WXML Quarterly Report on the Graphlopeida Project

By Sara Billey, Aaron Bode, W. Riley Casper, Dien Dang, Nicholas Farn,
Stanley (Sha) Lai, and Katrina Warner
March 23, 2017

Introduction

Our project, Graphs and Machine Learning, held under the WXML, was inspired by a paper Dr. Billey co-wrote, entitled “Fingerprint Database for Theorems” with Bridget Tenner. The initial goal of this project was to create a searchable database for graphs similar to the Online Encyclopedia of Integer Sequences (OEIS). The purpose of such a database is straightforward. Say a professor or researcher thinks they have made a new discovery and that said discovery produces a graph. Given the aforementioned database, the professor could then input a graph related to the theorem they produced and search if someone else has already discovered said theorem. If someone already has, then the professor could then find out who found the theorem and potentially other related theorems, via searching related graphs. Hence, our group aims to document certain forms of theorems from graph theory into an online searchable database, for which papers and references may be indexed by the graphs contained within them. In the process of creating this encyclopedia of graphs, or Graphlopedia, we developed a number of tools and made several design decisions for our database. During this past quarter, we created means by which our database could grow most rapidly.

Graph Recognition from Images

In this quarter, we have developed a new strategy to recognize a graph from an image. The new method overcomes an obstacle which the previous one cannot handle accurately due to the nature of the old approach. However, the new technique is not perfect: we still need to discover an efficient algorithm to have it solve more complicated cases.

Previous Work

Recall that a digital image is essentially a matrix of RGB tuples, and after we process it using some standard image analysis techniques, the image reaches a binary form: it becomes a matrix of 0’s and 1’s where 1’s indicate the contents while 0’s mark the background. A key procedure to extract information from a binary image is image thinning. To implement this step,

we applied a modified version Zhang-Seun's image thinning algorithm: besides the original algorithm, we added a feature to remove the middle pixel of any T-shape structure in a 9-pixel window. We did so to further thin the graphs. **Fig. GR01** shows an example of turning an image from its original form into a binary thinned form.

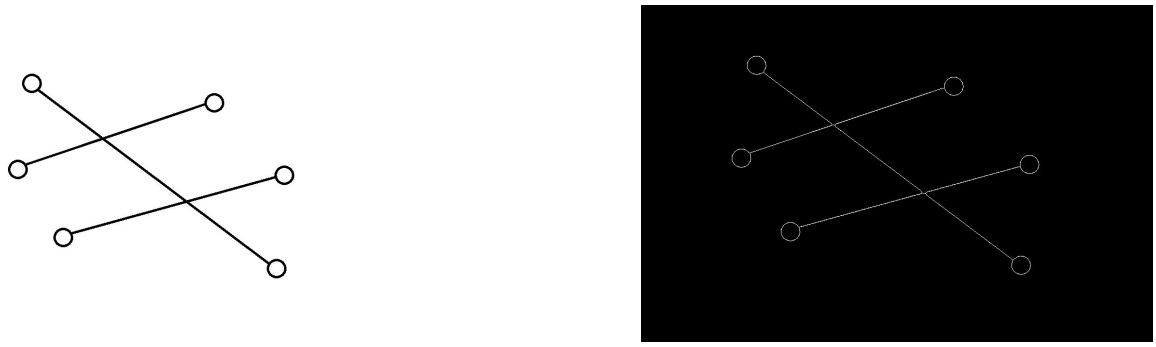
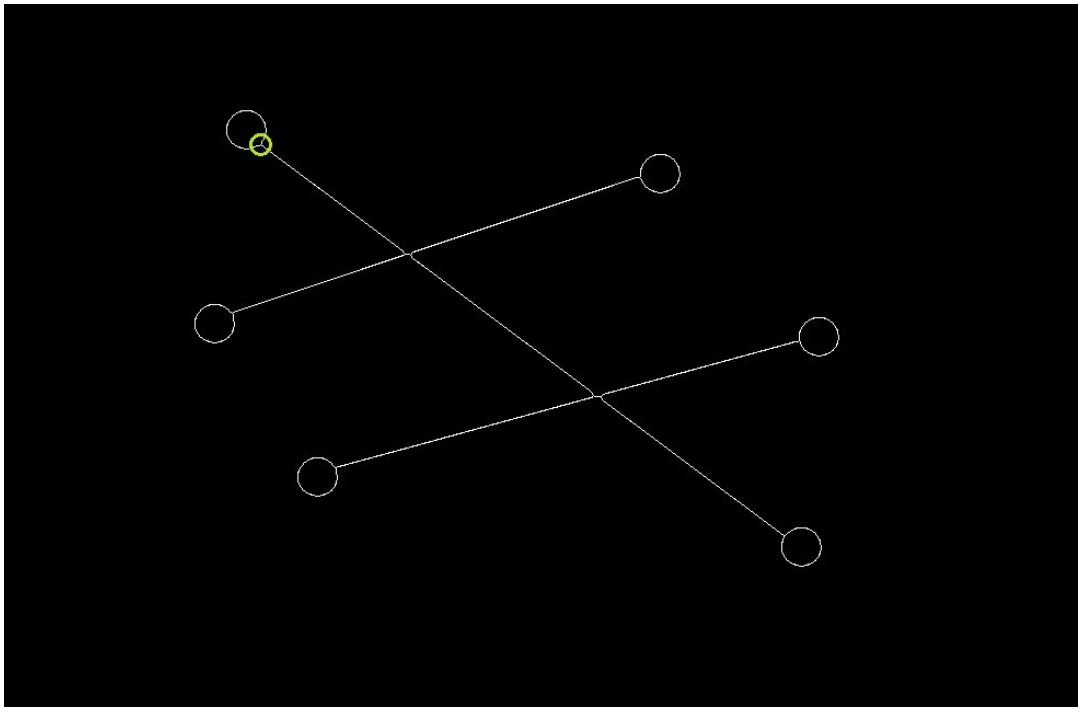


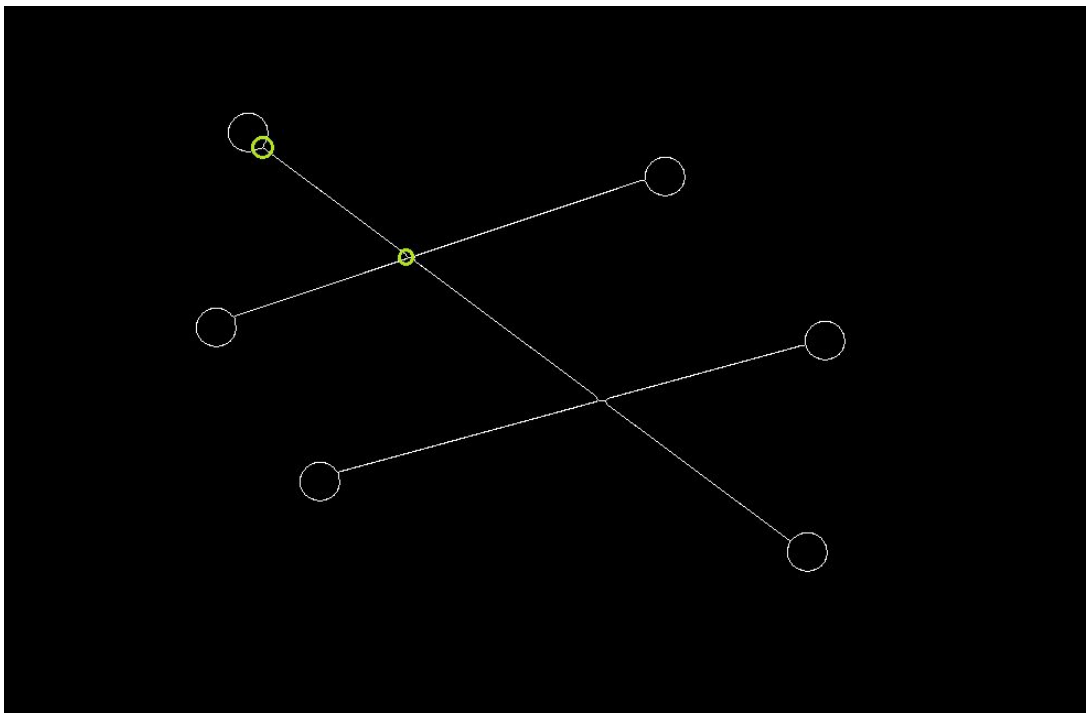
Fig. GR01 The left one is the original image while the right one is the binary thinned version.

Other than the standard image processing, there are two major parts of graph recognition: locating vertices and extracting edges. Finding vertices is simple. Although it may sound better to implement a feature to locate vertices automatically, we insist to ask the user to interact with our program to do this labor semi-manually. The reason is that the vertices in a graph can be drawn in various shapes, and we cannot find a robust way to recognize them from all images without adapting machine learning. The hard part is the edge extraction. We have been mainly focusing on this part, and we will continue doing so.

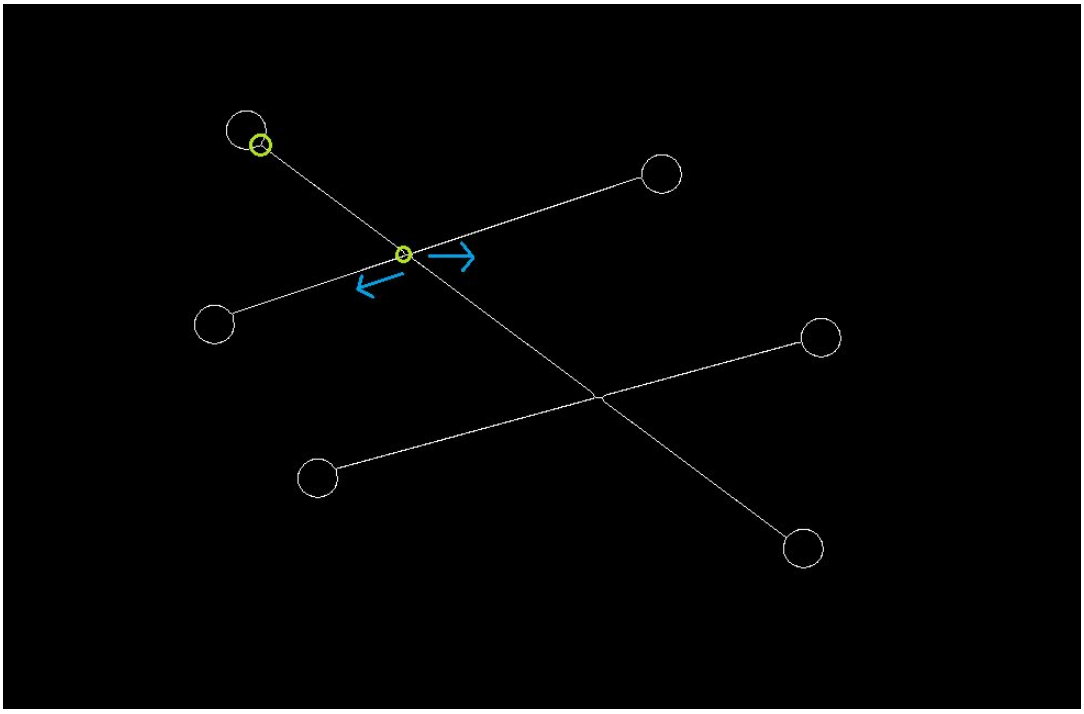
As shown in **Fig. GR02**, our old algorithm can be summarized as follows: first of all we begin from a starting point of an edge at some vertex, and then we explore the pixels on the edge one by one until we reach a point where there are more than one options to proceed; at this point we evaluate the information we have obtained along the exploration of the edge so far, and use it to determine which direction to choose. There are two possible way to determine the correct direction. One is proposed by **Auer Et Al.**. Using their approach, as shown in **Fig. GR03**, we first select a point we have been to a few steps back from the current position, and then evaluate a vector using that point and the current one, lastly we use this vector to determine which candidate vector is a better option. The other one is designed by us, as shown in **Fig. GR04**. The idea is similar, but instead of choosing one previous point, we select all the points from the previous path, and label them $[p_0, p_1, p_2, ..., p_n]$ where p_0 is our current point while p_n is the farthest one. We then build a vector for each pair of adjacent points in the list: $[p_n p_{n-1}, p_{n-1} p_{n-2}, p_{n-2} p_{n-3}, ..., p_1 p_0]$ where $p_i p_{i-1}$ is a vector pointing from p_i to p_{i-1}. For convenience, we let $v_i = p_{i+1} p_i$. We then define our reference vector $v_r = \sum_{i=0}^{n-1} w_i v_i$ where w_i's, the weights, follow some normal distribution like in **Fig. GR05** such that the closer p_i is to our current location, the more important its corresponding vector is to our decision. Finally we use v_r to determine which direction to choose.



(a)



(b)



(c)

Fig. GR02 Picture **(a)** shows that we start from some vertex, **(b)** shows that we are reaching an intersection, **(c)** shows that we have two directions to choose.

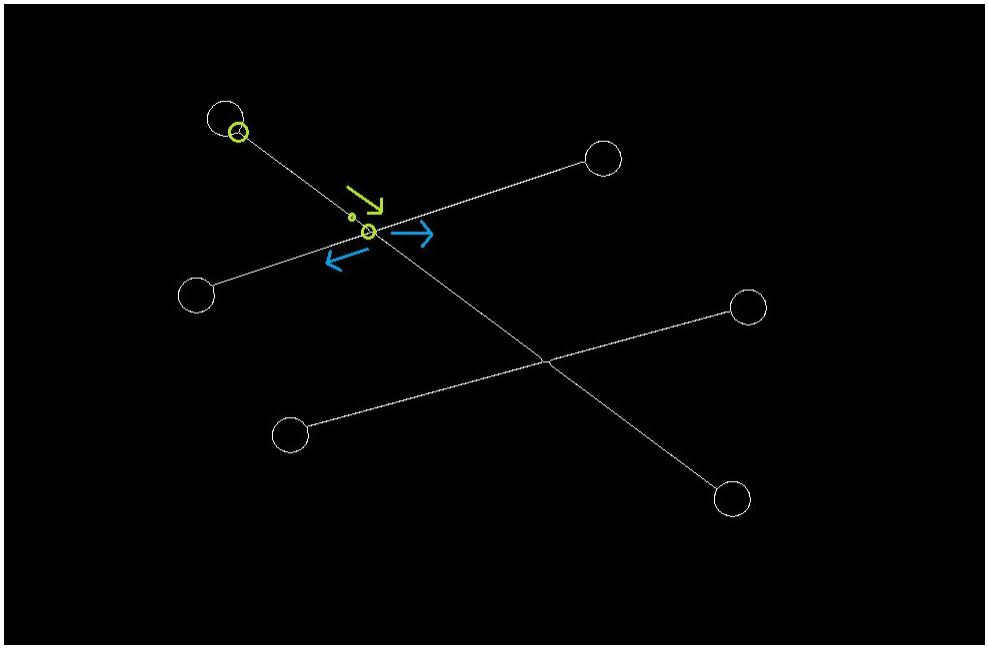


Fig. GR03 A point is selected some distance from the current one, and then vector can be evaluated using the selected point and the current one.

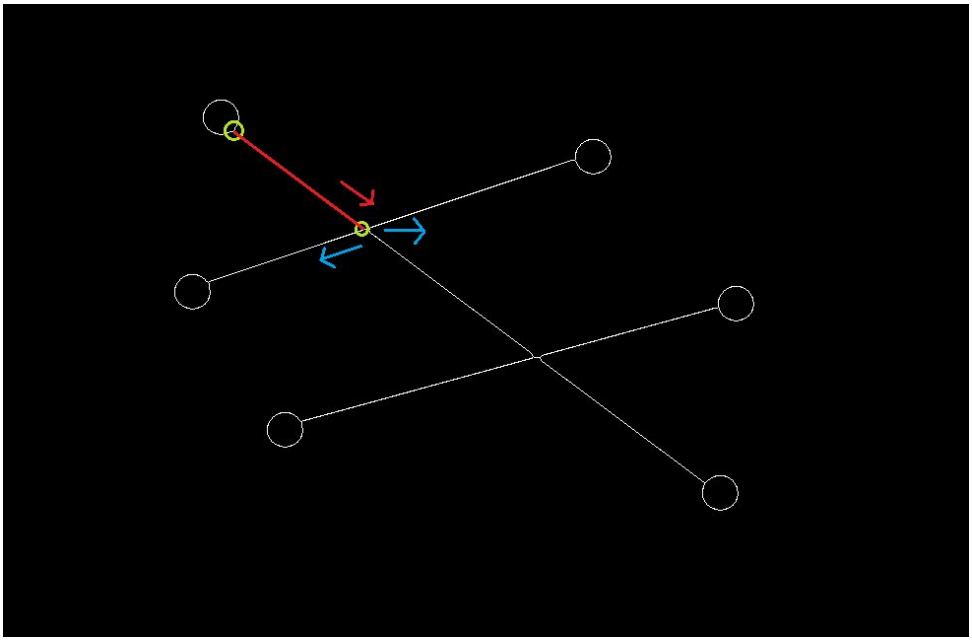


Fig. GR04 All the points from the starting point to the current place are selected, and a vector is evaluated from the weighted sum of the vectors along the path.

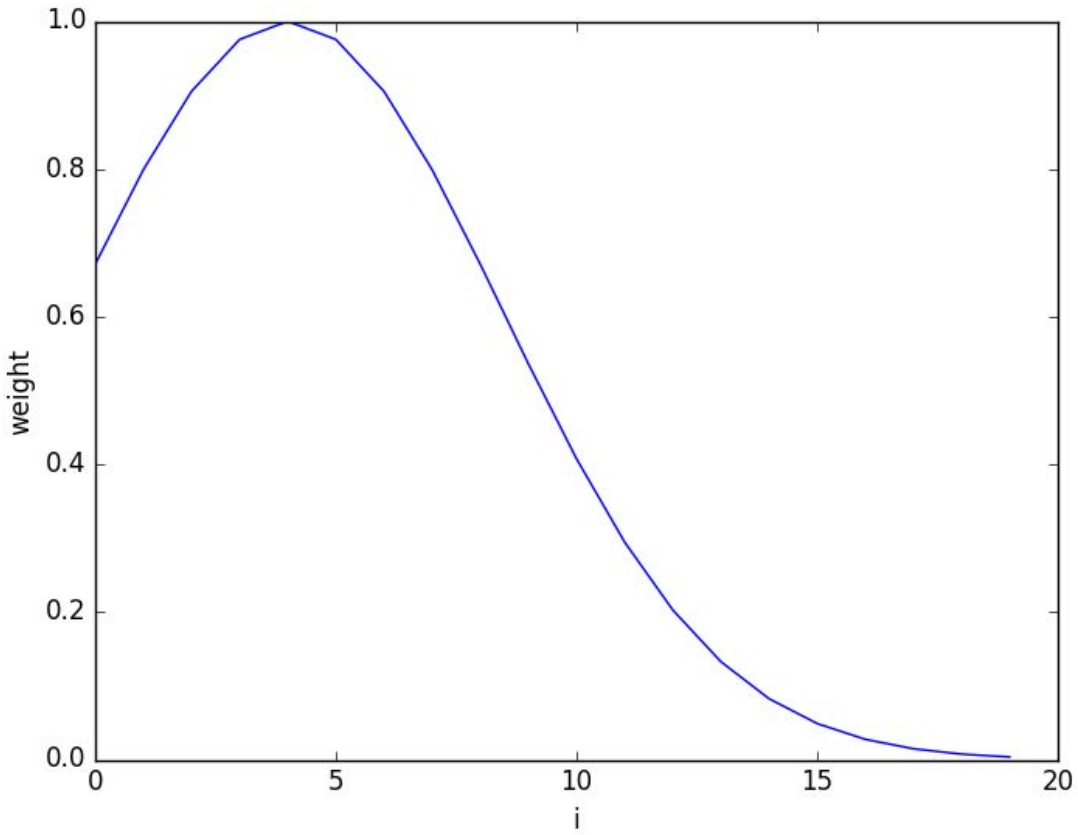
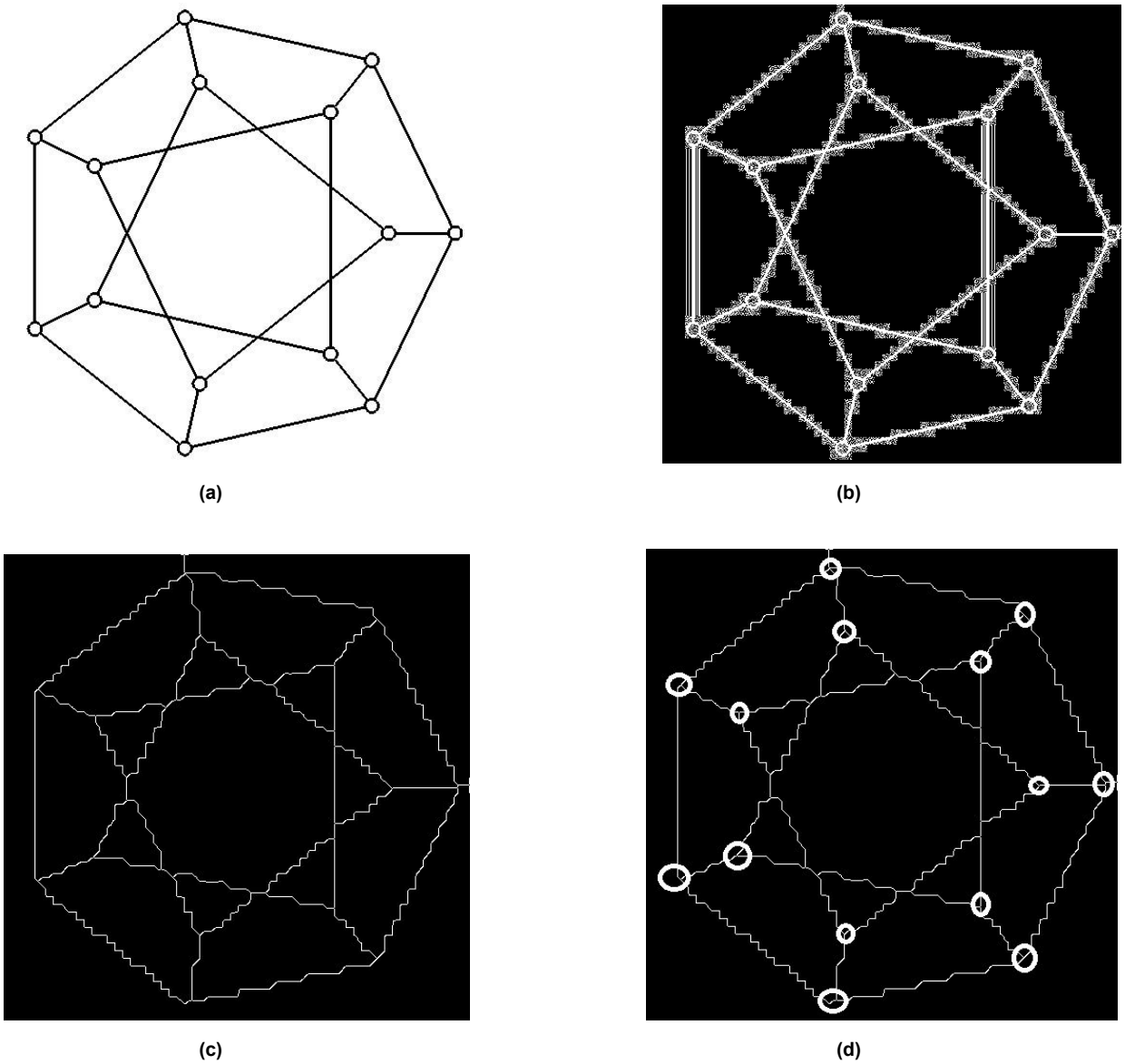


Fig. GR05 The weights follows some normal distribution.

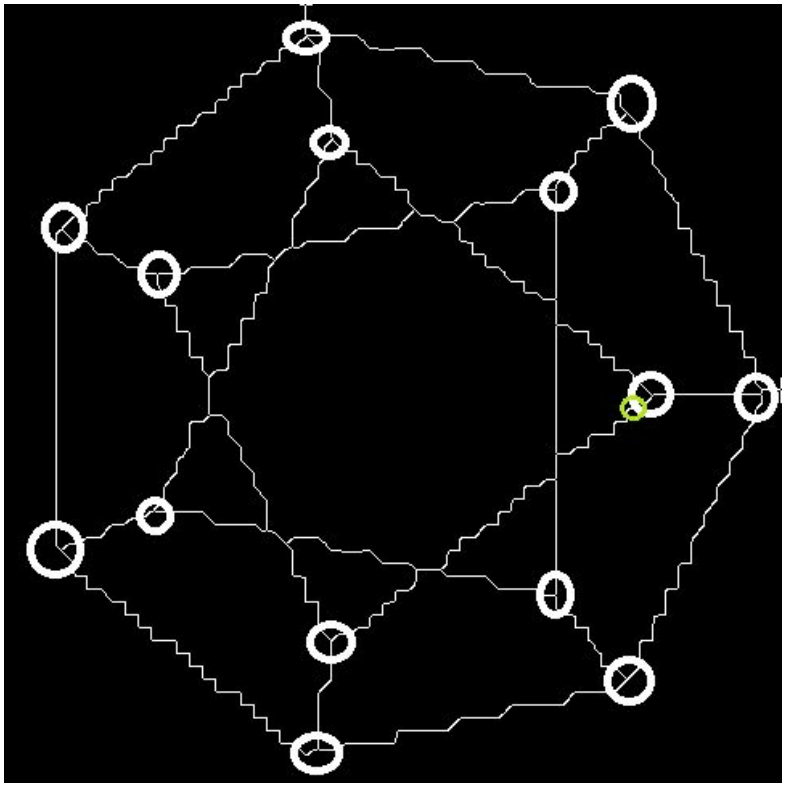
At a glance it may seem that our approach is redundant, since no matter what weights we choose, v_r will always equal to the vector pointing from the starting point to our current one. Nevertheless this is only true when the edge is a piece of a straight line. Moreover we claim that our approach is better than the other one when the image quality is low. Consider the image in **Fig. GR06**. If we adapt the two-point method, it will be possible that the previous point is badly selected and we will end up seeing an ambiguous situation in **Fig. GR07**. While if we choose our own method, the reference vector will help us identify the correct candidate, which is illustrated in **Fig. GR08**.



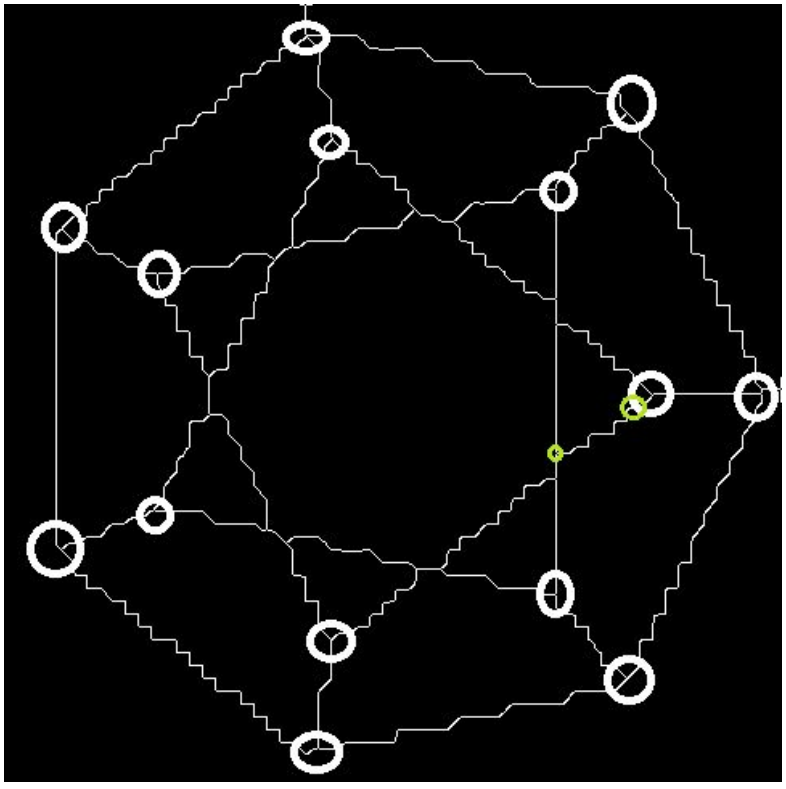
(a) **(b)**

(c) **(d)**

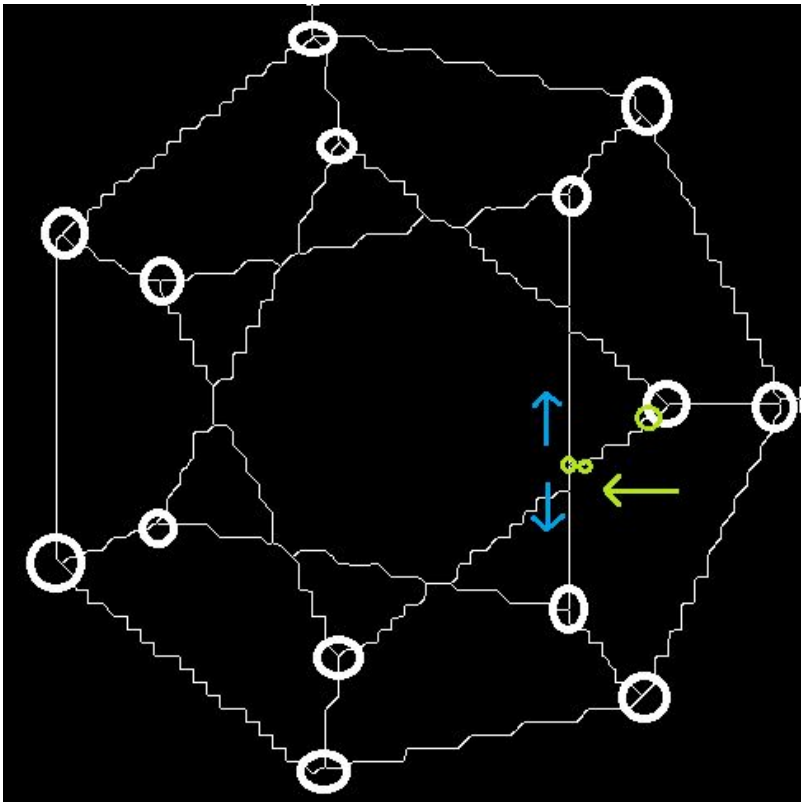
Fig. GR06 (a) is the original image, **(b)** is the binary version, **(c)** is the binary thinned version, and **(d)** is the same as **(c)** except for some artificially added ovals to indicate where the vertices are originally.



(a)



(b)



(c)

Fig. GR07 (a) shows where we start from, **(b)** shows where we at currently, and **(c)** describes a situation where we cannot make a decision.

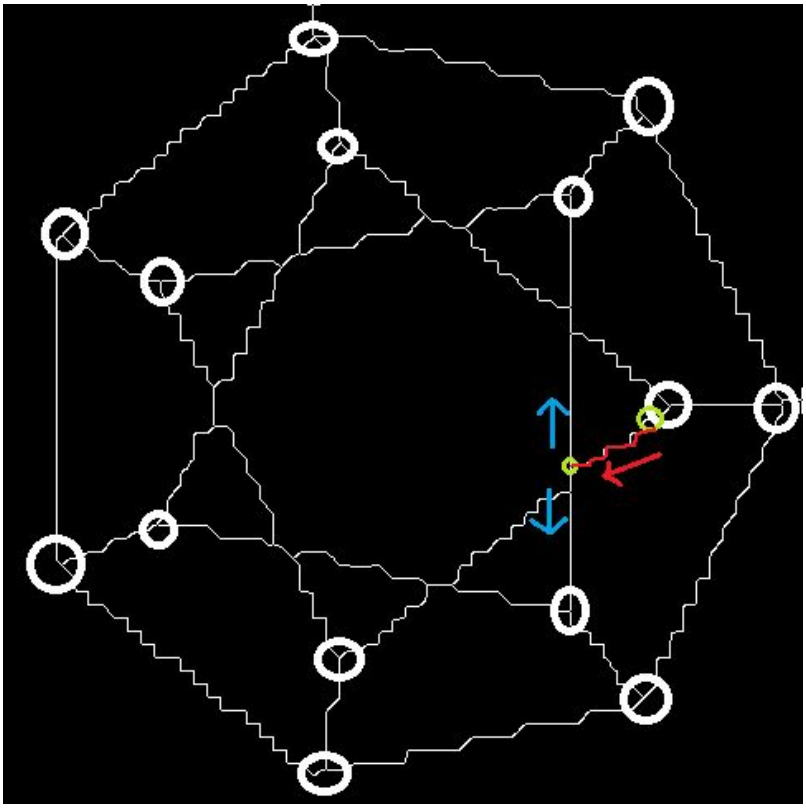


Fig. GR08 With all the points in the path being considered, it is now possible to identify the correct direction.

New Strategy

Despite the fact that our previous approach is better than the one we found in a published paper, our experiment shows that it may still be inaccurate at some situations because we do not have much control over the image thinning,. The limitation is due to the nature of the exploration: we make decisions as we go, without knowledge of a bigger picture. Using the old method, at a point where an intersection occurs, the lack of information about the entire image may sometimes cause a wrong decision. Therefore we need an algorithm for higher accuracy.

Fig. GR09 The two major structures in a thinned graph image.

The new idea comes from a key observation to the pictures we have seen. There are two types of structures as shown in **Fig. GR09**. Case A shows a structure where an edge is connecting two destinations, while Case B shows how a cross between two edges looks after being thinned. It is easy to handle Case A, but not the other one. The decision making is only needed in Case B. What makes the problem hard is essentially the part resulted from stretching a cross point, as highlighted in **Fig. GR10(1)**. We used to make a decision whenever we see such a structure, but in our new approach, we first mark the two intersections as in **Fig. GR10(2)**, and then we treat them as temporary vertices. Following this procedure, we can obtain a result like **Fig. GR11**. The good news now is that we no longer need to deal with the Case B structure in the resulting picture. Furthermore, we can use the weight function as mentioned earlier to evaluate vectors out of edges, and then store them as outgoing vectors for each vertex. In the end, we can completely get rid of the original binary image by creating a new graph for which it is possible to apply well-developed algorithms.

(1) **(2)**

Fig. GR10 Analysis of the problematic structure.

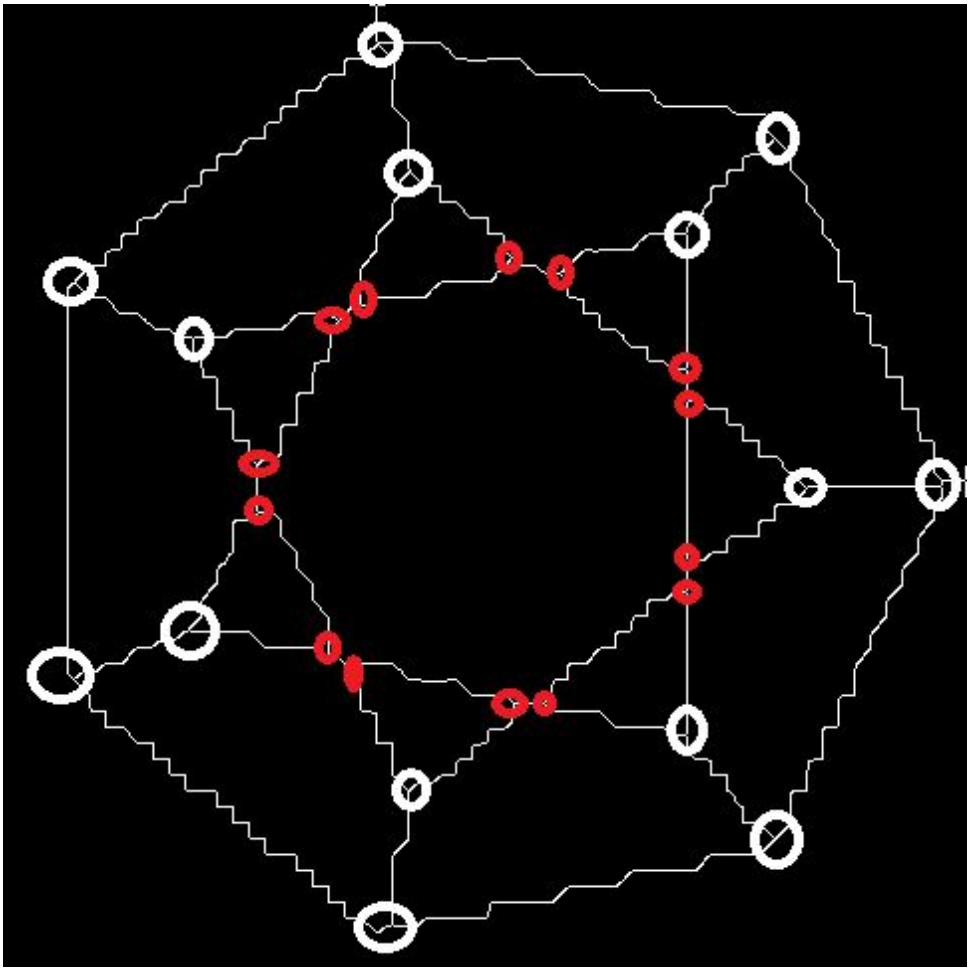


Fig. GR11 The temporary vertices, marked with red ovals, are originally intersections.

Overall, in the new graph we just obtained, each vertex is either a vertex of the original graph, or a temporary vertex we constructed from an intersection. An edge in this graph exists if and only if there is a pixel-bridge connecting two destinations which can be a vertex-vertex, vertex-intersection, or intersection-intersection pair. Moreover, each vertex is storing a list of outgoing vectors evaluated in the binary image in some reasonable way. Our goal now is to determine an efficient algorithm to restore the original graph by grouping the temporary vertices so that the stretched parts are all eliminated. **Fig. GR12** shows an example of ideal outputs.

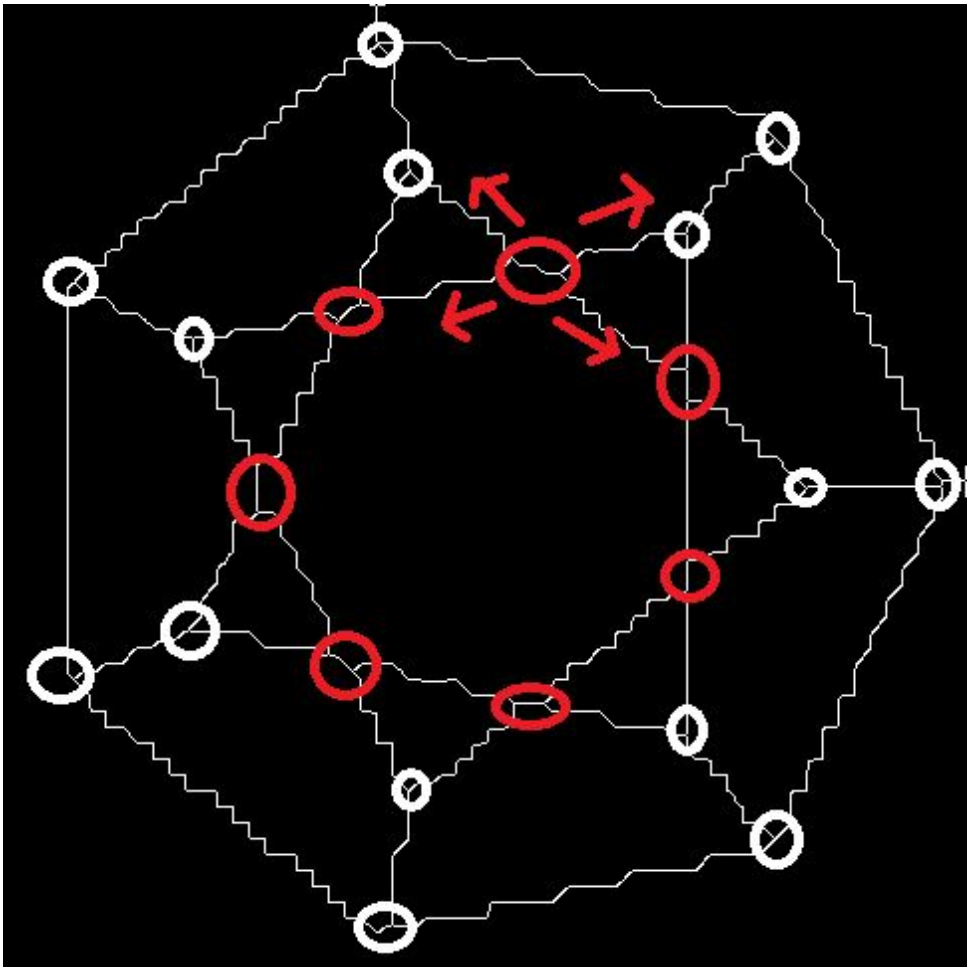


Fig. GR12 An ideal output where the temporary vertices are grouped correctly. Each group is now replaced by a single temporary vertex keeping only the outgoing vectors of the temporary vertices inside the group.

We need to define an objective function which measures the cost of grouping a certain set of temporary vertices. First we can set the cost to infinity if there is some vertex in the group isolated from the rest so that we only need to focus on valid grouping options. Then we say that the cost is low if there exists a pairing option among the outgoing vectors in the group such that the two vectors in a pair come from different vertices and the angle between them is as closed to π as possible. We then apply brute-force-search to exhaust all the results and find the one with minimum cost.

The experiments show that this new approach is highly reliable even when the image quality is low. This is not a big surprise since it is performing analysis with way more information than the previous methods do. Nevertheless, there are two problems of this new approach. Firstly, we know this method is time consuming due to the nature of brute-force-search. This issue may not be troublesome since our problem size is usually small. Secondly, we have not yet come up with a solution to the case where more than two edges are crossing at the same point. In this case, there are more than two intersections in each group when we are attempting to partition the temporary vertices. In the future, we will be focusing on solving these two problems.

Extracting Images from PDFs

In order to generate a large volumes of data, we want to autonomously locate graphs from publications. We have developed a program that can extract pixel images from PDFs, utilizing optical Graph Recognition and Computer Vision. We also can repurpose this code as another graph input method alongside strictly inputting edge lists and adjacency matrix in the future once we have a functional website.

Vectorized PDFs

Most PDFs are encoded as vector graphics, a.k.a. images and text are defined by functions or shapes rather than pixels so as to allow infinite resolution. This fact is exploited by converting PDFs to a different format, Encapsulated PostScript (EPS), which is a programming language that creates vector graphics. This is done using the linux program “pdftocairo”. When viewed in a text editor, the specific code that creates text, images, and lines etc. can be seen. We created a program that parses through the converted EPS document and extracts the lines of code that encode pixel images, lines, and nodes. The lines and nodes are assumed to belong to a vector image of a graph and is converted to a JSON format. The pixel images are placed into their own edited EPS document so as to create a PDF with for each image containing the image by itself. This PDF can be then converted whatever desired image format using the linux command convert.

Scanned PDFs

An alternative method that we developed is more general and does not require that the PDF file is encoded as a vectorized graphics. Therefore, it would allow us to process older scanned documents as well as check for errors in vectorized PDF method of extracting images. This method converts each page of the PDF document into a PNG image file. We then process each image file by applying filters like Gaussian blurs and binary imaging. Then we would use a contour detection method to retrieve all shapes in the page. Once we have the contours, we can compute properties like dimensions, area, and orientation to provide us information about whether or not the contour are text. Once we detected all of the text in the document, we want to fill the contour of the text with the background color to remove all in line text and leave the remaining figures to extract.

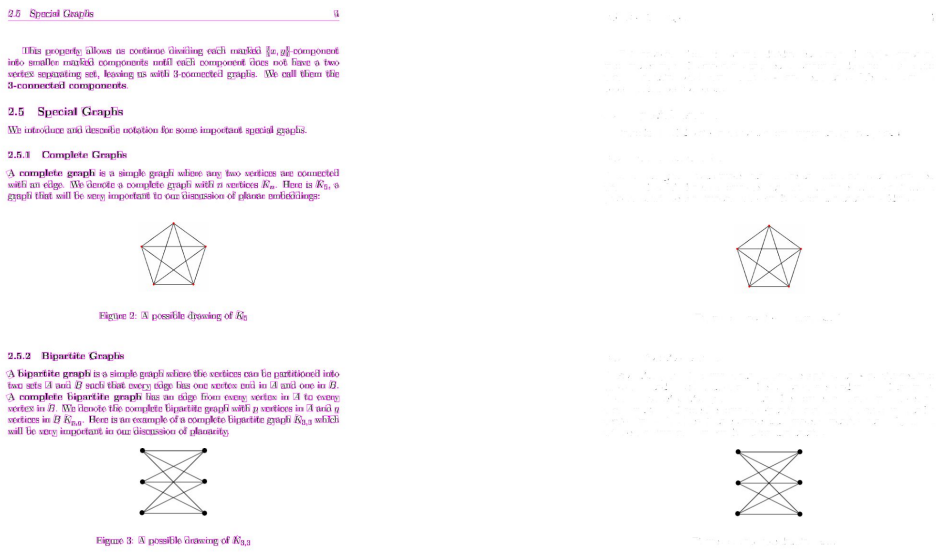


Fig. EI01 Detected contours that are suspected to be texts (highlighted in purple) and then filled in with the background color

Now that the texts are removed from the page, we need to use the contour command again to get the remaining figures. Then we would check if there are any overlapping contours, if so then we would treat those overlapping contours as part of one figure. When using this method, we get very good results for graphs does not contain multiple connected components that may confuse the contour command into thinking that it could be detecting multiple graphs.

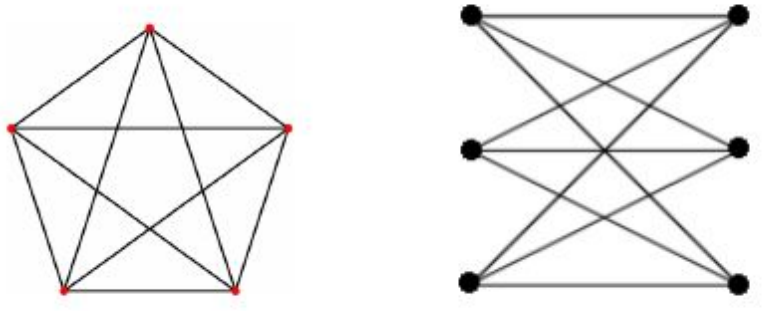


Fig. EI02 Results from applying extraction method on the example page

A future goal is to apply Optical Text Recognition (OCR) algorithms to detect English characters, Greek characters, math symbols, and other common symbols used in research papers instead

of using generic contour properties to detect and remove texts. Applying OCR would provide a more accurate extraction method, but it would also require more computational power compared to contours.

Graph Minor Theorem Characterization

As the database continues to grow in size, we hope to research means to characterize like papers. Currently, we are researching applications of the graph minor theorem as means to group papers in accord to the graph families contained within them. This methodology is promising in that the graph minor theorem already serves to characterize planar graphs by their inherent minors. In order to capitalize on this notion it is key to continue researching additional characteristic graph minors as well as develop python programs to rapidly discern key minors from the graphs indexed in our database. We hope grouping papers in this form will allow researchers to rapidly discover like papers and or theorems, facilitating research in graph theory.

Conclusion

We had several big achievements this quarter. First, a prototype version of our database is now available at https://sites.math.washington.edu/~billey/graphlopedia.html . This is a pdf document including 50 graphs which can be searched using the pdf viewer tools like command-f. One might search for a particular degree sequence or a key-word. The lead people on this effort were Katrina Warner and Aaron Bode. Second, improvements were made to the algorithm for graph recognition from images. The lead on this effort was Stanley Lai. Third, improvements were made on extracting graph images from pdf documents so we can more rapidly add entries to our database. We have high hopes of adding hundreds of new entries from the Math ArXiv to the database via this technique in the spring quarter. The lead people on this effort were Riley Casper, Dien Dang, and Nicholas Farn.

We also have some challenges remaining to address. Although our image extraction methods work effectively in generating our current dataset, they require human intervention to distinguish between regular images and those of graphs. We want to apply machine learning techniques to create a program to recognize graphs in our extracted pool of images. Training the machine learning algorithm will pose new challenges in terms of reliability and reproducibility of information from the original image.

Thanks to our new graph extraction tools, we have a system of efficiently generating many new entries. In fact, we have taken advantage of the arXiv, a publication database, and with it we were able to gain access to the original LaTeX files and extracted thousands of graphs for use

in our database. Next quarter we plan to create a functional working website (mainly on the server side) to input all of our current graphs to the database. Then we can try to work on improving the database to accept directed graphs and possibly even hypergraphs.