m = 5$.

This means that if two disjoint groups of users have internal correlation of 1 and an inter-group correlation of 1/2 or greater, the clustering algorithm will treat these two groups as a single cluster rather than distinct clusters. The critical value of 1/2 provides a convenient metric for estimating how well a given chat correlation measure matches with our real-life concept of a cluster. In this case, using any of the normalized correlation methods (Gaussian or histogram) results in two groups with roughly the same histogram within themselves being merged if the amount of time that they spend talking to each other is at least half the time either of them spends in the chat room.

The above argument also applies to arbitrary chat correlation methods where the intra-group chat is not guaranteed to be close to 1 (for example, the non-normalized Gaussian and histogram methods) by considering α to instead be the ratio between the inter-group correlation and the intra-group correlation. For example, according to the above argument, if the intra-group correlation for a given conversation is 8 and the inter-group correlation is 5, then the ratio between the two is $5/8 > 1/2$, and the groups will most likely be merged by the clustering algorithm.

4.3 Modification for Minimal Clusters

The above shows that the clustering algorithm, under ideal circumstances, will find meaningful clusters given a reasonable correlation matrix. However, correlations derived from real-life situations are considerably hairier simply because real-life chats are noisy. This noise exists in two different respects: first, users that would be completely inactive in a non-noisy environment sometimes post messages that cannot be explained by that user's membership in any groups; and second, the correlation between members within a single group is not constant, but rather varies according to the distribution of messages of that particular user, *not* the expected distribution of messages of a user of that group. The magnitude of this variation is small for well-chosen correlation parameters (for example, sufficiently large buckets in the histogram method), but it still makes some of the theorems proven above significantly weaker in that they are only now "approximately true."

4.4 Edge Probability Term

One possibility for fixing the merging-clusters problem is to add a term to the density formula to discourage large, yet sparse, clusters. We can do this by considering the ratio of the total weight within the cluster to the total "possible" weight within the cluster; if this value is high, then the cluster is sufficiently dense for our purposes. In particular, we can formulate the density now as

$$d(S) = \frac{E_{\text{int}}(S)}{E_{\text{int}}(S) + E_{\text{ext}}(S)} + \lambda \frac{E_{\text{int}}(S)}{\binom{n}{2}}$$

where n is the number of vertices in S. As it turns out, this helps group recognition under certain conditions.

4.5 Alternate Eigenvalue-based Approach

The sequential nature of the algorithm described above seems strikingly local in nature; if there happens to be some global optimal cluster (other than the cluster which contains all vertices, which is trivially the cluster with globally optimal density), then there is a good chance (especially if the graph is large) that our sequential algorithm may never find it. In addition, we might have a reason to look at large initial clusters, if, for example, we were looking for large, all-encompassing groups that form in a social network (i.e., groups that take up more than half of the community population).

One approach one might take to find globally optimal clusters is derived from a similar approach, known as *normalized min-cut*, used to perform image segmentation. This technique is inspired by a similar technique employed in [5] to perform image segmentation by converting the image into a graph of pixels. The technique essentially takes a linear-algebra relaxation approach to membership in the cluster. Let $S \subseteq U$ be a potential cluster of users (for now, just any arbitrary subset of S). Let

$$x_i = \begin{cases} 1 & i \in S \\ 0 & i \notin S. \end{cases}$$

We now attempt to represent the density of S in terms of matrix-vector multiplica-

tion. Let C be the matrix with entries C_{ij} (and diagonal entries zero), and let D be the diagonal matrix with diagonal entries

$$D_{ii} = \sum_{i \neq j} C_{ij}.$$

In other words, D_{ii} is the connectivity of user i with the rest of the graph. We can then express the energy terms used in the calculation of the cluster density of S:

$$E_{\text{int}}(S) = \sum_{\substack{i,j \in S \\ i<j}} C_{ij} = \tfrac{1}{2}\mathbf{x}^T C x, \qquad E_{\text{ext}}(S) = \sum_{\substack{i \in S \\ j \notin S}} C_{ij} = (\mathbf{1} - \mathbf{x})^T C \mathbf{x} = \mathbf{x}^T (D - C)\mathbf{x}$$

where $\mathbf{1}$ is the n-vector $(1, 1, \ldots, 1)^T$. Thus, we may express the density of S as

$$d(S) = \frac{E_{\text{int}}(S)}{E_{\text{int}}(S) + E_{\text{ext}}(S)} = \frac{\mathbf{x}^T(\frac{1}{2}C)\mathbf{x}}{\mathbf{x}^T(D - \frac{1}{2}C)\mathbf{x}}.$$

Let us relax the constraint that each x_i must be either 0 or 1. Then we can take the gradient of the density with respect to $\mathbf{x}$, and obtain

$$\nabla d(S) = \frac{[\mathbf{x}^T(D - \frac{1}{2}C)\mathbf{x}]C\mathbf{x} - [\mathbf{x}^T(\frac{1}{2}C)\mathbf{x}]2(D - \frac{1}{2}C)\mathbf{x}}{[\mathbf{x}^T(D - \frac{1}{2}C)\mathbf{x}]^2.}$$

We wish to maximize the density; in this continuous variant, a necessary condition for a local density maximum is having $\nabla d(S) = 0$. Setting the above expression to zero and rearranging terms, we arrive at

$$(D - \tfrac{1}{2}C)^{-1}(\tfrac{1}{2}C)\mathbf{x} = \frac{\mathbf{x}^T(\frac{1}{2}C)\mathbf{x}}{\mathbf{x}^T(D - \frac{1}{2}C)\mathbf{x}}\mathbf{x} = d(S)\mathbf{x}.$$

Note that the coefficient in the right-hand side is a scalar; therefore, $\mathbf{x}$ must be an eigenvector of $(D - \frac{1}{2}C)^{-1}(\frac{1}{2}C)$. Note that the largest eigenvector of this system will trivially be $\mathbf{1}$ (one can see this by noting that $C\mathbf{1} = D\mathbf{1}$), so we won't get any useful information until reaching the second-largest eigenvector. The corresponding eigenvalue, as it turns out, is the density of the cluster found.

Therefore, one approach to find clusters with locally maximal density is to

find the eigenvectors after the largest eigenvector of the matrix $(D - \frac{1}{2}C)^{-1}(\frac{1}{2}C)$, and then take the resulting values of x_i as a cue for what the discrete cluster should be. One approach that has worked in practice is to simply split the element of x_i according to which ones are positive and which are negative, and create candidate clusters from the corresponding indices, i.e.,

$$S_0^+ = \{i \mid x_i > 0\}, \qquad S_0^- = \{i \mid x_i < 0\}.$$

It's worth noting that computing the matrix $(D - \frac{1}{2}C)^{-1}(\frac{1}{2}C)$ is a non-trivial task, since it involves the inversion of a very large matrix. However, modifying the density measure slightly to

$$\tilde{d}(S) = \frac{2E_{\text{int}}(S)}{2E_{\text{int}}(S) + E_{\text{ext}}(S)}$$

and crunching through all the same equations gives the new eigenvector problem

$$D^{-1}C\mathbf{x} = \tilde{d}(S)\mathbf{x}.$$

which is much simpler to work with, since D, being a diagonal matrix, can be inverted trivially by inverting each of its diagonal elements.

5. Measuring Algorithm Effectiveness

One of the fundamental problems with the above approach is that it's difficult to tell just how well a given algorithm performs in terms of finding hidden groups in a chat room. Simply put, there is no ground truth — without conducting some sort of social survey on all the people who appear in the transcript, there's no good way to know how well the groups that the clustering algorithm finds match the groups that exist in real life.

One apprach that we can take to get around this limitation is to generate our own transcripts, with a known ground truth. We can create a realistic transcript with predetermined group structure which we run the clustering algorithm on, and then we can apply a set distance metric (to be described) to quantify just how close to the ground truth the clusters that we end up with are.

5.1 Random Chat Generation

The random chat generation is controlled by several parameters. A "normal" choice for each parameter is given after its description in parentheses.

- T, the duration of the transcript (24 hours).
- n, the number of users (100).
- n_g, the number of groups (5).
- n_u, the number of users per group (10).
- n_c, the number of conversations per group (3). This is roughly the number of times per observation interval (here, day) the group gets together to chat.
- $\ell_{\min}, \ell_{\max}$, the lower and upper bounds on the length of a conversation (30 minutes, 1 hour).
- Δt, the mean time between messages for a single user within a single conversation (15 seconds). Higher Δt equates to a more talkative user.

Note that the explicit structure of the group is not specified in these parameters, although that can be fine-tuned in the actual software. The chat is then generated according to the following procedure.

1. The set of users $U = \{1, \ldots, n\}$ is generated; each user is given a username of the form `user`i, where i is the index of the user.

2. The set of groups $\mathcal{G} = \{G_1, \ldots, G_{n_g}\}$ is generated; each group chooses n_u users from U at random without replacement.

3. Each username is annotated with the names (i.e., indices) of the groups that it belongs to.

4. Each group G_j has assigned to it n_c time intervals of the form $[t_1, t_2]$ where t_1 is chosen uniformly from $[0, T]$ and t_2 is chosen uniformly from $[t_1+\ell_{\min}, t_1+\ell_{\max}]$. The union of these intervals is stored as $\mathcal{I}_j$.

5. The set of intervals
$$I_i = \bigcup_{i=1}^{n_g} \mathcal{I}_j$$
is calculated for each user i; this is the time during which this user is active.

6. For each user i and each interval $[s, e]$ which constitutes I_i, user i emits messages in that time interval according to a Poisson process with mean time between events Δt. This is done by repeatedly incrementing the next message time by a random variable distributed exponentially with parameter $\lambda = 1/\Delta t$.

This procedure is fairly straightforward, and, fortunately, is amenable to customization. For example, if we wish to simulate a social network which has several different group sizes instead of just one, we can alter step 2 trivially to accomodate that. Likewise, we may enforce the presence of one group completely contained within another by altering the group formation step, or even assign different mean times between messages to different gruops (perhaps a more intellectual group's members would spend more time thinking before posting messages, for example). None of these modifications were undertaken in the present research, however.

One modification which is fairly relevant to real-world situations is the addition of *noise* to the transcript. We define noise as messages that a user emits which do not have anything to do with a conversation happening at the time. In a real transcript, this might be someone who posts a message asking whether anyone in the room is "alive" (i.e., at the keyboard and willing to converse), or simply someone reacting to some stimulus outside the chat room (a news story, for example).

In order to simulate noise, we introduce a new parameter Δt_{noise}, the mean time between messages for noise. We then have every user in the chat emit messages in the interval $[0, T]$ according to a Poisson process with mean time between messages Δt_{noise}. A usual value for mean time between noise messages is fifteen minutes, but could be lower according to the signal/noise ratio of the chat room being simulated.

5.2 Set Distance Measure

When these transcripts are generated, the groups formed in step 2 of the algorithm above can be saved to an external file for eventual comparison with the groups that the clustering algorithm finds later. For a while, the accepted technique for determining how closely the clusters that the algorithm finds matched the original groups was to eyeball the output clusters and determine qualitatively whether they matched or not. This may have worked for small numbers of users, but it quickly became infeasible for numbers of users greater than fifty or so. Therefore, we employ an automated way of determining the difference between two different clusterings of the set of users, which we call the *set distance measure*.

Suppose we are given two different clusterings of a set of users U: $\mathcal{S} = \{S_1, \ldots, S_m\}, \mathcal{T} = \{T_1, \ldots, T_n\}$, with $S_1, \ldots, S_m, T_1, \ldots, T_n \subseteq U$. (In our setting, $\mathcal{S}$ is the canonical clustering, the one output alongside the generated chat, and $\mathcal{T}$ is the empirical clustering, the one found using the clustering algorithm.) We wish to determine the minimum number of *edit operations* one has to perform on $\mathcal{S}$ in order to transform it into $\mathcal{T}$, where an edit operation is addition or removal of a single user from one of the clusters.

Let us make the simplifying assumption that $m = n$, i.e., the number of canonical clusters is equal to the number of empirical clusters. (As it turns out, with

our clustering algorithm, $m > n$ more often than not; this will be dealt with shortly.) Then the problem of finding the minimum number of operations needed to transform $\mathcal{S}$ into $\mathcal{T}$ becomes the problem of finding a permutation $\pi : \{1, \ldots, n\} \to \{1, \ldots, n\}$ which minimizes

$$\sum_{i=1}^{n} |S_i \ominus T_{\pi(i)}|,$$

where $\ominus$ is the symmetric difference operation

$$A \ominus B = (A \cup B) - (A \cap B).$$

If we create a matrix M with elements

$$M_{ij} = |S_i \ominus T_j|,$$

then this problem becomes an instance of the linear assignment problem, which is to find a permutation π which minimizes $\sum_{i=1}^{n} M_{i,\pi(i)}$. In my programs I chose to use the Hungarian algorithm ([3]), which is well known, to solve this problem; the implementation was adapted from a Lisp implementation found online.

5.2.1 Compensating for Cluster Merging

If $m \neq n$, then one can add empty clusters to $\mathcal{S}$ or $\mathcal{T}$ until they are the same size, and then run the set distance algorithm described above. However, this solution is a bit of a kludge in that it doesn't really reflect the reason that m might not be equal to n. In particular, the clustering method is prone to merging clusters together, so most likely $m > n$. This can lead to disastrous edit distances in the naive algorithm for what seems like a fairly common phenomenon.

For example, suppose that there are two canonical clusters S_1, S_2, each of which have size 10, and in the course of finding the empirical clusters they are merged into a single cluster T_1 of size 20. Due to the way that the set distance is formulated above, the set distance between these two clusterings is 20: 10 to remove the ten users from the "odd cluster out" of S_1 or S_2, and another 10 to add those ten users to the other cluster to form T_1.

We can remedy this for the case where $n = m - 1$ (i.e., there is one pair of clusters which has merged) by considering all pairs of input clusters S_{i_1}, S_{i_2}, and taking the minimum set distance over all such pairs between $\mathcal{S}'$ and $\mathcal{T}$, where $\mathcal{S}'$ is $\mathcal{S}$ with S_{i_1} and S_{i_2} removed and replaced with $S_{i_1} \cup S_{i_2}$.

We could take this further to arbitrary $m > n$ by considering all possible condensations of $\mathcal{S}$ into a merged clustering $\mathcal{S}'$ of size n, but there are $S(m, n)$ such ways to do so, where $S(m, n)$ is a Stirling number of the second kind; this can get big. Instead, we take a greedy approach: we find the pair of clusters in the canonical clustering that, when merged, result in the greatest decrease in set distance. We repeat this operation (hence the greediness) until $m = n$, and then we call the given cluster matching optimal.

5.2.2 Noisy Groups

Unfortunately, cluster merging is not the only thing that can account for a discrepancy between the number of canonical clusters and the number of empirical clusters. What often happens, especially in the presence of normalization and noise, is that all users who are not members of some explicit group are themselves collected into a sort of "noisy group," simply because their activity correlates with each other strongly — only the "noise" users fit the profile of having a roughly even level of activity over the entire observation interval, so they have relatively high normalized correlation between each other, and the clustering algorithm groups them together.

One approach that could be taken to deal with this could be to attempt removing each of the canonical clusters, run the set distance algorithm described above, and determine which group results in the greatest decrease in distance between the empirical and canonical clusterings, and declare that particular cluster the noisy cluster. I have not experimented with this particular idea yet, though.

6. Results

All of the pieces above were put together into a testing apparatus which is illustrated in Figure 6.1.

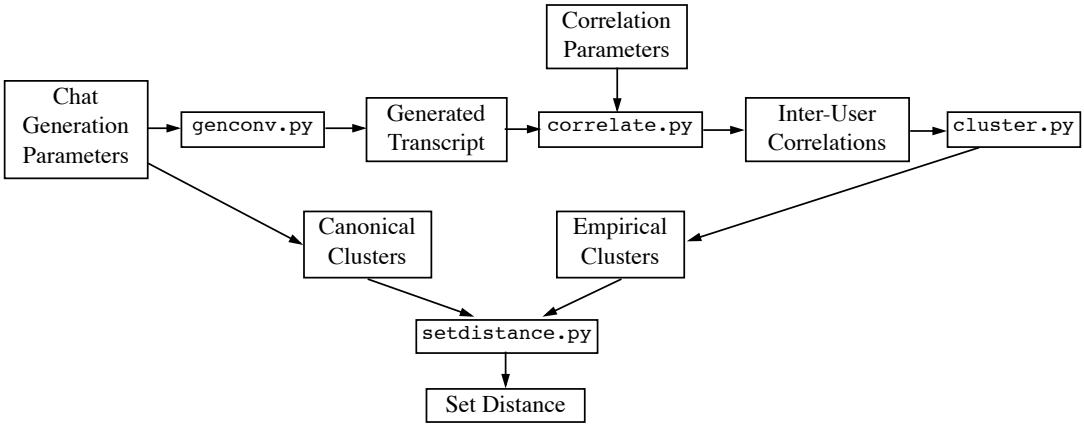

Figure 6.1: A flowchart showing the structure of information flow between programs. Boxes in `monospace fonts` represent programs.

The programs in the figure serve the following functions:

- `genconv.py` takes parameters for conversation generation (transcript length, number of users, number of groups, etc.) and generates a conversation randomly as described in Section 5.1. The generated transcript is saved as a single-table SQLite3 file.

- `correlate.py` takes a transcript and calculates the inter-user correlation values, according to a chosen correlation function and a given σ value.

- `cluster.py` takes the correlation graph and finds empirical clusters in it according to the clustering algorithm.

- `setdistance.py` finds the set distance between the canonical clusters output by `genconv.py` and the empirical clusters output by `cluster.py`. The final number that this program outputs becomes the measure of effectiveness of the

particular set of parameters given to the correlation and clustering programs with respect to the conversation generated.

This apparatus allows us to test the clustering algorithm with several different correlation and clustering parameters. The most easily varied parameter is the bandwidth of the Gaussian kernel, σ, as it gives us insight into what sort of associativity between messages is needed to best correlate different chats.

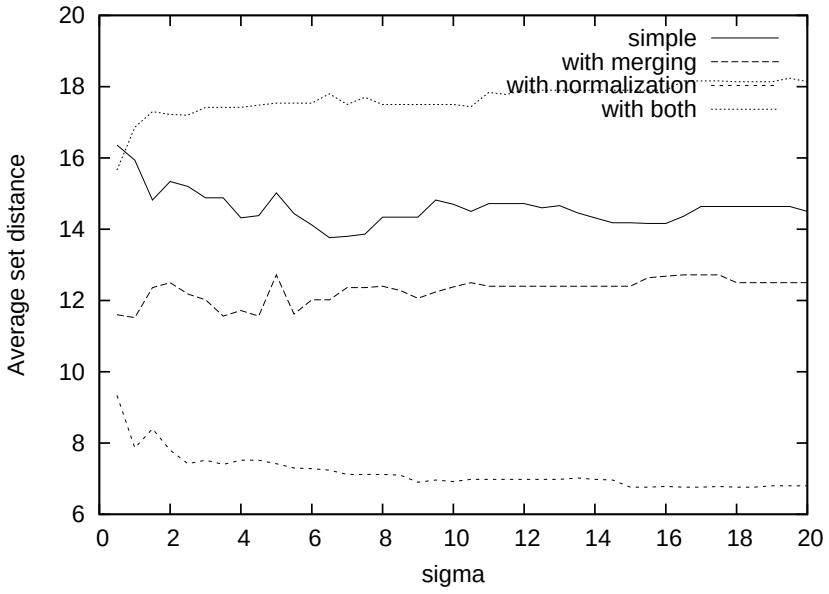

Figure 6.2: A graph of average performance (lower set distance is better) on a simple noiseless chat transcript with respect to σ, the Gaussian kernel bandwidth.

The plots in Figure 6.2 demonstrates the effectiveness of the above algorithm on a randomly generated chat with fifty users, three groups, and ten users per group,

with no noise. Clearly, when there is no noise in the transcript, the algorithm which performs best on average is the one using normalization but no merging compensation.

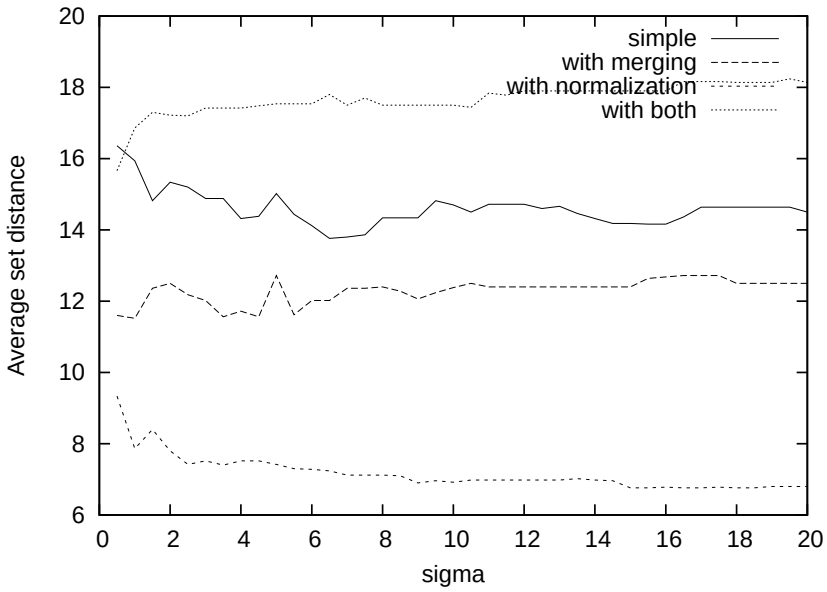

Figure 6.3: The performance on the same types of graphs as in Figure 6.2, but with noise (mean time between messages 15 minutes).

Strangely, the same appears to be the case in the presence of noise, although earlier experiments seemed to indicate otherwise. The plots in Figure 6.3 demonstrate the effectiveness of the same algorithms, but this time with noise ($\Delta t_{\text{noise}} = 15$ minutes) added for each user.

These results seem to indicate that performing normalization on these types of chat transcripts generally leads to more accurate clusters, whether noise is present

or not.

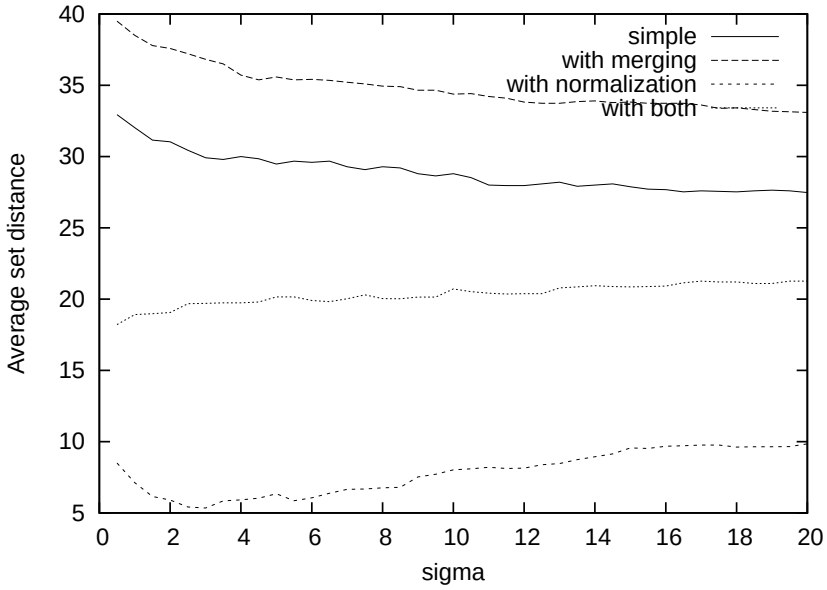

Figure 6.4: The performance on the same types of graphs as in Figure 6.2, but with noise (mean time between messages 15 minutes), and using an edge probability term with $\lambda = 0.8$.

Finally, the plots in Figure 6.4 show the effect of including the edge probability term, as described in Section 4.4. It appears that the edge probability term, in conjunction with normalization, helps with picking clusters out of noisy transcripts.

6.1 Conclusion

The algorithm described in this paper seems to, through limited testing, have been shown to be potentially effective as a method for finding hidden groups in chat

rooms. Although it still suffers from interference due to noise and group overlapping, there are a number of parameters of the algorithm that can be tweaked to give better results.

This is, of course, a curse of sorts, in that finding the correct parameters for a given type of chat can be hard. In addition, there is still the problem of there being no ground truth for many types of online interaction that one might want to subject to analysis. The random chat generation technique can aid here if one can agree that it's possible to randomly generate interaction that appears, qualitatively at least, similar enough to the interaction under investigation that it can be used to calibrate the correlation and clustering algorithms.

Literature Cited

[1] Baumes, Goldberg, et al. "Finding Communities by Clustering a Graph into Overlapping Subgraphs," *Proceedings of IADIS International Conference Applied Computing 2005*; pp. 97–104; 2005.

[2] Baumes, Goldberg, et al. "Finding Hidden Group Structure a Stream of Communications," *IEEE International Conference on Intelligence and Security Informatics (ISI 2006)*; San Diego, May 23–24; pp. 201-212; 2006.

[3] Kuhn, Harold W. "The Hungarian Method for the assignment problem," *Naval Research Logistics Quarterly*, **2**:83–97, 1955.

[4] Maxima — A GLP CAS based on DOE-MACSYMA. `http://maxima.sourceforge.net/`.

[5] Shi, Jianbo, and Malik, Jitendra. "Normalized Cuts and Image Segmentation." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8), 888–905, August 2000.

[6] Witten, Ian H., and Frank, Eibe. *Data Mining: Practical Machine Learning Tools and Techniques*, 2nd ed. Morgan Kaufmann. 2005.