WXML Quarterly Report on the Graphlopeida Project

By Sara Billey, Kimberly Bautista, Aaron Bode, Riley Casper,
Nicholas Farn, Stanley Lai, Adharsh Ranganathan,
Michael Trinh, and Alex Tsun

June 23, 2017

Introduction

Our project, Graphs and Machine Learning, held under the WXML, was inspired by a paper Sara Billey co-wrote with Bridget Tenner, entitled “Fingerprint Database for Theorems”. 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 to classify graphs and implement our database online.

Graph Recognition from Images

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. Lai’s focus for the quarter has been in improving the accuracy of these algorithms.

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.

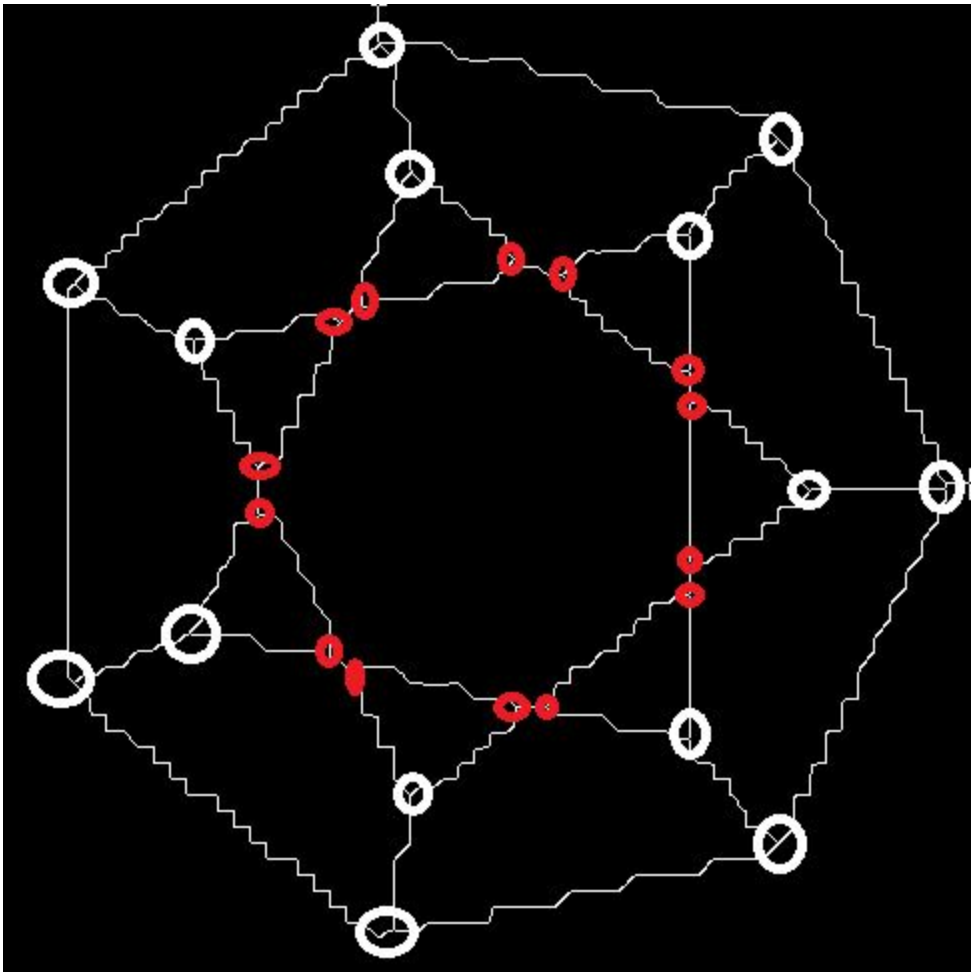

Fig. GR11 The temporary vertices, marked with red ovals, are originally intersections.

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.

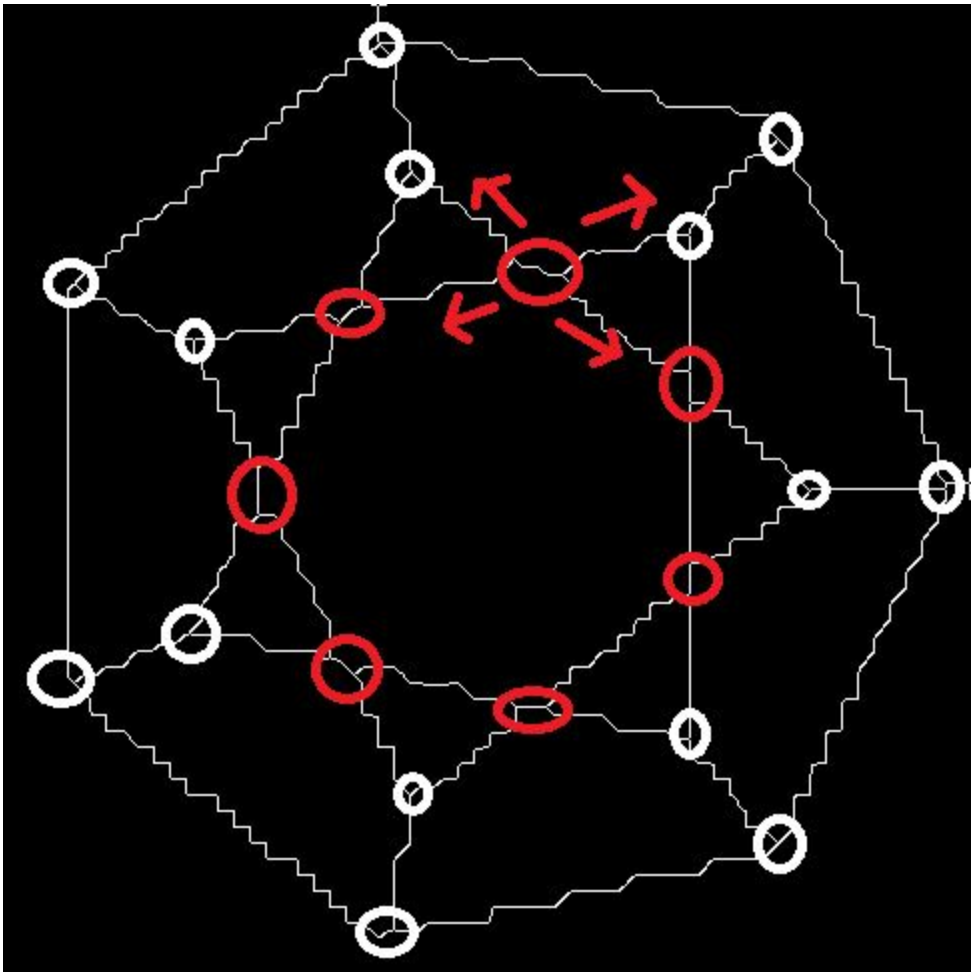

Fig. GR12 An ideal output where the temporary vertices are grouped correctly. Each group is now replaced by a single temporary vertex keeping only the outgoing vectors of the temporary vertices inside the group.

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.

Utilizing the arXiv

The arXiv is a vast database of published scientific articles from math, physics, astronomy, etc. We currently have downloaded a portion of the arXiv, consisting of compressed LaTeX and PDF files. We plan on systematically going through all articles in the arXiv and using our code to extract all images and vector graphics, determine which images and vector graphics are graph representations, and extract the graph from the representation and adding it to our database. Casper and Farn took the lead role this quarter on information extraction from the arXiv. This automation of information collection has been a huge success.

Graph Classification with Convolutional Neural Networks

The first challenge is to create a dataset for training. Our images are extracted from the arXiv, both graphs and ‘non-graphs’. These images are inputted through a program that looked for closed curves and lines between closed curves, and other criteria characterizing a scientific graph with vertices/edges. Using this method, we extract a number of graph and non-graph images in PostScript format for our training set. We then go through the "probably graph" images and manually label a few thousand examples (1 for graph, 0 for non-graph). We will use a random subset containing about 15-20% of the training set as a validation set. We use this rather than k-fold cross validation due to the training time of the CNN and lack of computational resources. The subsection below details the conversion from PostScript files to a usable format. Trinh and Tsun were the leaders on implementing CNN on graphs this quarter.

Image Preprocessing

To get our images in an acceptable format for applying machine learning techniques, we perform some preprocessing. Initially, we are given unlabelled images in postscript format. We simply loop through the postscript files in a python script and then convert/save them into JPEG files (Note: we specifically use GhostScript for the image conversion as other software had errors on older PostScript files). We then resize all of the images to have their minimum dimension to be at most M pixels (where M will depend on experiments in following sections), while maintaining the aspect ratio. We then convert all of these JPEGs into a matrix of pixels in grey scale, flatten them into a single feature vector, and write the data into a CSV. Finally, we also whitened our data, which improved our results. Since each image is of different size, we did a per-image normalization. For each image, we simply treat all of its pixels are an array of numbers. We calculate the mean and standard deviation, and standardize each image by subtracting the mean and divide by the standard deviation from each pixel. This helps a lot because there is a universal learning rate for all features, and they will all be on the same standardized scale now.

Convolutional Neural Networks (CNN's)

Our first approach for classification will be a small multi-layer convolutional neural network. For an initial experiment, we resize our images to be the same size of 28x28 and then used the example model CNN from a TensorFlow tutorial. We end up with an average of 70% accuracy. This provides us with a baseline for comparison with accuracies after optimization and tweaks.

Image Sizes

The dataset we use contains images of varying dimensions, ranging from a minimum width/height of 13 pixels to 10,000. Due to the varying image sizes, we test larger images since 28x28 could result in way too much a loss of information for larger images. We run the same experiment as the initial, but with 64x64 inputs. However, the run time ends up to be more than double the initial experiment, with no improvement in accuracy at all. It seems that either most of the images in the dataset have their main content scaled in a way to not be terribly affected by dramatic reduction in size, or the reduction in size with a square ratio is ineffective. Thus, we find that one of the issues that arises is loss of aspect ratio which may be important information, so we keep the minimum dimension to be at most 28 pixels for the following experiments while preserving aspect ratio.

Spatial Pyramid Pooling

We want our CNN to run on images of arbitrary aspect ratio (with minimum dimension M pixels), so we implement Spatial Pyramid Pooling (SPP). The only thing in the CNN architecture that requires all images to be of the same size is the fully-connected layer. We use the SPP layer to replace last Max-Pooling layer after the second convolution layer. This layer allows us to take the output of the convolutional layer and create a fixed size output, which we can then connect to the fully-connected layer as usual. SPP is still a pooling layer, but it is different only in how it applies pooling. We have pre-specified parameters {a1, ..., an}, and the output will be flattened with dimension $\sum_{i=1}^{n} a_i^2$. For each i = 1, ..., n, we partition the image into a grid of size $a_i \times a_i$. We then apply pooling to each grid section and get an output of size a_i. For example, if a_i = 2, we would split the image into 4 equal sections and apply max-pooling on each, to get 4 numbers. The SPP layer performs this pooling function for each depth of the feature map that comes out of the second convolution layer. Then, all the outputs are concatenated into a flattened array and fed into the fully connected layer afterwards. Otherwise, the network is the trained in the same way.

Results

We achieve our best results on the CNN, using 128 minimum dimension image inputs, no data augmentation, 3x3 filters for our two convolution layers, with 48 and 96 number of filters respectively, 25 Epochs, [1,2,3,4,5] SPP Pyramids, 512 features for our first fully connected layer with a keep probability of 0.55 during dropout, then map to 2 features for our CNN output. Then we introduce non-linearity with the sigmoid function, and minimize our Cross Entropy loss function using the Adam Optimizer with learning rate 10^{-4} and epsilon 10^{-5} for our best consistent results. We thus, achieve an average accuracy of 88.53% using our CNN. The best accuracy we achieved was 89.9%.

Getting Online

Another of the main tasks we were working toward this quarter was launching a live website for searching the database. In the fall quarter, we created a mock-up web application for this purpose, but we only hosted it locally prior to this quarter. As the leading provider of web hosting, Amazon Web Services (AWS) was an enticing path to our online goal. We were able to receive free credit, through AWS Educate, which eased our minds of any initial costs. The application will be run on a linux based EC2 instance, with the database build in MongoDB. MongoDB is a flexible document database software; it is easily scalable and allows new input as we expand the database. We are grateful that Bode and Bautista have taken the lead on implementing MongoDB on AWS.

We currently have all of the necessary requirements loaded on our AWS server, but the server runs python 2.7 by default. Our application is written using python 3, so we will need to set up a virtual environment on the server to run the correct version on our programs. This summer, we will be rebuilding the web application using a different development tool. It is currently built in Flask, a lightweight, python-based web framework. Flask is not optimized for scaling and long-term web use, so we will build a new version in a more traditional way for easier implementation.

Error Checking

As we move toward hosting a bigger broader database, we need to make sure all of our information is accurate, well written, and not redundant. Information integrity is critically important to our mission to create a useful database of graphs for researchers and students around the world.

We have started implementing some error checking procedures. Some of the currently automated checks include matching degree sequences to edge sets, and other isomorphism invariants. Using Sage and the McKay canonical form, we can now easily detect isomorphic

graphs up to a reasonably large size. A more difficult check is to confirm that a graph is the actual graph that wants to be inputted, that is really does appear in the cited papers, and that it is an “interesting graph”. Thankfully our graph recognition and extraction programs have become very accurate and can be used to assist this, but some of it requires human intervention. We have started including more images in our database entries. We use both the stock image, produced by the publication, and an embedding, created by one of our programs. Billey is currently leading the charge on database editing along with recent help from Zach Hamaker.

Conclusion

We had several big achievements this quarter. First, since the website is not quite live, a prototype version of our database - complete with images! - is now available at https://sites.math.washington.edu/~billey/graphlopedia.html . This is a pdf document which can be searched using the pdf viewer tools like command-f. One might search for a particular degree sequence or a key-word. Second, improvements were made to the algorithm for graph recognition from images. Third, we have a new tool to classify graphs using machine learning, which will help automate database expansion. We are hopeful to increase the size of the database rapidly now that we have a well defined set of tools. Lastly, we are on the verge of launching our interactive, dedicated website for the database. With many error checking procedures in place, we are nearly ready to implement the database at full scale.

Our future plans include adding different types of graphs like directed and hypergraphs. Furthermore, we wish to implement features such as graph families and more searchable graph invariants. All of these tools will make the website more user-friendly, and make finding the right graph easier.