The Matrix: A Parallel System for Analyzing Friends in a Social Network

Michael Matczynski

6.388 Final Report

May 18, 2006

Introduction

If we are to understand human socialization, we must first understand to what degree similar interests determine friendship. This project aims to shed some light on this saying "opposites attract" by analyzing friends on Facebook.com, a popular social networking site used by millions of college students and faculty. We hope to discover if similarities in group associations can systematically predict the likelihood of two people being friends.

Background

Social Networks are online websites that allow users to create profiles about themselves and link to the profiles of their friends. Uses often list interests, hobbies, contact information, and pictures about themselves. Additionally, user-created groups allow users with similar interests to find and communicate with each other. For example, if a user on Facebook.com was interested in Mathematics, he or she might join a "I Love Math" group find others with the same interest.

Figure 1 shows an example screenshot from Facebook.com. The left side of the page shows MIT friends and lets a visitor view all the people this user considers a friend in real life. On right side is a listing of groups that this user has selected to be a part of. Group types are user-defined and commonly include living groups such as "Simmons Hall," entertainment and movies such as "Casa Blanca," and events such as "Boston Marathon 2005."

Figure 1: An Example Facebook.com User Profile

Steps

Analyzing social network data to determine if similar interests imply common friendship required a series of steps:

1. Gathering friend and group data from all MIT users on Facebook.com
2. Building a graph of users and determining their features
3. Developing algorithms to classify users based on the similarity between two people
4. Parallelizing three stages of computation

Gather Data

The first step of the project involved retrieving friend information from Facebook.com. To protect the innocent and respect the privacy of individual users, no personal information was collected. The only identifying information was a Facebook-defined user ID. Every college has a different range of user IDs, and MIT has the range beginning with 700,000. User IDs starting at 700,000 were collected until no more users were found. This resulted in 11,744 MIT user profiles being collected.

Once the friend data was collected, common interest group membership was retrieved. This involved retrieving group information for every user collected above. Groups have a wide range of popularity, though at MIT all the groups had between 1 and 350 members in them. Around 3,000 groups were collected in total.

Build Graph

The data collected represents an undirected graph with users as nodes and every friendship as a link between nodes. Facebook allows users to change their privacy settings to keep their friendship or group associations private. However, because a friendship link is undirected, even if a specific user had their friend information private, this user would still be represented in the graph because their friends may still have links to that user visible.

Once the graph of friends was built, the system then created a FeatureVector for each user node. A FeatureVector is simply a way to project a user into a multi-dimensional space using features specific to a user. Two different sizes of features were used for comparison, small features and intermediate features.

Small Features

Small features involved represented a user's group membership as 0's or 1's in the FeatureVector. This meant that a FeatureVector for MIT users is a 1x3000 vector.

Intermediate Features

Based on the idea of intermediate features used in image classification[1], many of the groups were first classified into around 80 intermediate features. This process was done by hand and involved picking features that seemed to be general enough to encompass multiple groups, yet small enough that it didn't have too many groups. Unlike FeatureVectors built using small features, FeatureVectors based on intermediate features may have values greater than 1 in them as a user can be in multiple groups that share the same feature description.

This table shows examples of intermediate features used for various Facebook groups.

Group Name		Intermediate Features
Classical Musicians	→	+Classical Music
John Williams Fans	→	+Classical Music
College Democrats	→	+Politics, -Conservative
College Republicans	→	+Politics, +Conservative
Napoleon Dynamite is a Retarded Movie	→	+Movies, -Napoleon

Develop Algorithms

An algorithm was developed to determine the similarity between two users' features. For intermediate features, the multi-dimensional FeatureVector was normalized and then compared to another FeatureVector using a distance metric. This resulted in values ranging from 0 for two users with identical intermediate features to large numbers for users with radically different features.

For small features, a different metric was used that determined a value between 0 (similar) and 1 (different) for every pair of users. This metric involved finding the amount of group membership overlap. Suppose users A and B have group sets X and Y respectively. The metric used to gauge similarity was $f(X, Y) = 1 - (X \cap Y) / (X \cup Y)$.

[1] Ullman, S. Visual features of intermediate complexity and their use in classification. 2002. http://courses.csail.mit.edu/6.803/pdf/features.pdf

Parallelize Computation

Because of the $O(n^2)$ complexity used to compare every possible user-user pair at MIT, the different stages of computation were parallelized. Because this was an open research project requiring the algorithm to undergo much iteration, the goals of the system were to isolate faults and provide modular stages of computation. These goals sped up development time and testing as computation could be reused. Additionally, crashes occurring in one stage would not affect previous computation.

JavaMPI was used to parallelize the three stages of computation as described below. Due to the large size of the dataset, the Java virtual machine's heap size was increased to 1 GB using the command line parameters "-Xms1024m -Xmx1024m" to prevent OutOfMemoryException.

Partitioning User-User Computation

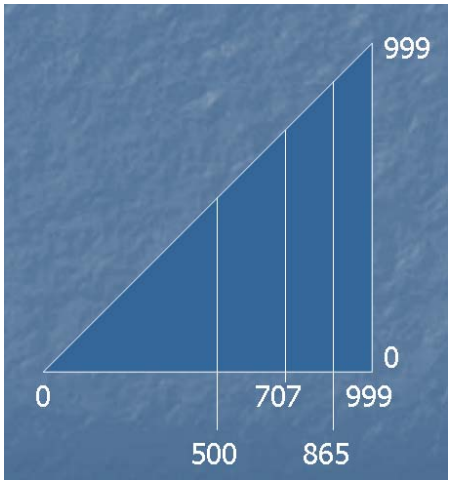

Figure 2: Triangular Parallelization

The difference metric between two users does not depend on their order as f(a, b) = f(b, a). Thus, to parallelize the task of inspecting all user-user pairs the following formula was used split the computation regions into equally sized areas: c = sqrt((a2 + b2) / 2). Applied recursively, this assigned each processor an equal amount of work. This technique was used only for stage 1, which is described next.

Parallel System Architecture

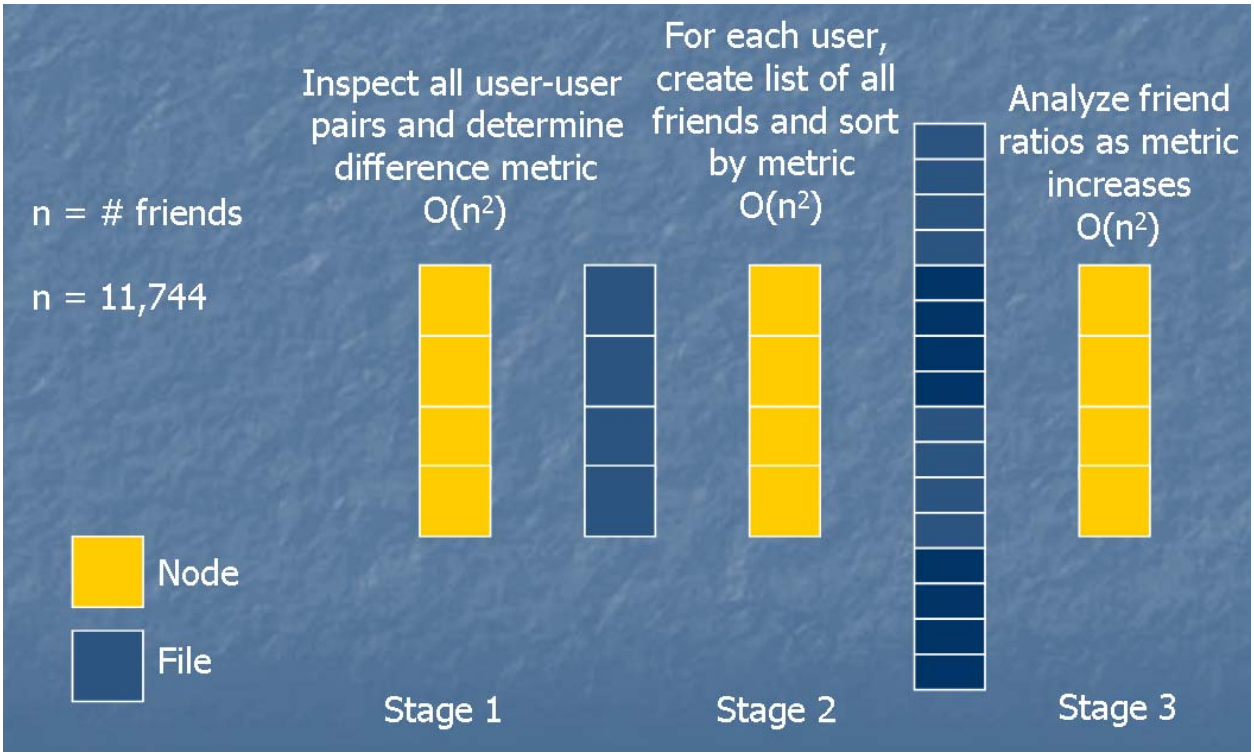

Figure 3: Parallel System Architecture

Figure 3 shows the parallel system architecture used to analyze the social network data.

Stage 1 inputs friend connection and group membership data to analyze every user-user pair. For each pair, a difference metric between 0 (similar) to 1 (different) is determined. Each processor outputs its section of computation to a large file named [myrank].txt.

Every stage 2 processor reads every large file from stage 1 and inspects each user-user pair. If either user is in the range of users the processor is monitoring, it keeps that pairing in memory. Thus, the processor builds up multiple lists of metrics, one for every user in the range of users the processor is monitoring. Once all data is read, each processor sorts the lists and writes them to individual user files resulting in nearly 12,000 lists written to disk.

Stage 3 reads the user list files and calculates statistics such as the number of friends and friend probability for different ranges of the difference metric.

Algorithm Results

Figure 4 and Figure 5 show the results of using intermediate and small features in friend classification. The "Friend Ratio" in both figures shows the percentage that users are friends given a user difference metric (plotted on the X axis). The metric of 0 represents a pair of people who are exactly similar in feature/group associations whereas a metric of 1 represents a pair of people with nothing in common. This ratio is calculated as the (# of friends) / (# of pairs of people). Each figure also plots "Num friends" which shows the percent of the total number of MIT users that fall under that user difference range.

As used in this project, intermediate features derived from group associations do not effectively predict friendships as varying the amount of user difference from 0 to 1 had negligible impact on the friend ratio.

On the other hand, small features had remarkable predictive ability. This means that if two users have a large number of common group associations, they are much more likely to be friends. It should be noted that for small features, a large majority of user pairs fell between 0.8 and 1.0 in the difference metric. In this region the predictive power is much weaker, varying from 16.2% to 4.7% and below.

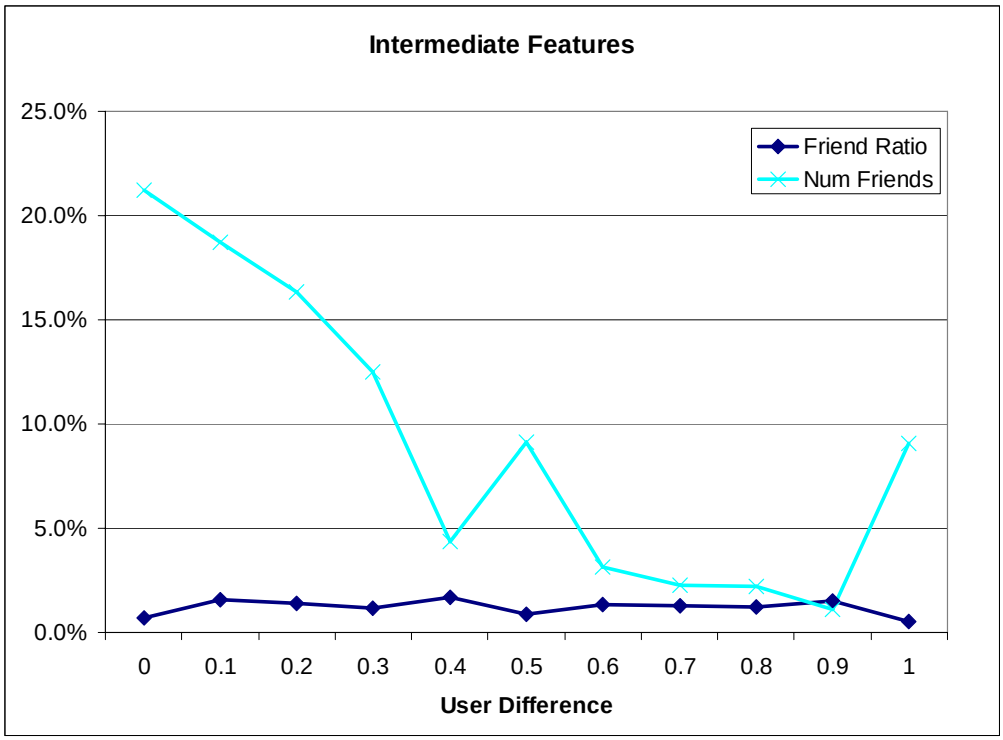

Figure 4: Results of Intermediate Features

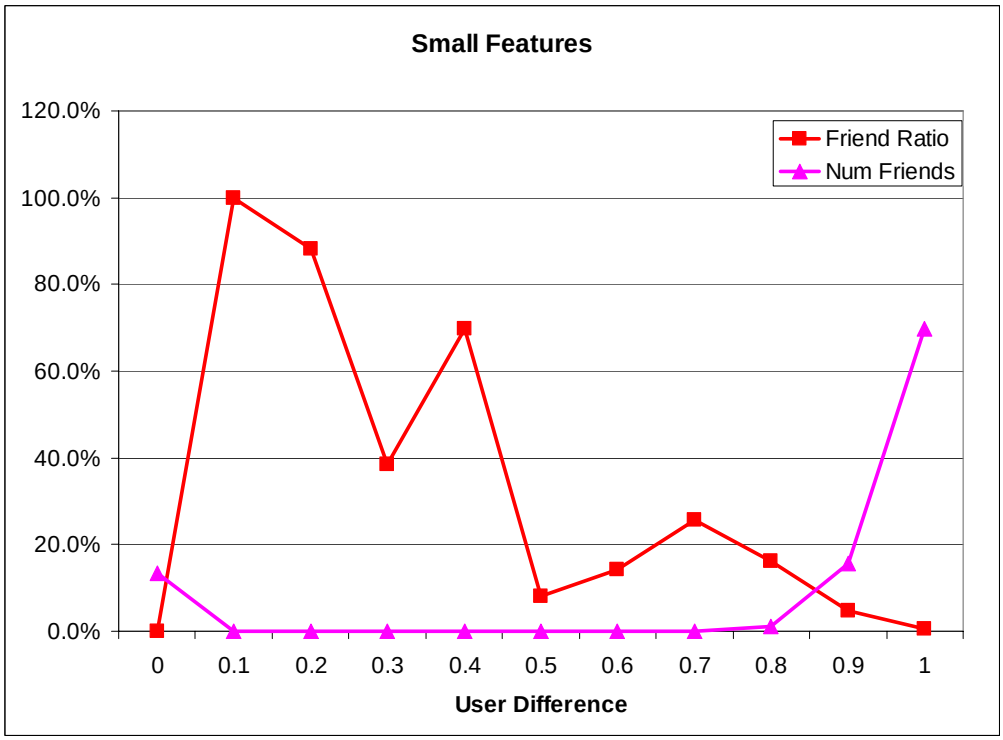

Figure 5: Results of Small Features

Performance

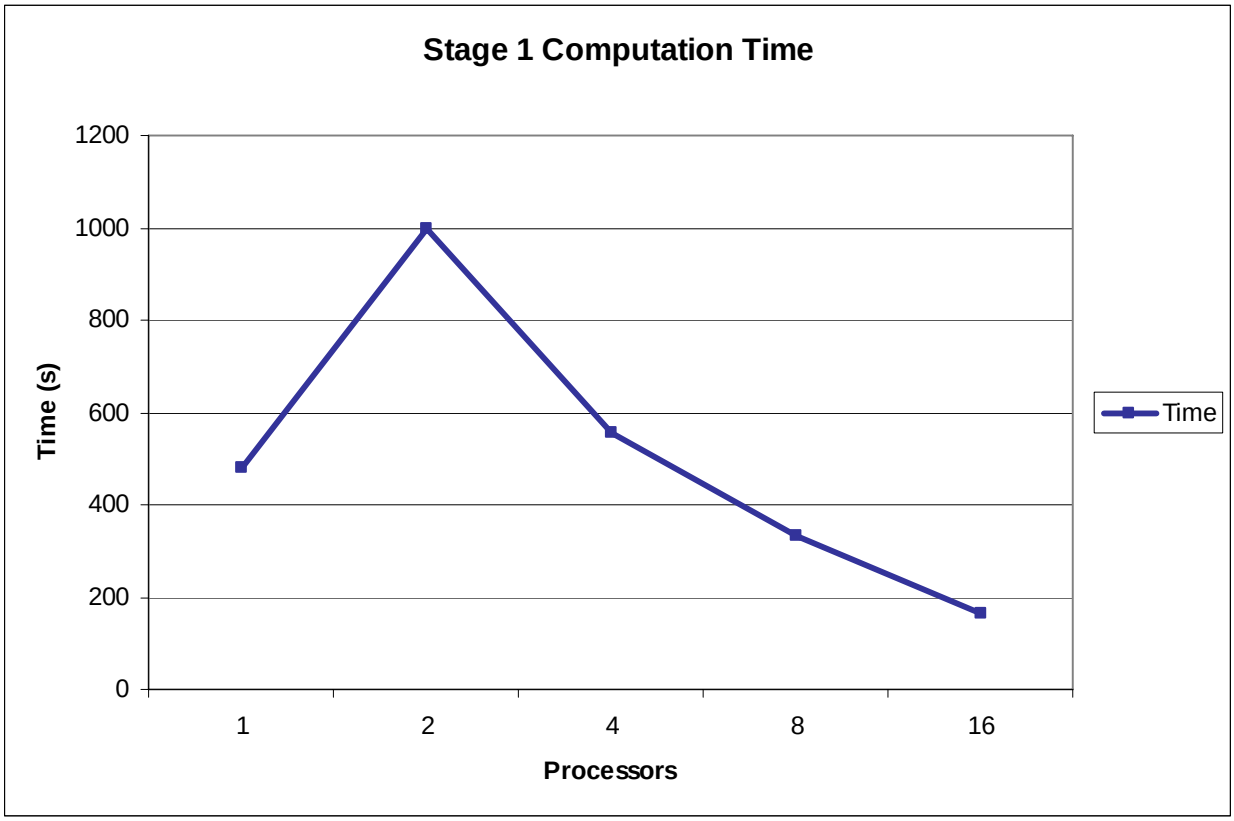

Figure 6: Stage 1 Performance

Stage 1 involved inspecting every pair of users and calculating a difference metric based on their FeatureVectors. Although one processor can perform better than two or four processors, eight and above processors see performance gains. This initial increase in computation time using multiple processors is suspected to be the result of a large amount of duplicated computation as cached results in the software is not shared among processors. A distributed cache scheme may have prevented the initial increase in computation time.

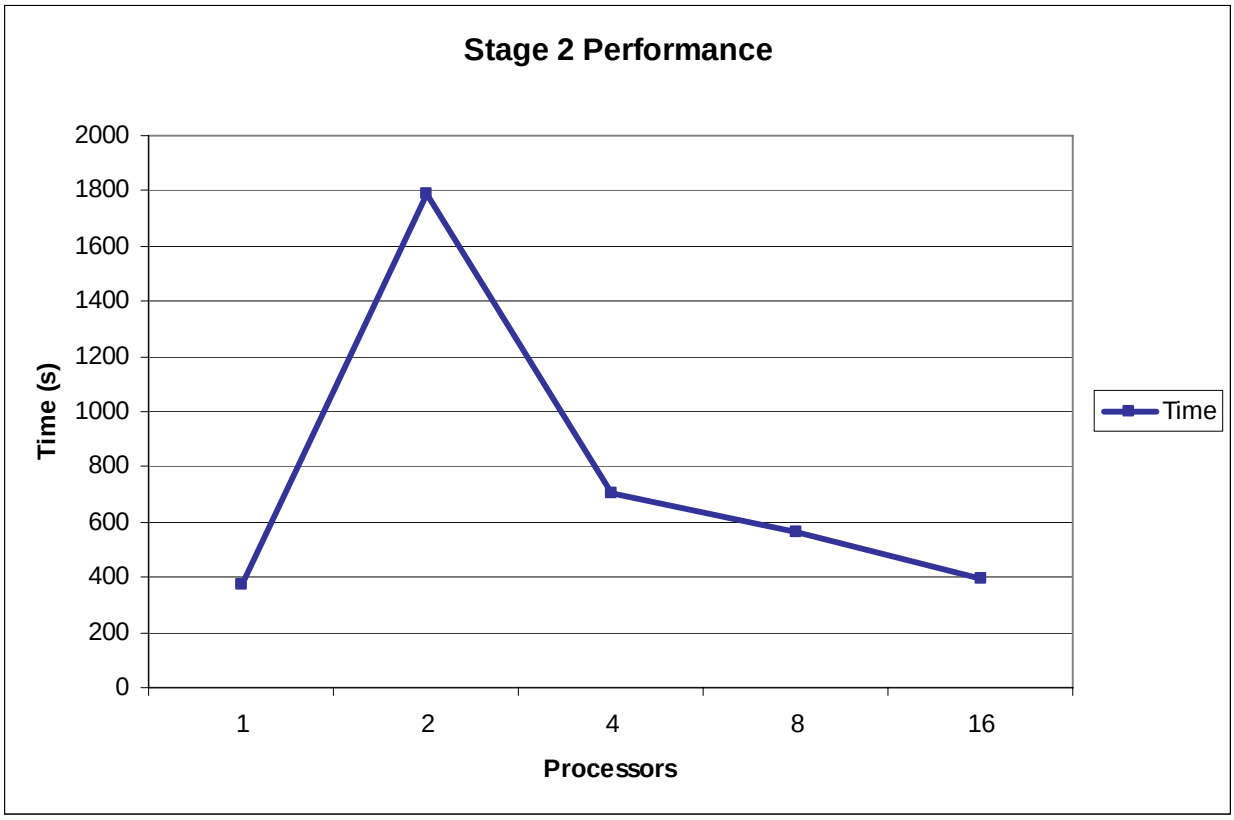

Figure 7: Stage 2 Performance

Stage 2 involved reading the friend pair metrics from stage 1 and creating a sorted list of every user's friends. This computation did not scale and the best performance was on a single processor. The bottleneck in this case is suspected to be reading the results of stage 1 which where 1.2 GB in size. This stage required all processors to read through the entire stage 1 result and only keep the subset of friends in memory. Read contention may be to blame for the poor performance.

Although the sixteen processor run was equivalent time-wise to the single processor run, it required approximately only a sixteenth of the amount of RAM as the single processor run. Thus, if the data set was too large to fit in memory on a single machine, parallelization may still be useful.

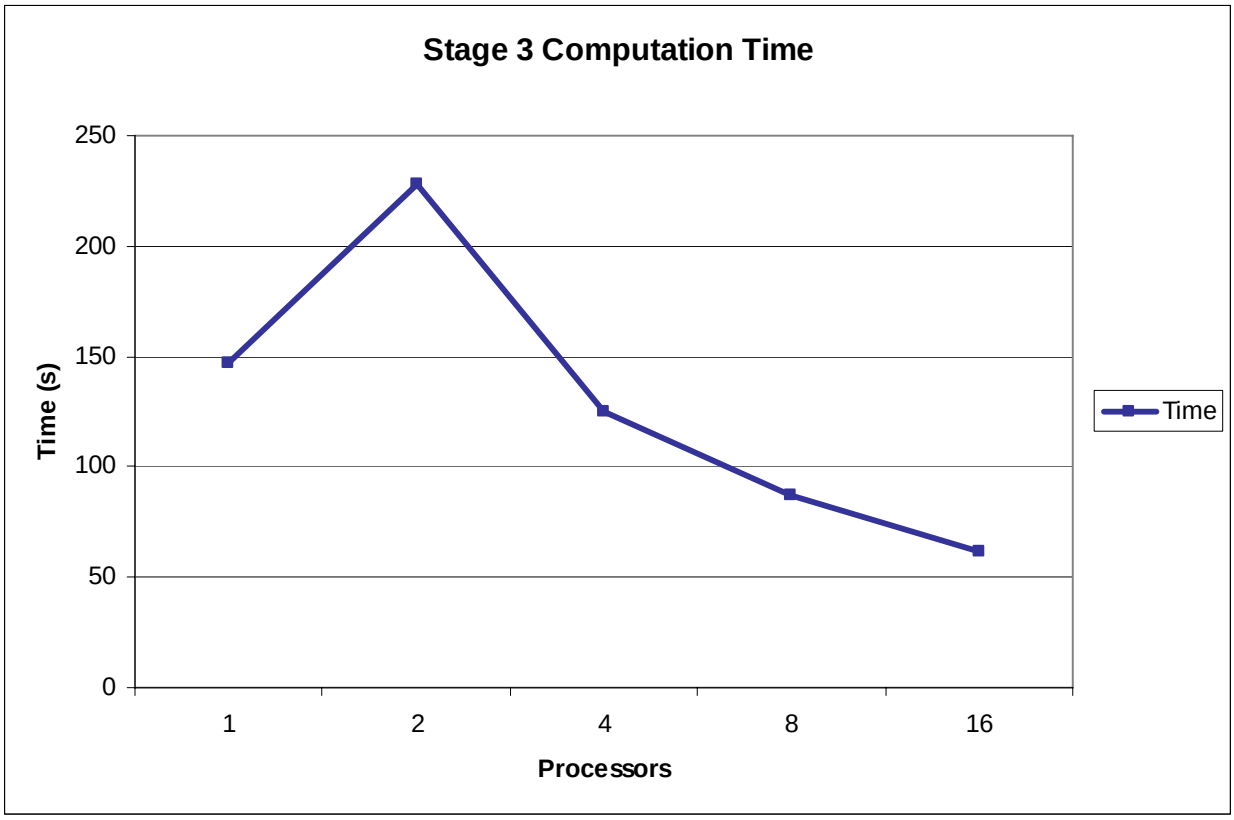

Figure 8: Stage 3 Performance

Stage 3 involved reading the sorted lists for each user and finding statistical information such as friend ratio and number of friends for varying ranges of the difference metric as displayed in Figure 5 and Figure 3. Although the two processor test performed worse than the single processor test, additional processors greatly increased performance.

Contributions

In this project I have:

- Developed an algorithm that provides evidence that people with similar interests are more likely to be friends
- Showed that intermediate features may not be the best measure of similarity as small features provided greater predictive power
- Parallelized multiple stages of computation with JavaMPI, increasing performance up to three times