

On Clustering Images Using Compression

Benjamin Hescott
Daniel Koulomzin

February 22, 2006

1 Introduction

1.1 Cluster analysis

The need for the ability to cluster unknown data to better understand its relationship to known data is prevalent throughout science. One example is classifying species by genetic structure, another is grouping stars by temperature and size. Besides a better understanding of the data itself or learning about a new unknown object, cluster analysis can help with processing data, data standardization, and outlier detection. These reasons for clustering data have created a wide range of algorithms. Most clustering algorithms are based on known features or expectations, such as the popular partition based, hierarchical, density-based, grid based, and model based algorithms. The choice of algorithm depends on many factors, including the type of data and the reason for clustering, nearly all rely on some known properties of the data being analyzed.

Recently, Li et. al. proposed a new “universal” similarity metric [3], this metric needs no prior knowledge about the object. Their similarity metric is based on the Kolmogorov Complexity of objects, the objects minimal description. They use Kolmogorov Complexity to measure objects’ similarity by considering how much information one object contains about the other. While the Kolmogorov Complexity of an object is not computable, in “Clustering by Compression”, Cilibrasi and Vitanyi use common compression algorithms to approximate the universal similarity metric and cluster objects with high success [4]. They show for genomic sequences, music, and literature it is possible to cluster accurately using a metric calculated from the compression of an object.

Unfortunately, clustering using compression does not trivially extend to higher dimensions. Informally one can consider the dimension of an object to be the number of directions possible when describing it fully from a starting point. For example consider a novel, we say it is one dimensional as there is a beginning, an end, and the words in between are in a specific order in one direction. A story begins “Once upon a time” and finishes with “The End”, and is read in a linear fashion. However, some data naturally manifests in more than one dimension, and is in fact difficult to express reasonably in fewer dimensions.

A simple example is an image. At its simplest representation as digital data, an image is a set of instructions on how to display a specific coordinate in two dimensional space. Of course, we routinely represent this sort of data in linear fashions (e.g., as a file); however, the information itself is multi-dimensional. The semantic gap between one and two dimensional representations is easy to demonstrate. Consider a picture that has 100 pixels, arranged in 10 columns and 10 rows. In converting this to a linear representation, we might choose to start at the pixel in the 1st column and 1st row. However, after this, our troubles begin, which pixel shall we move to next? The two closest pixels, to the right, and below the first pixel, that are equally close. Even if we arbitrarily choose one (say the one to the right), then our problems get worse, now we are forced to choose between a pixel closest to the second pixel, and the other pixel that was closest to the first. With either choice, we have misrepresented the closeness of all the pixels in question. This problem compounds as we consider more pixels. It is exactly this issue considered in this work, we test different methods of translating images into one dimensional “text like” objects and use compression for clustering like images.

This report is presented in five sections. In section one we introduce the methods and concepts of the work previously done, reviewing both Kolmogorov Complexity and the metric defined in Li et. al. [3]. Section two outlines the implementation used, while section three describes the preprocessing used for images. Section four contains experimental results on a small data set, and finally section five concludes the discussion and proposes future work.

1.2 Kolmogorov Complexity

One way to classify a particular object is to measure the amount of information inherent to the object, this measure of information can be formalized by Kolmogorov Complexity. Before defining Kolmogorov complexity we can motivate these ideas with a few simple examples. First we can think of the Kolmogorov complexity of an object as its shortest description. Considering this measure we can say that an object is complex if its shortest description is very long. For instance, if we compare the description of a car horn and Mozart’s sonata for two pianos, to write down the car horn is simple, most car horns play an F, but to write down the sonata is more difficult. Of course one could argue that the sonata lasts longer, but consider a horn that is pressed repeatedly for an hour. Here we need only describe the length of time and the note. In contrast, knowing a single note within the sonata and the duration of the concert is not nearly enough information to describe the work. In this example an important property of Kolmogorov complexity is witnessed that carries through for binary strings (the only objects we will formally study.) The minimum description of any binary string is bounded by its length, we also know the upper bound is the object itself — to describe the sonata we can use the sheet music.

In order to define Kolmogorov Complexity we need to review some standard definitions. Let M describe a Turing machine, and U be some fixed universal Turing machine, and when we consider an object, we refer to a binary string.

Note that a Turing Machine, U , is universal if on input M and x , with M encoding a specific Turing machine and x a valid input to M , U will simulate M on input x . For more information on these standards one can read Homer and Selman [1].

Adopting this notation we can now formally define Kolmogorov Complexity as follows:

Definition 1 (Kolmogorov Complexity) *The Kolmogorov Complexity of a binary string x denoted $K(x)$ is:*

$$K(x) = \min\{|d| : U(d) = x\}$$

Here it is important to note that the choice of machine does not alter the value of $K(x)$ by more than a negligible amount, enabling us to only consider the universal Turing machine U . Notice that with this definition that $K(x) \leq |x| \forall x$, since any machine can be preprogrammed to just output x given all of x by copying it to its output tape.

The above definition considers unconditional Kolmogorov Complexity, that is the universal machine is not given the knowledge of any other string. It will be necessary to consider the conditional version of Kolmogorov Complexity in this project.

Definition 2 (Conditional Kolmogorov Complexity) *The Conditional Kolmogorov Complexity of a binary string x given y , denoted $K(x|y)$ is:*

$$K(x|y) = \min\{|d| : U(d, y) = x\}$$

Here we give the universal machine, U , the additional input of y , that is $K(x|y)$ is the shortest description of x given y . Informally one can think of the conditional complexity as the amount of information there is in x given y . It is easy to see how this might be used to compare two objects, objects which are similar would have a low conditional complexity, that is if x and y are similar very little additional information would be needed to describe x given y .

Notice that $K(x|\epsilon) = K(x)$ where ϵ is the empty string, and that $K(x|y) \leq K(x) + c$ where c is a small constant from the overhead of simulating the universal machine. This follows from the fact that y can only help U describe x , at worst the machine can simply ignore y and continue with its computation.

Also it is important to understand the Kolmogorov Complexity of concatenation. Here let $\langle x, y \rangle$ denote concatenation. We know that

$$K(\langle x, y \rangle) \leq K(x|y) + K(x) + O(\lg(\max\{|x|, |y|\}))$$

Also it is known that this bound is tight,

$$K(x|y) + K(x) \leq K(\langle x, y \rangle) + O(\lg(\max\{|x|, |y|\}))$$

This relationship is called symmetry of information. The information content of y in x is defined as $I(x : y) = K(y) - K(y|x)$ which yields

$$I(x : y) = I(y : x) \pm O(\lg(\max\{|x|, |y|\}))$$

or that information is in fact symmetric, at least to a log of log factor. Notice that the above formula can be rewritten as

$$K(x|y) \geq K(\langle x, y \rangle) - K(x) - O(\lg(\max\{|x|, |y|\}))$$

and

$$K(x|y) \leq K(\langle x, y \rangle) - K(x) + O(\lg(\max\{|x|, |y|\}))$$

It is this value $K(x|y)$ that will give us the metric needed for clustering.

1.3 Clustering and Compression

It is very natural to use the concepts of information theory and Kolmogorov Complexity to characterize data. How much information an object has can say a lot about that object. Even more telling is how complex an object is when given another object: the more one object can “say” about another, the more similar the two objects. It is this idea that is used in both “The Similarity Metric” and “Clustering by Compression”. In the former paper Lit et al. [3] consider the following as a metric to measure similarity.

$$\frac{\max\{K(x|y), K(y|x)\}}{\max\{K(x), K(y)\}}$$

Of course, there are problems with this definition. First, it is impossible to calculate the Kolmogorov Complexity of an object, and second, $K(x|y)$ must be approximated using $K(\langle x, y \rangle)$. However, the authors define a more practical metric by considering standard compression algorithms as procedures to calculate the Kolmogorov Complexity, and approximating $K(x|y)$:

$$\frac{\min\{C(\langle x, y \rangle), C(\langle y, x \rangle)\} - \min\{C(x), C(y)\}}{\max\{C(x), C(y)\}}$$

Here C refers to a standard normalized compression algorithm. In practice a compression algorithm will typically compress $\langle x, y \rangle$ and $\langle y, x \rangle$ to the same size since most algorithms are block based, in our test we will only calculate $\langle x, y \rangle$.

The authors use this new metric to obtain a distance matrix by calculating pairwise distances within the data set. They use a hierarchical clustering algorithm based on quad trees to cluster objects, and perform many statistical tests to verify that this is in fact a reasonable metric. We propose to consider higher dimensional objects by relaxing some of the assumptions about the metric.

2 Implementation Specifics

2.1 Calculating the Dissimilarity Matrix

We ran several experiments, each consisting of several trials. In every experiment, our sample data consisted of images, each of a letter A, B, or C. In the

first experiment, we considered images of letters that varied in position and size. In the second experiment, we considered images in which font, size and position were fixed, but the letters were rotated 0, 90, 180, or 270 degrees. In the third experiment, we considered images where typeface varied, but size and position were kept constant.

In each trial, we varied the pre-processing and concatenation of the objects as described in the next section.

Equipped with our pre-processed data set, we conducted our trials in the following manner: first, we used the BZIP2 algorithm as an approximation of Kolmogorov Complexity to calculate the distance metric. Next, we applied a clustering algorithm to group the objects based on their respective distances.

We chose the BZIP2 algorithm because it is a block compression algorithm which ensures that the compression of the concatenation of two objects will not vary appreciably with the ordering of the objects in the concatenation. This allows us to calculate the distance metric with a simplified formula:

$$\frac{C(\langle x, y \rangle) - \min\{C(x), C(y)\}}{\max\{C(x), C(y)\}}$$

Using the Apache project’s Java implementation of BZIP2, we computed the metric for all possible pairs of test images and stored these into a matrix. We then used this matrix as the dissimilarity matrix for our clustering algorithm.

2.2 Clustering

After the dissimilarity matrix was calculated for each test sample, with each ordering method and concatenation method, we clustered the set. We used the agglomerative hierarchical clustering algorithm AGNES (AGglomerative NESTing). Specifically we started with the a singleton cluster for each image, and then began iteratively merging the two closest clusters (as defined by our distance matrix) until a sufficiently low number of clusters was reached. We clustered the data twice for each trial, determining the closeness of clusters either by considering average distances or minimum distances of members. In the former approach, two clusters X and Y had distance equal to the average distance of each member of X from each member of Y. In the latter, clusters X and Y had distance equal to the distance between the closest images x and y where x was in X and y was in Y.

Between each iteration, we examined the dendrogram created. We generally chose our target number of clusters to be equal to the natural clustering of the images, but we also chose lower and higher numbers to force incorrect clustering as well.

3 Preprocessing

3.1 Images

The importance of clustering images is prevalent in many research areas. It is not only possible to build efficient databases for images, but also to do more general pattern matching, such as face recognition, sign language, and human computer interface. It is these issues that warrant an investigation of how to cluster images using compression. While one of the most desirable properties of clustering by compression is the ability to cluster without prior knowledge of the sample space, it is necessary to relax this condition for clustering images. We limited the investigation to images which have not been compressed such as bit maps, specifically pbm files. PBM files are black and white (we will not consider color images in these experiments.) The files are not compressed and are a true bitmap, each entry in the file is the bit representing the pixel. Note that we converted the PBM style usage of ascii characters to actual bits before processing the image.

3.1.1 Test Samples

We first considered clustering images of letters of the alphabet, specifically A, B, and C. We created black and white PBM files of images of one of each letter, we varied the position, font, size, and rotation of the letter chosen. We chose these three letters since they are not similar and should group into separate categories even when the above attributes are changed. We worked with a small set for each test ranging from 30-40 images drawn from a sample space of nearly 300. These images were created specifically for these tests using Paint Shop Pro. In future work we will move from simple letters as the test sample to images of landscapes and portraits, we hope to see clustering around whether an image contains a face or does not contain a face.

3.1.2 Two Dimensions to One

The first step for processing images is to convert the two dimensional object into a one dimensional object for compression. As a base case we considered the metric without change, specifically we did not convert the image from its natural form, and consider the bits row by row, then we considered space filling curves, specifically the z-ordering curve.

3.1.3 Concatenation

In order to calculate the similarity metric we needed to concatenate the images. Once they are in one dimension, one easy approach is to simply concatenate them trivially. We tested this naive method first, since it is the most universal and canonical. Since we have violated the “universal” properties by assuming we have images and we have calculated the space filling curves, we also sought to discover if it is effective to interleave the two images. That is, we calculated

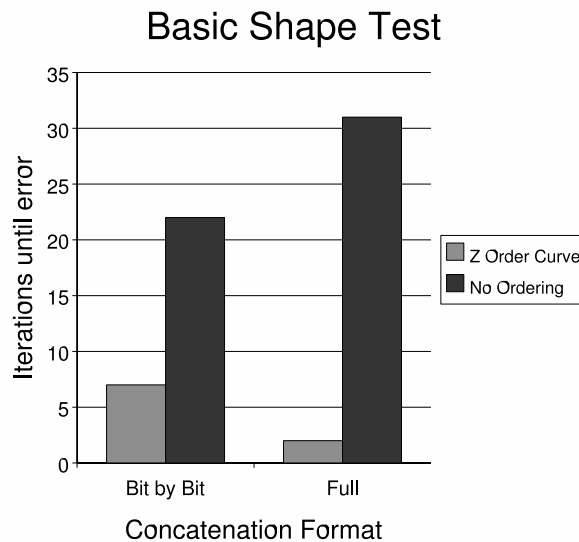
the metric using the Kolmogorov Complexity of the bitwise interleave of the images rather than their concatenation.

4 Experimentation Results

4.1 Basic Test Case - Size and Position

Here we used a sample size of 35 different letters. We had twelve different images of each letter with three different sizes in four different positions. We varied the position of the letter inside of the image from top left, bottom left, top right, bottom right. We varied the size between 36 point, 72 point, and 149 point. The images were 256 by 256 pixels in size.

We considered a trial's success to be measured by how many iterations of the clustering algorithm could make before merging two clusters containing different letters.



4.1.1 Z Ordering vs No Ordering

Surprisingly, the z order did not result in better clustering in any of the trials. When using the z order with a bitwise interleave and calculating the distance using averages, our clustering algorithm incorrectly placed A's and B's in the same group by the seventh iteration. When using a one bit interleave with minimum distance the z ordering strategy was even worse, incorrectly clustering an A with a C at the fourth level. The worst trial was a z order with full concatenation using the minimum distance clustering algorithm: it failed after the second iteration grouping a B with a C. It is important to note that the z order mistakes did seem to group by the size of the characters, specifically clustering letters

of 72 point together first. This contrasts strongly with our results when no ordering was applied: no ordering resulted in perfect clustering in both full concatenation tests.

4.1.2 Bit Interleave

When performing trials using bit interleave, we noticed immediately that the distances we obtained did not seem to normalize well, with values in excess of 1.3.

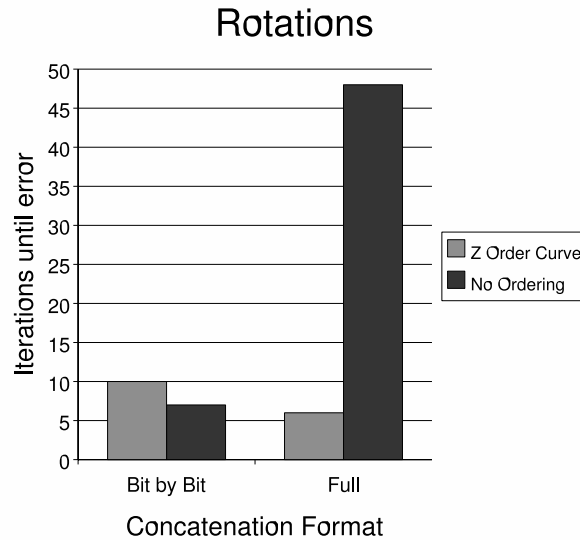
However, bitwise interleave gave better results than concatenation when using the z-order curve: the first clustering error appeared after 7 or 8 iterations with interleave, rather than the 2nd and 4th iteration with concatenation. However, it did not help the no order trials, which did not perfectly cluster the last two images when using a bit by bit interleave. It is interesting to note that it was again the size rather than letter that determined the clustering when errors began to occur.

4.1.3 Min Distance vs Average Distance

Within this data set there was no appreciable difference with these two measures. The only possible thing to notice would be that using the average seemed to make z order worse than it was without. This seems to agree with our understanding that an average distance algorithm can fail much more quickly than a minimum distance algorithm.

4.2 Rotations

Here we considered 60 images, 20 of each letter A, B, C with 4 rotations at 90 degree intervals, with 5 each for the samples. This time the font, size, and general position were fixed. The file size was 512 pixels by 512 pixels.



4.2.1 Z Ordering vs No Ordering

Once again, considering the pixels in normal order resulted in better clusters than z ordering. The z order began clustering the A's together with ones that had been rotated by 180 degrees. Similarly it would join a sideways A with a C within the first 20 iterations. Once these mistakes were made most A's and C's would cluster. It is interesting to note that B's were treated as outliers in all of the z order tests. These errors were in stark contrast with the no order, which achieved perfect clustering when the full concatenation was run with an average distance calculation. It is very interesting to note that while z order treated B's as outliers the only perfect clustering in this set of test clustered B's first.

4.2.2 Bit Interleave

Here the bit interleave did make a difference: while the z order errors seemed to outweigh the concatenation method chosen, the differences with no space filling curve were far greater. Running the trial with normal ordering and bitwise interleave, within the first 10 iterations our algorithm began to cluster upside down and regular A's. Soon after, C's and A's were clustered; again B's were outliers, not clustering together until the very late iterations.

4.2.3 Min Distance vs Mean Distance

Interestingly, full concatenation, and normal ordering, clustered using average distance achieved a perfect clustering. Using min distance instead introduced an error in which the final A joined with one rotated 180 degrees.

4.3 Unusual Fonts

For this set of trials, we included nearly 50 images of approximately 15 images of each letter. We fixed the font size and position but varied the style of the letter to some extreme cases. Each image had size 512 pixels by 512 pixels. None of these trials produced successful results.

4.3.1 Z Ordering vs No Ordering

It is noticeable that the no order was more resilient than the z order: in this set of trials, traversing the files in z order resulted in errors within the first clustering iteration. We noticed that typically the clusters represented similarities of typeface instead of letter.

4.3.2 Bit Interleave

Using bitwise interleave instead of concatenation tended to strengthen the tendency to cluster by typeface. Concatenation was more likely to correctly group images by letter, although never very successfully errors occurred at the seventh iteration.

4.3.3 Min Distance vs Mean Distance

There was no difference between the cluster-distance method used, other than using mean distances seemed to cause errors to propagate more quickly.

4.4 Final Analysis

It seems overwhelming that the correct method is the normal order traversal with concatenation. The remaining question is the distance used for clustering. This seems to be a lesser consideration, especially if we alter the actual clustering algorithm. The extreme outliers should be thrown out of the font cases and retried on an easier subset; we conjecture that the algorithm would perform well once these have been removed. We informally tried the algorithms on a superset of the first images and it seems to do well again using no order with full concatenation; we suspect that it can be adapted to other letters.

5 Conclusions and Future Work

5.1 A different cluster method

While there were some interesting results with a simple clustering algorithm, they did not clarify when the algorithm was working perfectly. Perhaps an algorithm suited for higher dimensions such as CURE would provide different results. It seems as if a method which is more sensitive to similar distances would work the best. Most metrics were within two tenths of a point of each other in many of the test cases.

5.2 A Larger Database

Subsequent experiments would benefit from the consideration of a larger sample set. While over a hundred images takes some time to process, it does not model a real database. We have talked to members of the Image and Video Computing Group here at Boston University to access their extensive library of images. Our next set of trials will be with a database of images of letters and numbers with a total sample size of nearly one million. Also, it seems that outliers can quickly spoil the results of a test run. Instead of focusing on simply whether a weirdly written B clusters with a strange C, potentially more letters and more consistent fonts may yield results which can determine if this method is worth pursuing.

5.3 Different Images

Is it possible to branch out beyond simple black and white bit maps? While it is imperative that the image not be compressed, can we use a more complicated image? Can we cluster colored images? Can we consider this as a face recognition program? Can we count fingers, or understand sign language?

5.4 Graphs and NP Approximations

Many important computation problems involve determining specific properties of graphs. Questions like does this graph have a Hamiltonian path or given two graphs are they isomorphic are believed to be infeasible. Again by relaxing the requirement of no prior knowledge about the data set is it possible to cluster graphs by compression? Could we use compression to decide whether two graphs are isomorphic? Given a set of graphs with Hamiltonian cycles and graphs without Hamiltonian cycles can we add an unknown graph and decide with some amount of certainty if it had a Hamiltonian cycle? Another possible approximation is MAX-CLIQUE, here it may be possible to approximate the max clique problem by considering the distance of a unknown graph to a known one. The hope is that clique size will be a cluster property. It is known that even approximating MAX-Clique within a constant is NP-Hard, this could be a natural heuristic without a specific error bound.

5.5 A Better Error Bound

All of this work follows an extreme heuristic; not only is the pure metric infeasible, it is not calculable. Is it possible to transform this metric into something that is decidable? There is a proof that any reasonable time bounded machine which can calculate this metric could also be used as a universal decoder for public key cryptography. Can we achieve a space bound? What about a provably infeasible time bound, perhaps something within exponential time?

References

- [1] S. Homer and A. Selman, *Computability and Complexity Theory* Texts in Computer Science. Springer, New York, 2001.
- [2] M. Li and P. Vitanyi *An Introduction to Kolmogorov Complexity and Its Applications*. Graduate texts in Computer Science. Springer, New York, second edition, 1997.
- [3] M. Li, X. Chen, X. Li, B. Ma, and P. Vitanyi, The similarity Metric, Proc. 14th ACM-SIAM Symposium on Discrete Algorithms, 2003.
- [4] R. Cilibrasi, and P. Vitanyi, Clustering by Compression, IEEE Trans. Information Theory (submission).

A Instructions for Running Chachkee

This document explains how to use the Chachkee software. If after reading this document you still have questions or comments, please write to “dkoulomzin@comcast.net”.

A.1 System Requirements

It is assumed you know how to open and navigate your computer using a command line prompt.

You need Java version 1.5 or higher to run the software. Verify that you have an appropriate version of java by typing at the command line:

```
java -version
```

The output should indicate that you have Java 1.5 or higher. If you do not, you can download the latest java from java.sun.com. Follow Sun’s installation instructions to install java.

You may also wish to have an image editor with which to edit the pbm files. The recommended option is GNU’s GIMP project, which you can download for free at www.gimp.org/. GIMP is easier to obtain for Linux, but windows versions exist; information is available at www.gimp.org/windows.

A.2 Obtaining and Installing Chachkee

You can obtain a compressed archive of the project from Dan Koulomzin [dkoulomzin@comcast.net]. To install chachkee, simply uncompress the archive in a directory of your choice. By doing so, you will have created a directory called chachkee in the desired location.

A.3 Using the Program

A.3.1 Windowed Mode

Running Chachkee for the First Time:

With the correct java installed, from chachkee directory, do:

```
java -classpath "../../classes/lib/bzip2.jar" chachkee.gui.ChachkeeUI
```

This will start the application and open a window. The first order of business is to start a new experiment. Notice that the window has a menu, akin to what you might have seen before in Microsoft Word or similar applications. The menu contains two items: File and Matrix. The File menu allows you to tell the program where your data is. Start a new experiment by selecting "New..." from the File menu. A File Chooser will appear and prompt you to choose a place in which to save your new experiment. Any place is fine; the program will also be putting backups of your pbm files in the directory in which you saved your experiment. When you press "OK", the File Chooser will close. You have created a new experiment.

The experiment consists of two phases. In the first phase, you will add raw data (in the form of pbm files) to the experiment by adding the data to a distance matrix. Chachkee will calculate an approximate for the Kolmogorov distances between these data as you add them. In the second phase, you will cluster the data.

Phase 1: Adding Data to the Matrix

You can add a pbm file by selecting "Add Nodes..." from the Matrix menu. A File Chooser will assist you in browsing to and selecting the files you wish to add. You may select multiple files in a single folder by holding the control key as you click on files. Keep in mind that the more files you have already added, the longer it will take to add a new file. It can take a long time to create a large matrix. When the application is done adding the files, it will show a matrix in which the columns and rows are the data files, and the values are the distances between them.

The distances are calculated using normal order traversal of the image files, and regular concatenation to approximate conditional Kolmogorov complexity.

Phase 2: Clustering the Data

When you have added as many nodes as you wish, you can begin clustering the data. The clustering algorithm is Agnes, and it uses the minimum-distance algorithm to compute distance between clusters.

The clustering is typically quick, and when it is done, a new window pops up displaying a dendrogram of clusters. Each cluster displays the number of files it contains in brackets. You can examine the cluster by clicking on the widget to the left of the cluster. Clicking the widget again will collapse the cluster to the original view. A cluster has two members. Members can be other clusters or data files. This dendrogram allows you to easily visualize the results of the clustering operation.

When you are done viewing the clusters, you can close the clustering window and return to the application to add more files or exit. You can exit by closing

the window. Be sure to save your Matrix before quitting the program. You can do so by choosing “Save” from the File menu.

Loading an Experiment:

When you open the Chachkee application a second time, you might want to revisit old work. You can load an experiment by choosing “Open” from the File menu. This will open a File Chooser. When you have selected your file, you will be able to resume work from the point at which you saved.

A.4 Command Line Mode

Command line mode is not for the feint of heart, but it rewards the brave with additional control over how the application operates. By running in command line mode, you can choose to have Chachkee compute the distances using a Z-Order as opposed to normal order traversal, approximate conditional Kolmogorov complexity using bitwise interleave rather than concatenation, and cluster the data using the average distance algorithm rather than the minimum distance algorithm.

When discussing the command line interface, we will use `courier` for options you are expected to fill in.

The command line interface works by interpreting commands you send to chachkee when you run it. You can run chachkee in command line mode this way:

```
java -classpath “../classes:/lib/bzip2.jar” chachkee.matrix.Matrix command
where command is your list of commands. A command is any of the following:
new filename traversal interleave
load filename
cluster-avg target
cluster-min target
pbm-file
```

A.4.1 Explanations of commands

new creates a new matrix that will be saved to `filename`, and which will use the traversal and interleave strategy specified. Choices for `traversal` include “chachkee.matrix.NoOrderStrategy” and “chachkee.matrix.ZOrderStrategy”. The former walks the file row by row, and the latter does a Z-Order traversal of the file. Choices for `interleave` include “chachkee.transform.DefaultJoinStrategy” and “chachkee.transform.InterleaveJoinStrategy”. The former will concatenate the files during approximation of conditional Kolmogorov Complexity, and the latter will interleave the files bit by bit.

load loads the matrix saved to `filename`.

cluster-avg runs Agnes using the average distance strategy until `target` clusters are left. The result of each iteration is printed. `target` should be a positive integer.

cluster-min runs Agnes using the minimum distance strategy until **target** clusters are left. The result of each iteration is printed. Again, **target** should be a positive integer.

pbm-file adds the given pbm file to the matrix.

The state of the matrix is always saved before the program returns you to the prompt.

Typically, you'll want to have your first command be either **new** or **load**. Then you'll want to either add data (by simply giving the filename), or cluster the data using **cluster-avg** or **cluster-min**.

A.4.2 Example

```
new example chachkee.matrix.ZOrderStrategy chachkee.transform.DefaultJoinStrategy
images/1.pbm images/2.pbm images/3.pbm cluster-min
```