Computer Systems Lab Senior Research Projects
2007-2008
2nd Quarter vers.

TJHSST

February 29, 2008
Contents

Exploring Artificial Societies Through Sugarscape 6
Jordan Albright

Breaking a Visual Captcha 9
Tianuhi Cai

Excursions into Parallel Programming with MPI 14
Michael Chen

Exploration of Genetic Algorithms Through the Iterative Prisoner’s Dilemma 19
Aaron Dufour

TJHSST Website Redesign 21
Martin Elthon

Analysis of Runner Biomechanics Through Image Processing 22
Asa Kusuma

Particle Swarm Optimization and Social Interaction Between Agents 24
Kenneth Lee

Genetic Algorithms to Find Near Optimal Solutions to the Traveling Salesman Problem 27
Karl Leswing

JLSim - Visual Microscopic Traffic Simulation Application with Extensive User Interface 29
Jinyu Liu

An Interactive, User-driven Physics Simulator 35
Tom Smilack

Applications of Parallel Programming 38
Alex Valentin

Crowd Dynamics Artificial Intelligence with Particle Swarm Optimization 41
Keith Ainsworth

Elementary Education in a Technology Age 43
Gregory Gates

Development of a Generic Font OCR 46
Nathan Harmata

Using Genetic Algorithms to Optimize the Traveling Salesman Problem 50
Ryan Honig

Three Dimensional Collision Detection for N Solids Using OpenG 53
Richard Hooper

Prisoner’s Dilemma with N participants and Optional Cooperation 55
Matt Lee

Pathfinding Algorithms for Mutating Graphs 57
Haitao Mao

Sugarscape: An Application of Agent Based Modeling 59
Andy Menke
Excursions into Logic Based Computation using Conway’s Game of Life
Liban Mohamed

Use of Various Techniques Implementing Procedural Generation and Rendering in Video Game Design
Justin Warfield

Interactive Geometry in 3D
Jacob Welsh

Development of a German-English Translator
Felix Zhang

Study of the Evolution of Organism Combination
Nicholas Brown

A Dynamic Model of Human Populations
Joshua Choi

Programming a New Sugarscape
Patrick Coleman

Creation of an Air Traffic Simulation Using Agent Based Modelling
Sam Eberspacher

Dynamic Image Resizing
Patrick Elliott

Agent Based Simulation, Negotiation, and Strategy Optimization of Monopoly
Nicholas Loffredo

The Applications of Image Processing Techniques to Sign Language Recognition Through a Web Camera Interface
Byron Hood

C to Java Programming Language Translation
Jamie McAtamney

Analysis of spectro-temporal receptive fields in an auditory neural network
Madhav Nandipati

Accurate 3D-Modeling of User Inputted Molecules Using a Hill Climbing Algorithm
Ben Parr

Advanced Automobile Recognition Through the Use of Image Processing Techniques
Andrew Stebbins

Cayley graphs formed by conjugate generating sets of Sn
Jacob Steinhardt

Conformal Mapping Using the Schwarz-Christoffel Transform
Evan Warner

Implementation of Image Deblurring Techniques in Java
Peter Chapman

Java Decompiler
Joshua Cranmer

3
Simulation of the Spread of a Virus Using Agent Based Modeling
   Matt Wade

Simulating Evolution
   Tasha Wallage
Projects

3rd Period
Abstract

Agent based modeling is a method used to understand complicated systems through the simple rules of behavior which its agents follow. It can be used to explain simpler systems, such as the pattern in which birds fly, to more complicated systems, such as self-segregating neighborhoods (Macy, 2001). Though the systems resulting from the interactions of the agents are not perfect replicas of more complicated societies, they lend insight into the way in which they develop. One common application of agent based modeling, Sugarscape, developed by Epstein and Axtell, creates an environment where agents follow simple survival rules within their society. Sugarscape allows for analysis of a variety of trends resulting from the agents interactions, among which is wealth distribution, and is a useful tool for social science.

 Keywords: agent based modeling, wealth distribution, Sugarscape, social science

1 Problem Statement and Purpose

Agent based modeling, a bottom-up method of modeling complex situations, has become a useful method for simulating problems in the field of social science. The agents, the main building blocks of the model, are designed to follow a set of rules or guidelines. Their interactions result in a more sophisticated global result. This approach programming lends itself naturally to social sciences because of simplistic way in which it creates societies through its components.

One common simulation using agent based modeling is Sugarscape, designed by Epstein and Axtell (1996) which is comprised of a set of agents who make calculated moves through a sugarscape a landscape with varying amounts of sugar, a renewable source of energy for the agents. The agents, limited by vision, move around the sugarscape grid gathering sugar for energy. As time goes on, the agents continually gather the sugar, gaining energy, may reproduce, and eventually die. Some of the agents are endowed with better vision than others, and tend to be more successful than other near-sighted agents. This, and other factors, such as initial placement, creates an unequal distribution of wealth among the agents.

This behavior, though different for each simulation of Sugarscape, follows certain trends. These trends of wealth distribution naturally lend themselves to analysis using functions common to income distribution and disparity studies. Three functions that lend themselves to this type of problem are the Lorenz curve, the Gini coefficient, and the Robin Hood index (sometimes called the Hoover index).

Although Sugarscape and other forms of agent based modeling lend themselves to social sciences, because the results of such simulations focus on simpler interactions among the agents, with simpler global results, rather than complicating the interactions in favor of more realistic outcomes, sociologists are somewhat hesitant to rely on agent based modeling, favoring differential equations instead (Macy, 2001). More research, such as the reliability of statistical analysis of the results of the interactions of agents in Sugarscape, needs to be done before agent based modeling will be used more widely in sociology.

2 Background

The application of agent based modeling, specifically Sugarscape, to study wealth distribution and disparity has been undertaken by a number of researchers in economics and social sciences. Sugarscape does not model a typical modern society of today in which production and skill acquisition are factors in the success of agents, but rather more closely models a hunter-gather society in which gathering and trade are the way in which agents accumulate wealth in the form of sugar.

In "An Agent-Based Model of Wealth Distribution", Impullitti and Rebmann used a Netlogo version of Sugarscape to look at wealth distribution from both a classical and a neo-classical approach to...
economics. Impullitti and Rebmann found that inheritance of non-biological factors increased wealth distribution while inheritance of biologically based factors decreased it. Kunzar did a similar analysis of wealth distribution, though the analysis was heavily concerned with the trend of nepotism. "Simulating the Effect of Nepotism on Political Risk Taking and Social Unrest" showed that descendants of the wealthiest tended to become second class citizens and that the descendants of the lowest class remained so.

Many agent based modeling problems, such as the Impullitti and Rebmann version and this particular problem using Sugarscape, are programmed using NetLogo.

This program uses three main functions to show wealth distribution: the Lorenz curve, the Gini coefficient, and the Robin Hood index.

The Lorenz curve shows what percent of the population owns what percent of the wealth. It is usually compared to a line of perfect equality, in which 10

The Gini coefficient is derived by comparing the area between the Lorenz curve and the line of perfect equality to the integral of the line of perfect equality. It ranges from 1 to 0, with 1 representing perfect inequality, and 0 representing perfect equality.

The Robin Hood index is the greatest vertical distance between the Lorenz curve and the line of perfect equality. Also called the Hoover Index, this is proportional to the amount of wealth that would need to be taken from the rich and given to the poor for perfect equality to be achieved.

3 Research Theory and Design Criteria

The Sugarscape agents behaviors are specified by a set of guidelines. One of these guidelines involves searching for food: in each timestep, each agent determines which patch or patches of the Sugarscape would be the best place to move. This is done within each agent's scope of vision, a number specified by the user (usually between 1 and 10 patches). The agent looks north, south, east, and west, that far in its vision and determines the patches with the most sugar that is not already occupied by another agent. Then the agent randomly selects one of the best patches and moves to that patch. This is done by each agent individually, rather than simultaneously, to prevent two agents from occupying the same patch. The agent then gathers all sugar on the square, which it stores as energy, and subtracts from its energy stores various unit of energy for metabolism, which varies randomly from turtle to turtle and one unit of energy for each square forward it moved from its previous location.

At each timestep, the agent may also reproduce. This occurs if the agent has enough energy to do so: this amount of energy (between 1 and 100 units of energy) is determined by the user. If the agent reproduces, it subtracts the birth energy from its energy store, and another agent is hatched on the same square as the agent.

At each timestep, the agents may also die. This happens either after 80 timesteps to simulate death due to age or if an agent cannot maintain a positive amount of energy.

Each timestep, the amount of sugar in the patches adjusts to reflect the consumption by the turtles. If a turtle moves to a specific patch, that turtle removes all sugar energy from that patch. Every other timestep, patches regrow their sugar by one increment.

While the turtles are moving throughout the Sugarscape, a number of different mathematical analyses run in the background and graphical representations of these analyses are shown as well.

3.1 Algorithms

This version of Sugarscape utilizes three different algorithms to analyze wealth distribution: the Lorenz curve, the Gini coefficient, and the Robin Hood index. Both the Gini coefficient and the Robin Hood index are derived in relation to the Lorenz curve, but they offer different information regarding wealth distribution.

The Lorenz curve is usually plotted in relation to the line of perfect equality. The line of perfect equality describes a population whose wealth is distributed evenly among individuals. For instance, ten percent of the population would own ten percent of the wealth, fifty percent would own fifty, and so on. The Lorenz curve plots the actual distribution of the wealth. For instance, sixty percent of the population may own forty percent of the wealth, and seventy may own forty-five percent. The Lorenz curve is usually calculated using the cumulative distribution and the average size.

The Gini coefficient represents the ratio of the area of the Lorenz curve to the area of the triangle of perfect equality (the integral of the line of perfect equality). It is usually calculated using the mean difference between every possible pair of data points.

The Robin Hood index represents the amount of wealth that would need to be redistributed taken
from the wealthy individuals and given to the poorer ones) in order for there to be perfect equality. It is calculated by finding the greatest vertical distance between the Lorenz curve and the line of perfect equality. Robin Hood index is also a good indicator of public health, though that is not the purpose for which it is used here.

This is a fairly simple version of sugarscape. However, at the moment, the level of simplicity is best because the movement of the turtles coupled with the mathematical analyses creates a very slow program. This is especially so if the turtles are reproductively successful, and the number of turtles increases to 500 or more turtles.

The graphs produced during a typical run of this Sugarscape are indicative of a typical free trade society. This is especially true of the Gini coefficient results. The Gini coefficient typically falls between .4 and .5, which shows an average wealth distribution. That is, there are clear divisions, but the wealthy class does not completely control the wealth of the society.

4 Expected Results

The goal of this project is to provide insight into how wealth is distributed in a free trade society. The society is limited in its production and resembles more of a hunter-gatherer society in which each agent gathers as much food as it can. This model is developed using a Sugarscape society written in Netlogo, whose agents are limited by age, metabolism, and vision.

Though this project is beginning to mathematically show the relative wealth distributions, more analysis needs to be done before the data provided is meaningful. Though there is analysis of the wealth distribution of this particular Sugarscape, it may need comparison to other analyses of similar problems before the data can be useful.

This project and others like it is attempting to make simulation models more useful to social sciences. Small disturbances and changes in initial conditions can be quickly quantified here, and though the resulting interactions are much more simplistic than real interactions in societies and organizations, the insight taken from simulation models can be used to make improvements in real societies and organizations.

5 Bibliography


Tianushi Cai

Abstract

CAPTCHAs, Completely Automated Public Turing tests to tell Computers and Humans Apart, are tests to determine if a user is human. They are often found on registration webpages to prevent automated signups and spam. The goal is that its challenges are easy for humans to solve, but difficult for computers to solve. Common variants are audio and visual CAPTCHAs, which consist of an image with letters or numbers that are to be typed in to a form by the user. The goal of this project is to devise a system to break a particular CAPTCHA using image processing, optical character recognition, and an artificial neural network.

Keywords: CAPTCHA, turing test, median filter, optical character recognition, OCR, neural networks, computer vision

1 Introduction

Completely Automated Public Turing tests to tell Computers and Humans Apart (CAPTCHAs) are tests to determine if a user is human. They are often found on registration pages, such as when registering for an email address at Gmail. It is sometimes referred to as a reverse turing test, since it is a test administered by a computer to distinguish whether the user is human or computer, rather than a test in which a human questions whether a user is human or computer. It is a system with challenges that are easy for humans to solve, but difficult for computers to solve.

Websites often use variants such as audio and visual CAPTCHAs. Visual CAPTCHAs consist of an image with letters or numbers that are to be typed in to a form by the user. Often, the letters are moved and rotated, noise is added, and the image is distorted, depending on the specific CAPTCHA. In this case, the problem may be solved by the computer, using computer vision techniques, as a combination of image processing and character recognition. The image is extracted from the web page, background clutter (such as noise) is removed, segmentation (separating letters) is performed, and the letters are identified.

The goal of this project is to break a visual CAPTCHA - specifically the one at captchas.net, which is provided for free. It is a black and white image with 6-16 letters (depending on specifications) that are all lowercase, rotated, and translated. Black and white noise is also added.

There are two parts in the project: getting rid of noise as well as other image processing (segmentation), and the use of the artificial neural network.

1.1 Background

A CAPTCHA's purpose is to distinguish between a human and a computer by presenting a challenge that is easy for most humans, but difficult or impossible for a computer. Visual CAPTCHAs present a user with an image with letters or numbers, and the user is to enter the letters or numbers into a form. The image is often constructed such that the letters or numbers are distorted, or contain noisy data. However, visual CAPTCHAs often do not stand up to this challenge. Many have been broken in the past, and this research project will attempt to do something similar by using image processing and character recognition, along with a neural network. Alternatives for the more easily broken visual captchas are ones that include logic (such as a captcha with a math problem), logic questions without an image (though it would take a large database of questions and answers), a classification problem (showing a user a variety of pictures depicting the same thing, and expecting an answer), or an auditory CAPTCHA (in which the user listens to a file and transcribes the words).

The goal of this project is to break the captcha at captchas.net, which provides a free CAPTCHA service. The CAPTCHAs they provide are in black and white, with six or more (user-defined) lowercase letters of the same font. They are rotated and translated, and the image has a significant amount of black and white noise.

The noise will be removed using a variation on a median filter. The letters will be separated using a flood-fill. The letters will be identified using an artificial neural network.
If there is time, CAPTCHAs from sources outside of captchas.net will be used.

The artificial neural network will be used to match the black and white image of a letter to the letter itself. Neural networks are important in that they can model highly complex, nonlinear systems and can be proficient in classification and pattern recognition.

Before using the neural network to classify images, it must first be trained. In this case, inputs of image and letter pairs will be used to train the network. The actual letters of the CAPTCHA can be identified using a formula provided by captchas.net, as the letters in the image correlate to a function of the address.

Training, in this case, will be done through backpropagation. Backpropagation is a standard neural network supervised training algorithm.

In an artificial neural network, a group of simple neurons are used to display a more complex behavior as a group, which is determined by their connections and parameters. There is generally an input layer, an output layer, and possibly many hidden layers. The input layer takes in data, the output layer spits out data, and hidden layers do intermediate processing. Neurons fire with a value between 0 and 1; when a neuron receives input, it weights the inputs by neurons connected to it, and determines whether or not to fire by the sum of the weighted inputs. A backpropagation network, in short, starts out with a set of randomized weights, and adjusts the weights on each layer depending on the error produced. In this project, the neural networks will use supervised learning. This is where the neural network is trained with a set of example pairs, and where the goal is to find a set of weights that can match the pairs.

Research on neural networks has been in existence for several decades. In particular, the use of neural networks for classification has been used.

Although neural networks do not accurately represent their biological counterparts, they have been shown to successfully classify images, as long as a large enough training sample is available.

Greg Mori and Jitendra Malik, from the UC Berkeley Computer Vision Group, broke the Gimpy and EZ-Gimpy CAPTCHAs. Their research is documented in their paper, Recognizing Objects in Adversarial Clutter: Breaking a Visual CAPTCHA, as well as in their website at http://www.cs.sfu.ca/ mori/research/gimpy/. The challenges they faced were much harder than the ones in my project, as EZ-Gimpy included severe distortions, varied fonts, and a really noisy background. In addition, GIMPY contained multiple pairs of words overlapping each other, in the same color. Because their challenge is different, the techniques they use are not all applicable to my project. For instance, I do not have to deal with overlapping characters, nor do I identify entire words. However, much of the beginning process is similar: remove noise, separate out letters, and identify letters.

Handwritten Digit Recognition with a Back-Propagation Network by Y. Le Cun et al at AT and T Bell Laboratories used a large back-propagation network to read hand-written zip codes. It took inputs in the form of black and white images. The project focused on the effect of architecture on the ability of the network to recognize digits. Its architecture was modeled in such a way that not all neurons from each layer were connected; there were five layers in total, with some neurons receiving only local input. This architecture performed well in identifying the handwritten digits, regardless of position.

1.2 Expected results

Over the course of the next few months, a basic neural network will first be built. Then, more advanced and complicated neural networks will be created from that. Research, design, programming, and testing will all be done throughout the year as different networks will be created. Handwriting data will be needed for testing and training neural networks. For the language, JAVA will be used, and Eclipse or JGrasp would be used for software.

The goal of this project is to develop a program that will break visual CAPTCHAs, such as the one at captchas.net. The goal is also to learn more about different types of image processing and the steps in OCR, as well as how neural networks work. It is also to gather evidence of how this is best done.

1.3 Type of research

The type of research in this project is use-inspired basic research, to pursue fundamental understanding but motivated by a question of use (Pasteur’s work on biologic bases of fermentation and disease).

2 Procedures and Methodology

The project will be deemed successful if the neural networks created can successfully identify handwritten characters.

To test if the program works, a neural network will be trained with test data. Then, the network
will be tested with data that was not from the training sample. If the program can accurately identify the characters, it will be successful.

To aid in the portrayal of the data, the weights of the connections between neurons can be continuously printed out, and then graphed, possibly using GNUplot.

To test the performance of a neural network, it will first be trained with a set of images. Then, it will be run to identify another set of test images. If it identifies the test images correctly, it will have performed correctly.

The requirements for the program would be to create a functioning neural network, and preferably with one that could correctly identify handwritten characters.

3 Development

Two sections - neural networks and image processing - have been worked on in the past two quarters.

3.1 Neural Network

In the first quarter, a working back-propagation neural network class was created in Java. A neuron layer class was used, and a neuron class was used for the neuron layer class.

The final back-propagation neural network class was written with three layers - one input, one output, and one hidden. Each node in each layer is connected to each node in the next layer; the number of nodes in each layer varies depending on constructor input.

Neurons from the input layer would receive input, and pass forward a signal to the neurons in the hidden layer, which would pass a signal to those in the output layer. The output of each neuron depends on the output of the neurons that feed into it, as well as the weights that connect it to the neurons that feed into it.

Then, once an output is obtained, it is compared to the correct output. The network then uses backpropagation to adjust the weights in the layers in order to obtain a more correct solution the next time around. The speed at which the weights conform to the input is dependent on the learning rate, which is also specified.

The neural network was tested initially for binary operations, such as AND, OR, and XOR.

Later on, the network was adjusted to learn rudimentary numbers. The inputs, as always, were numbers – each pixel in the number image was a 0 or a 1, representing white and black, respectively.

These "images" were actually text files with these numerical representations.

The neural network succeeded in both instances. Although the testing was not a lot, it did demonstrate that it worked.

In the future, the weights of the neural network will be able to be saved and re-loaded so that it can be trained multiple times without starting from random weights.

3.2 Image Processing

During the second quarter, the image processing side of the problem was worked on. Java, again, was used, despite the fact that C or C++ may be better for dealing with images. However, it would be cleaner to use one language for the whole project, even if the output of the image processing is a text file of numbers (regardless of language). Luckily, Java’s ImageIO classes and BufferedImage classes helped facilitate this.

The first step was to retrieve the images. ImageIO has a command for retrieving an image from the internet, bypassing wget and other terminal commands to download images from the internet. In addition, ImageIO allows loading a file into a BufferedImage, which is convenient.

After that, I wrote a method to convert the BufferedImage into an int[][] array, which would be easier to work with, as well as a method to convert the int[][] array back into a BufferedImage (to save). I then wrote filters for the image, such as a black and white filter, which takes a threshold and makes everything under that threshold one color, and everything other that threshold another color. In addition, I wrote a modified median filter. Rather than taking the median of a large area, it takes the median of the 3x3 square surrounding a pixel, since noise is only on the scale of pixels. It is not a median filter in another way: it has the options to take the average of the middle three, five, seven, or all nine neighbors, rather than just the straight median. It was also written to be able to pick not only the median value, but also the least, greatest, second-greatest, and so on - as specified by an argument in the method.

This was then tested on images retrieved from capchas.net.

Surprisingly, the extra features written into the median filter were useful. I chose to use a median filter rather than a gaussian blur for two reasons: a gaussian blur requires more coding (because of the statistical aspect), and a gaussian blur makes a shape lose the definition in the edges.

The fact that not the exact median can be chosen
also became useful to remove mostly dark-colored noise, because there was a lot of it, and it helped to lighten the picture. A slight blurring was combined with the modified median filter, and then it was changed to black and white to remove grey areas.

The code I wrote also flood-fills letters. This means that letters must be continuous and not touch other letters, limiting the function of this program. However, it still suits the samples at captchas.net.

4 Discussion, Conclusion, Recommendations

The purpose of this project was to develop a program that could break a CAPTCHA, thereby impersonating a human. Because this project is not yet complete, a discussion has not been finished. It is underway. Conclusions and recommendations have not been made yet.

5 Appendices

5.1 Code - neural network

Code contains neural net class, neural net layer class, and neuron class. However, this is not included because if all the code was actually here, the report would be 34 pages.

5.2 Code - image processing

Code contains image processing methods, as well as a sample implementation. Selected methods shown.

```java
public static int[][] median(int[][] img, int blur, int offset) {
//blur from 1-9, 9 is all blur, 1 is median
int img2[][] = new int[img.length][img[0].length];
int width=img[0].length;
int height=img[0].length;
//median filter
for(int i=1;i<width-1;i++)
{
  for(int j=1;j<height-1;j++)
  {
    int[] neighbors=new int[9];
    neighbors[0]=img[i-1][j-1];
    neighbors[1]=img[i-1][j];
    neighbors[2]=img[i-1][j+1];
    neighbors[3]=img[i][j-1];
    neighbors[4]=img[i][j];
    neighbors[5]=img[i][j+1];
    neighbors[6]=img[i+1][j-1];
    neighbors[7]=img[i+1][j];
    neighbors[8]=img[i+1][j+1];
    Arrays.sort(neighbors);
    if(blur<=1) {
      img2[i][j]=neighbors[4+offset];
    }
  else if(blur<=3) {
      img2[i][j]/=3;
  }
  else if(blur<=5) {
    }
    }
  }
}
```
Figure 5: Captcha, after a few rounds of median-filtering. The shape is less sharp now, but at least noise is decreased.

```java
    img2[i][j]/=5;
    } else if(blur<=7)
    {
        img2[i][j]/=7;
    } else
    {
        img2[i][j]/=9;
    }
}
return img2;
}
public static void to_BW(int[][] img, int threshold)
{
    int width=img.length;
    int height=img[0].length;
    for(int i=0;i<width;i++)
    {
        for(int j=0;j<height;j++)
        {
            if(img[i][j]>=threshold)img[i][j]=255;
            else
            img[i][j]=0;
        }
    }
}
public static int[][] to_Array(BufferedImage image)
{
    int width=image.getWidth();
    int height=image.getHeight();
    int img[][] = new int[width][height];
    for(int i=0;i<width;i++)
    {
        for(int j=0;j<height;j++)
        {
            Color c=new Color(image.getRGB(i,j));
            img[i][j]=c.getRed()+c.getGreen()+c.getBlue();
            img[i][j]/=3;
        }
    }
}
```

6 Literature Cited

References


7 Acknowledgements

I had a lot of help from Mr. Latimer, who provided knowledge and guidance.
Excursions into Parallel Programming with MPI  
Computer Systems Lab, 2007-2008  
Michael Chen

Abstract

With more and more computationally-intense problems appearing through the fields of math, science, and technology, a need for better processing power is needed in computers. The solution can be found through parallel processing, the act of linking together multiple computers and cutting run-time by using all of them efficiently. MPI (Message Passing Interface) is one of the most crucial and effective ways of programming in parallel, despite its difficulty to use at times. However, there has been no effective alternate to this date, and continues as the de facto standard of parallel programming. But technology has been advancing, and new MPI libraries have been created to keep up with the capabilities of supercomputers. Efficiency has become the new keyword in finding the number of computers to use, as well as the latency when passing messages. The purpose of this project is to explore some of these methods of optimization in specific cases like the game of life problem and heat dissipation. Keywords: Message passing interface, ecosystem simulations are just a few of the multitude of applications which will surely require parallel computing. Though my plans only include optimization of the game of life and heat dissipation problems, it is these basic skills that carry over into real-world applications, except on a much larger scale.

2 Background and review of current literature and research

Parallel programming is the concept that multiple processors can be used to split up a task and then combine the separate parts to enhance processing speed. With advancements in many fields requiring computation-intensive calculations, parallel programming has become increasingly popular. This has been explained in many books such as Introduction to Parallel Programming as well as Parallel Programming in MPI. In specific, the type of parallel programming being used in this project is message passing, having computers send and receive data from each other to complete programs. The first message passing interface was released in 1994 for Fortran. 14 years later, though it retains much of its old functions, MPI-2 has become a new standard, focusing more on parallel I/O, dynamic process management and remote memory operations. Project ideas that have been explored include taking images taken from aircraft and placing them on a map in terms of longitude and latitude, as well as symbolic computations for recognizing speech and facial features. The complexity of algorithms and functions ranges greatly and often depends on the efficiency of the code. For example, though master-slave message passing is probably the easiest to code, it runs much slower than the divide and conquer strategy. (Figure 1) Whereas the first only has messages passing between the main computer and slave computers, divide and conquer has each processor communicating with any other processor it needs information from. But these are still just the basics. As stated above, companies like IBM are actively searching for new methods to tackle their
problems, using collective and torus networks (that allow quick communication in the least number of steps), and different modes with which to use dual processors.

Also, a quick overview of the game of life, one of the projects I am working on. It begins with a board with a certain number of cells, with each one either alive (1), or dead (0). It is a very basic simulation of social interaction, since whether or not each cell lives or dies depends on the status of the cells around it in all eight directions. If 4 cells around it are alive, then it dies from overcrowding. If less than 2 cells are around it, it dies from overcrowding. It lives if 2 or 3 cells are alive, and if the cell is currently dead, it comes back to life (reproduction) when there are 3 cells as neighbors. Running this repeatedly leadings to a series of interaction between cells that can cause eventual patterns to form (squares and crosses survive).

### 3 Development

#### 3.1 Requirements and Limitations, Overview, Development plan

The project will be deemed successful under a few conditions. At the beginning of the year and up till now, it has been to tackle problems and programs involving MPI, like the wind velocity lab from first quarter, and Mandelbrot and game of life from second quarter. However, from third quarter on, the focus will shift from implementation to optimization. My goal by the end of the year is to write a program that can test the run results of two specific problems: the game of life and heat dissipation problem, which both use similar cell by cell processing.

The technology demand of MPI will be no problem to meet, since at TJHSST, all the computers in the Systems Research Lab are compatible with MPI, and have enough processing power to suffice for any computational power I will be using.

#### 3.2 Research Theory

Although I started with programs like embarrassingly parallel computations, I have moved through more difficult aspects of coding, namely the divide and conquer algorithms. My programs begin with writing a non-MPI program, and converting it to work in parallel. I now have a clear grasp of the basics of MPI (through much trial and error), and it will allow me, in the next quarter, to change the view of my project.

Latency has many definitions, but in terms of MPI, it represents the amount of time a processor has to wait to receive a message from another processor. In MPI, this number is essential in making a program as efficient as possible, and will become a major focus in the next quarter. Run-times for parallel processing is basically a combination of two factors: latency and processing power. Depending on which one the computer excels at, a program will be better off either sending more messages and calculating less per message, or sending large messages at once, and spending more time between messages calculating. Finding the perfect balance is the problem that is being posed.

#### 3.3 Developmental Procedures

During first and second quarter, I have been following along with the supercomputing class, which has been diving into parallel programming with MPI. However, nearing the end of the 2nd quarter, I decided to break off, and work more on the game of life program. The process began with writing the game of life without MPI, which was done quickly. However, making it run in parallel was the hard part, hindered by two problems: Java has been my main language for the past three years, so I run into syntax errors in C quite often, and because I encountered problems at first with sending and receiving in MPI (array sizes caused problems in the game of life).

The theory behind the game of life with MPI is that the board will be split by the number of processors used in calculation. But each section also needed a limited amount of information from surrounding areas, and this is where message passing came into play. My time focused on writing code that would allow each processor to send one row or one column to another computer, as well as receive it. This is also where the latency problem comes into play. If a computer sends and receives not one, but two rows and columns at once, then the first time it runs, the data will be accurate up to the first row. Then, after being run the second time, the data is accurate inside the assigned section. Thus, by doubling the amount passed (or actually more than doubling), the amount of time needed before information needs to be passed again is also doubled. (Figure 2) But a limit has to be drawn eventually. There is little use in, say, passing the entire board to each cell. Also, when the main computer receives a print command, it will wait for all the other computers to send information on its section, and combine them to display output on the screen (this is just parent-child).
3.4 Testing and analysis

Because of the nature of MPI, it is not restricted to certain capabilities, though it does excel at some. Thus, a variety of possible displays and processes can be run, and depending on the specifics of each programming, different debugging and error analyses are required. In specific, every program I write needs to have the number of processors specified. For the game of life, I am working on a feature that allows the user to input the exact width and length and number of cells to start alive (in the form of a percentage), before the program runs, and also use mouse clicks to let the simulation run, or to run it step by step. The heat program will have a similar interface.

But what is important is not what the programs look like when they run, since they will be the same if I pass one row or five rows. What is important is the records of their run-times, which I will try to write a general program to record, so I dont have to manually run everything. Fortunately, since the game of life and heat programs use similar boards and characteristics, it should be relatively simple to write one testing program for both of them that will record speeds based on amount of message passing vs. amount of internal processing. The final product will be creating a program that can find the optimal conditions for a certain problem (in this case, game of life, though it can be expanded to fit for other problems).

4 Results, Discussion, Conclusion, Recommendations

4.1 Expected Results

Once again, I am hoping to learn about parallel programming more through MPI, a field that will become more important in the future. Even now, there are many research teams comparing and contrasting different ways of using MPI, including IBMs supercomputing team. For example, one research I read was exploring the possibility of a graphical interface called MPI-Delphi for workstation networks that allows quick and easy access for programmers. Even at the professional level, I read about testing done on blocking v. non-blocking coordinate checkpointing, another method of MPI. So whether or not the research I do yields a substantial result this year, or if I find a direction to follow, the knowledge and skills I obtain will become indispensable in the future.

The methods and programs to find optimization pale in comparison to the works that are being studied by programmers outside of a classroom setting. But even these simple tests provide an important basis for problems that are sometimes not more complex, but just much larger in magnitude. Though I know I will not be able to try the coprocessor and virtual node modes (which require dual processor nodes on each computer, the one IBM uses has over 65,000 nodes, each dual), by learning from the basics and working through these ideas, I can grasp the concept behind research that is going on now and will continue in the future.

4.2 Results

From my results from the wind velocity lab, the efficiency in adding number of processors increased, but steadily increased less until the number hit 8, at which the processing time increased with each added computer. From that point on, adding any more processors only made the time initializing and passing messages wasted. However, I do not have information to use for the game of life, which is a much more difficult problem to analyze. That is what I am going to work on for the majority of third quarter, along with efficiency of heat dispersion.

The problem with measuring these will be that it will measure not on the number of processors used, but also the amount of information passed between steps (passed every step? Or every three steps?) This will affect the latency and the amount of processing that each computer has to do. Depending on which task the processors are more suitable for, one will yield better results.

4.3 Future Testing

There is much to be done in the following two quarters, and more research to be done. Mr. Torbert has suggested using latency algorithm and formulas that can be found online to gauge efficiency and run-times, and I need to look into that while working on the testing programs in the near future. And by doing so, I will to move my testing and programming more from the theoretical and experimental, into something more practical.

5 Appendices

Sample Code from Wind Velocity Lab:

```c
for(angle=0.0;angle<=maxangle;angle+=0.5) //parent process
for (wvel=-50.0;wvel<=wmax;wvel+=0.1)
{
//if all child processes are busy, wait to receive info
if (flag>=size)
```
MPI_Recv(args, 3, MPI_DOUBLE, k, tag, MPI_COMM_WORLD, &status);
printf("receiving! endx:%3.2f\n", args[2]); //recalculates acceleration and accounts for it
flag--;
}
//after receiving, send info to the same process
args[0]=wvel;
args[1]=angle;
args[2]=1337.0;
MPI_Send(args, 3, MPI_DOUBLE, k, tag, MPI_COMM_WORLD);
//move to the next process and go around the for loop
k++;
flag++; if (k>=size)
k=1;
//making sure not to exceed number of computers available
printf("sending! angle:%3.2f,wvel:%3.2f\n",angle,wvel);
}
//this section of code makes sure that all values have been received
int recall=0;
int ori=k-1;
if (ori<1)
ori=size;
while (recall==0)
{ if (size==2)
{MPI_Recv(args, 3, MPI_DOUBLE, k, tag, MPI_COMM_WORLD, &status);
printf(%3.2f\n",args[2]); break;
}
MPI_Recv(args, 3, MPI_DOUBLE, k, tag, MPI_COMM_WORLD, &status);
printf("%3.2f\n",args[2]);
k++;
if (k>=size)
k=1;
if (k==ori)
recall=1;
}
//this portion of code ends all child processes
args[0]=0.0;
args[1]=0.0;
args[2]=666.0;
int n;
for (n=1;n<size;n++)
MPI_Send(args, 3, MPI_DOUBLE, n, tag, MPI_COMM_WORLD);
while (py>=0) //as long as the projectile has height
{
MPI_Recv(args, 3, MPI_DOUBLE, k, tag, MPI_COMM_WORLD, &status);
printf("receiving! endx:%3.2f\n", args[2]); //recalculates acceleration and accounts for it
ax = (vx-vw)*(vx-vw)*rf;
//acceleration of wind depends on direction of wind
if (vw<vx)
ax=-ax;
//checks if wind velocity is working with/against gravity
else
ax=gy*vy*vy;
vx+=ax*dt;
vy+=(ay*dt);
px+=vx;
py+=vy;
}
//this portion of code ends all child processes
args[0]=0.0;
args[1]=0.0;
args[2]=666.0;
int n;
for (n=1;n<size;n++)
MPI_Send(args, 3, MPI_DOUBLE, n, tag, MPI_COMM_WORLD);
MPI_Recv(args, 3, MPI_DOUBLE, 0, tag, MPI_COMM_WORLD, &status);
double vx = v0*cos(args[1]*M_PI/180);
double vy = v0*sin(args[1]*M_PI/180);
double vw = -args[0];
//checks to see if parent process ends this process
if (args[2]==666.0)
break;
for(m=ya;m<ya+xymax;m++)
arr[xa+xymax][m]=rec[xa+xymax][m];
}
if (ys>0)
{
MPI_Recv(rec,max*max,MPI_INT,rank-sqrt(size),tag,MPI_COMM_WORLD,&status);
for (n=xa;n<xa+xymax;n++)
arr[n][ya-1]=rec[n][ya-1];
}
if (ys<sqrt(size)-1)
{
MPI_Recv(rec,max*max,MPI_INT,rank+sqrt(size),tag,MPI_COMM_WORLD,&status);
for (n=xa;n<xa+xymax;n++)
arr[n][ya+xymax]=rec[n][ya+xymax];
}
//1 if you want to print!
//computers all send print info to main computer
if (rank!=0 && pflag==1)
MPI_Send(arr,max*max,MPI_INT,0,tag,MPI_COMM_WORLD);
//main computer prints
if (rank==0 && pflag==1)
{
for (k=1;k<size;k++)
{
MPI_Recv(rec,max*max,MPI_INT,k,tag,MPI_COMM_WORLD,&status);
int xtemp=((int)k)%((int)(sqrt(size)))*xymax;
int ytemp=((int)k)/((int)(sqrt(size)))*xymax;
int xmax=xtemp+xymax;
int ymax=ytemp+xymax;
for(n=xtemp;n<xmax;n++)
for(m=ytemp;m<ymax;m++)
arr[n][m]=rec[n][m];
display();
}
//one step over
flag++;}

6 Literature Cited
P. Pacheco, Parallel Programming with MPI, M.
Almsi, G., Archer, C., Castaos, J. G., & Gunnels,
J. A. (2005, March). Design and implementation of
message-passing services for the Blue Gene/L super-
computer. IBM Journal of Research and De-
velopment, 49(2/3), 393-406. Retrieved from Pro-
Quest database.
Buntinas, D., Coti, C., Herault, T., Lemarinier,
Blocking vs. non-blocking coordinated checkpoint-
ing next term for previous term large-scale fault tol-
erant MPI Protocols. Future Generation Computer
Systems, 24(1), 73-84. Retrieved from ScienceDi-
rect database.
Gregoretti, F., Laccetti, G., Murli, A., Oliva,
enabled MPI library. Future Generation Computer
Systems, 24(2), 158-165. Retrieved from ScienceDi-
rect database.

7 Acknowledgements
Special thanks to Mr. Latimer for help throughout
the year.
Also thanks to Mr. Torbert for help with MPI
with programs like wind velocity and Mandelbrot,
as well as help with Game of Life and ideas for
future projects
Exploration of Genetic Algorithms Through the Iterative Prisoner’s Dilemma

Aaron Dufour
TJHSST
Alexandria, Virginia

Abstract
Genetic algorithms are used for many optimization problems to find a near-optimal solution when finding the optimal solution would be too time-consuming. Although unable to tell when it has found the optimal solution, a genetic algorithm works continues until the probability of having found the optimal solution is sufficiently high. There are many methods used to perform each step of a genetic algorithm, but it is not easy to identify which will work best for a specific problem. The goal of this project is to compare these different methods through the iterative prisoner’s dilemma, and to hopefully find which methods work best generically.

1 Introduction
The prisoner’s dilemma is a problem involving two players. The players each must decide whether to cooperate with the other or to defect. These decisions must be made without knowledge of the other player’s decision. Points are given to each player based on the moves they made. If both players cooperate, they are each given R points. If one cooperates and the other defects, the cooperating player receives S points and the defecting player receives T points. If they both defect, they are each given P points. In order for it to be a prisoner’s dilemma, the values must follow the inequality T > R > P > S.

In the iterative prisoner’s dilemma, the additional inequality 2R > T + S must be satisfied. In this scenario, the same thing happens, except that the players are against each other many times with memories of the past. In this scenario the best outcome is for both players to cooperate each time, because the total points given when both cooperate (2R) is greater than the number of points given if one defects (S + T) and greater than the number of points if both defect (2P). The problem associated with the iterative prisoner’s dilemma is to find the rule that should be followed in order to maximize the number of points received when it participates in this scenario with a variety of other players. This is a good problem on which to use a genetic algorithm because there is no algorithm faster than brute force that has been proven to find the optimal rule. In my genetic algorithm, I made each solution a collection of bits that represent whether the player should cooperate or defect given a past collection of turns. The fitness value for each possible solution is the number of points it accumulates after going through a set number of turns with each other possible solution in the population. The methods by which the each part of the genetic algorithm is done can be changed easily because each possible solution is a simple string of bits.

2 Background
The iterative prisoner’s dilemma has been studied extensively in the past. Because the best rule is agreed upon, it is a good case with which to test genetic algorithms. It has been shown that the best rule is to cooperate on the first turn, and then do the same thing that the opposing player did on the previous turn for the rest of the turns. The only exception is if the opposing player defected the previous turn, there should be a 3% chance for the rule to tell the player to cooperate instead of following the opposing player. This rule was found by Robert Axelrod through a series of tournaments in which he invited colleagues to devise rules for the iterative prisoner’s dilemma, and then had them all play against each other. Many people have written a genetic algorithm that solves the iterative prisoner’s dilemma, but I have not found a case in which this problem was used to study genetic algorithms.

3 Development Sections
My program can run a genetic algorithm to find a solution to the iterative prisoner’s dilemma using many different genetic algorithms methods. It can
take user input to tell it which method to use for each part of the algorithm, and can set essential constants such as the mutation rate, the number of generation, the size of the population, etc. As the program runs, it displays a graph showing the fitness of each of the current solutions. It also shows a graph that shows the average fitness of each previous generation. Finally, it outputs a file with these average fitness values so that they can again be graphed after the program has completed the run. I will use averages of the number of generations each method takes to get to the optimal solution in order to judge the usefulness of each method. Before I can do this, I will need to write a program that determines when the optimal solution was reached, because the mutations prevent this from being a simple procedure. My results will be graphs of how different genetic algorithm methods compare against each other. This will hopefully aid in the deciding of which genetic algorithm methods to use by future programmers.
TJHSST Senior Research Project: TJHSST Website Redesign
2007-2008

Martin Elthon

1 Abstract

The purpose of this project is a redesign of the TJHSST website backend. Through the use of PHP and MySQL databases, this project will result in a redesigned administrative interface for the TJHSST website. The current state of the TJHSST website is in a state of disrepair, and web pages have to be edited manually. To resolve this, and help with the general overhaul of the current site, this project will form the foundation of the future web site. The current state is in a deteriorating condition, with system failures becoming more and more common.

2 Background

By the very nature of this project, there is not a tremendous amount of research that can be done. The research that has been done has focused on Intranet 2, which is about as close to this project as it can get. Additionally, the code for Intranet 2 can be viewed and the coders of it can be interviewed. I have found, however, that little research is needed, since the

3 Introduction

The current state of the TJHSST website is decrepit. It was written a long time ago in a language that does not exist anymore. After a collapse of the system the previous year, a "hackish" job was done to bring the site back online. However, at a sharp cost—the whole of the core site is not dynamic. This means that the administration has to manually edit the page. Late last year, a team was formed to redo the current site, and this project is a large part of that effort. This project's goal is to provide a new management interface for the administration to manage news posts, and the various dynamic content that the TJHSST site provides. Written using PHP, XHTML, and CSS, using LightHTTPD and MySQL 5 for the web and database server, this new site is using the latest web-development technology to create a lasting site. Hopefully, through careful documentation and good coding practices, the nightmare of the current site can hopefully be avoided in the future.

4 Preliminary Testing

As of right now, there is only one user of this backend. He is the main user of the current TJHSST backend, and can therefore provide useful suggestions as to what features need to be implemented. Basic testing has been done with all the features, such as testing authentication when a user is not authenticated. Meanwhile during these tests, the database was monitored for changes, to see whether a function actually made a change. The almost all of the features have been implemented. Secure login and authentication, account management, staff email management, news management, and spell checking for news submission.

5 Analysis

The results so far have shown the software used to be reliable and speedy. Since the site is under a significant amount of work every day, it is hard to say the true reliability of the software itself. I have never had the system crash or become unusable.
Abstract

The biomechanical features of a runner in an image can be analyzed by using certain image processing techniques, the primary method being edge detection. By constructing an accurate, two-dimensional model of a runner’s lower body from a rear angle, it is possible to extrapolate the underlying qualities of that runner’s biomechanics. This is done by creating an outline of a runner’s lower leg and feet. An edge detection algorithm is applied on an image to create this outline. In this type of situation, algorithm speed is not a very relevant issue; accuracy is far more important, the reason being that you only need to analyze a few images to create a two dimensional model of the lower body, as well as the fact that the time it takes to analyze a runner does not directly affect his performance as a runner.

Keywords: Biomechanics, Running, Edge Detection, Image Processing

1 Introduction - Elaboration on the problem statement, purpose, and project scope

1.1 Scope of Study

The goal of this project is to analyze images of a runner and extract biomechanical information about the runner from the images. Among runners, a major cause of injury is over pronation. Pronation is the natural inward rolling of the ankle to absorb impact. All runners should pronate to a degree, but many runners pronate to much, causing misalignment, knee problems, and problems with the muscles and ligaments around the ankle. Using only images, the project will determine the degree of pronation of a runner, which could be instrumental in determining the proper shoe type and diagnosing injuries. The project will strictly be involved in analyzing images from a controlled environment and determining biomechanical features from analysis of images. This means that the project will not be concerned with selecting images from a video feed or trying to analyze images taken in random and widely varying situations. The images used in the project will be taken from the back of a runner running on a treadmill, not from a runner running in stormy weather in an urban environment, taken at an awkward camera angle. There is very little purpose in trying to determine the biomechanics of random people walking in the street, so focusing on controlled environments makes the project much more feasible at almost no cost to applicability in the real world.

1.2 Expected results

The expected goal of this project is create a system that can accurately determine the biomechanical features of a runner and come up with a verdict concerning the efficiency and proper shoe type of the runner. The system is not designed to totally replace human analysis of runner biomechanics, but rather to assist human analysis and eliminate at least partially mistakes made in the human biomechanics analysis process.

1.3 Type of research

This project is pure applied research. The fundamentals of human biomechanics are already known. A lack of knowledge is not a problem that this project is proposing to address, rather, the project is addressing the problem of automatically determining variables in different situations and assisting current methods of a process that is already being conducted by humans.
2 Background and review of current literature and research

The aim of this project is largely unique in the academic world. However, the commercial sector has tackled this problem, but only because of motivation to sell more running shoes and increase profits. Fundamentally, this project is venturing into unknown territory. It is a melding of known information about biomechanics and edge detection and image processing techniques. Consequently, the background of this project lies in two, distinctly different areas.

3 Procedures and Methodology

The first step in the process is to get an outline of the lower leg and feet from an image. Using gaussian blurring, noise removing techniques, and outlier removal algorithms, and edge detection program creates this outline. Once an outline is made, there are two possible methods for determining the degree of pronation. The first method is will find the degree of pronation by determining the general angle of the lower ankle and foot and comparing it to the angle of the leg. Disparities in the two angles will conclude either pronation or supernation, depending on the sign of the angle difference. The second method is probably the one that will be further developed in the later stages of the project. This method requires two images, an image of the runner right before and right after impact. An algorithm is applied to both images that extracts an outline and the average x value of the outline. These two values are compared between each image, and the higher the value, the higher the degree of pronation. However, the output of this method, the pixel difference between the two values, is relative to the distance between the runner and the camera. Thus, it is important to standardize this distance. To determine this relative factor, the program will be tested and analyzed on a multitude of neutral runners, runners with a correct amount of pronation. After storing the outputs of all the neutral runners, the outlier outputs will be discarded, and the remaining range of output values will be designated as the neutral output range. Output values above the neutral range will be over pronators and output values below this range will be superantors.

4 Expected Results

This project is expected to create an overall algorithm that can analyze images of a runner under a controlled environment and come up with a biomechanical fingerprint for the runner, namely, the degree of pronation of the runner. The implication of the project is a new precedent for accurately determining proper shoe type for runners and preventing common injuries.
Abstract

Particle Swarm Optimization is a method of optimization for n-dimensional infinite search spaces. This project aims to test different social influences, the way in which the particles communicate with each other in order to find a global minimum, on the particles and their ability to converge on a correct solution. The different versions of the social interactions are tested using various benchmark functions and then the different methods are compared to each other.

Keywords: Particle Swarm Optimization, Fully Informed Particle Swarm, Social Interaction.

1 Introduction - Purpose and Scope

Particle Swarm Optimization (PSO) is a technique used to optimize n-dimensional infinite search space problems. A large amount of particles exist in the search space and "fly" through it searching for the global minimum. Particles are influenced by both cognitive and social interactions changing their course of "flight". This project aims to alter the social interactions in order to increase the efficiency of the swarm.

If a way of social interaction is proven to be more efficient than the others, then it can replace the older method, and thus make the algorithm more powerful. This could also lead to other improvements increasing the quality of algorithm.

Anyone who has a problem in which an optimal result needs to be found quickly can use this algorithm. It has seen extensive use in neural networks and finds potential in time-critical optimization problems where the constraints quickly change.

This project will deal only with the social interactions between agents. It will not deal with inertia or cognitive influences and they will remain constant throughout the program. It should also be noted that thusfar in the project, the only form of comparing the different influences will be by iteration count not by actual time to run the program.

2 Background

PSO is a relatively new swarm intelligence technique. It was first created in 1995, inspired from flocks of birds and schools of fish. As the technique progressed, however, various modifications were made in order to improve both reliability and time cost. It was not until the work of Clerc that the technique was truly analyzed from an algebraic point of view. In his work, Clerc describes a simple one-dimensional, one-particle, Particle Swarm and then proceeded to rebuild the swarm back to its original form. PSO is used for n-dimensional optimization problems, because it is relatively easy to implement. A set of particles is randomly created in the search space. Each particle is given a random velocity to move about the search space. Its velocity can be adjusted during the run by both cognitive and social interactions. The cognitive interactions involve the particle remembering where it had the lowest fitness value, the particle’s velocity is adjusted to move towards this point. The social influences adjust the particle’s velocity based upon other particles (one or more).

Though different types of social interactions have been tested in the past, the conclusions have not been conclusive. This could be in part due to the so-called No Free-Lunch Theorem, which states that because there are so many numerous testing types that if all possible tests were performed over any algorithm, they would all be equivalent. However, since not all functions are being tested, but rather only a small subset wherein PSO has application, it is unclear as to whether the No Free-Lunch theorem will hold or not.
3 Social Interactions

The social interactions currently covered by this project include:

1. Non-Informed Particle Swarm (NIPS)
2. Singly Informed Particle Swarm (SIPS)
3. Fully Informed Particle Swarm (FIPS)

The section below will describe the different forms of social interaction and how they are implemented and act on the swarm.

3.1 Non-Informed Particle Swarm

NIPS works on a very basic principle that the particles do not in any way associate with each other. The only method by which the particles velocities are adjusted in any meaningful way is by cognitive means. The particles velocity is updated by the following method:

\[
\begin{align*}
\vec{v}_{t+1} &= \alpha \vec{v}_t + \varphi (\vec{P}_t - \vec{x}_t) \\
\vec{x}_{t+1} &= \vec{x}_t + \vec{v}_{t+1}
\end{align*}
\]  

(1)  

(2)

For each particle, the velocity, \( \vec{v}_{t+1} \), and position, \( \vec{x}_{t+1} \), vectors are updated. \( \vec{P}_t \) is representative of the particles best previous value, the pbest. \( \varphi \) is a random number between 0 and 1.0.

3.2 Singly-Informed Particle Swarm

SIPS is the basic version of PSO. Simply put, the swarm finds the particle with the lowest fitness value, and all the other particles are drawn towards it. The mathematical view is very similar, but only slightly differs in the adjustment of the velocity.

\[
\begin{align*}
\vec{v}_{t+1} &= \alpha \vec{v}_t + \varphi_1 (\vec{P}_t - \vec{x}_t) + \varphi_2 (\vec{P}_g - \vec{x}_t) \\
\vec{x}_{t+1} &= \vec{x}_t + \vec{v}_{t+1}
\end{align*}
\]  

(3)  

(4)

In this method of swarm, \( \varphi_1 \) and \( \varphi_2 \) are between 0 and 1.0. The particle has an equal amount of influence between the swarm’s best and the particle’s own best position.

3.3 Fully-Informed Particle Swarm

The last swarm being tested by this project thus far is the FIPS. This particle swarm is slightly different than the others. Instead of the canonical version of velocity testing (as seen in SIPS). The particles find the best point, collectively and then move towards that point.

\[
\vec{P}_m = \frac{\sum_{k \in N} W(k) \varphi \otimes \vec{P}_k}{\sum_{k \in N} W(k) \varphi}
\]  

(5)

\[
\vec{v}_{t+1} = \alpha \vec{v}_t + \varphi (\vec{P}_m - \vec{x}_t)
\]  

(6)

The best point found by the swarm, \( \vec{P}_m \), is found by adding all particle’s position vector multiplied by \( W(k) \), a weighting factor of some sort. What is actually contained in \( W(k) \) does not matter to a large degree as it is averaged out over the entire swarm. The weighting factor only determines to what extent the particle is influential in the swarm.

4 Procedures

4.1 Testing

For this project, it would not be very possible to use a mathematical formula to judge performance for the swarm, due to the fact that a great part of the algorithm (including starting position and velocity) are derived randomly. Therefore, the program will be tested by running the program multiple times for each social interaction and function tested, and from it determining the average running time and number of time steps needed for the swarm to converge on the correct answer, if the swarm indeed converges.

4.2 Software

For this project C was used for coding purposes. In addition, the OpenGL library was used in order to graphically depict 2 dimensions of the benchmark function tested.

5 Expected Results

Expected Results and Value to Others
I expect that the FIPS will fare better than some of the other types of interaction tested, but it will not be the most efficient in terms of time due to the $n^2$ addition required for that method. Overall, I believe that the Single Influenced Particle Swarm will do the best overall because of its simplicity and robustness. The NIPS will do the worst due to the tendency of its particles to reach and maintain at local extrema.

References


TJHSST Senior Research Project
Genetic Algorithms to Find Near Optimal Solutions
to the Traveling Salesman Problem
2006-2007

Karl Leswing

Abstract

My main areas of interest within Computer Science are machine learning, and optimization algorithms.

Keywords: Genetic Algorithms, Ant Colony Optimization, Traveling Salesman Problem

1 Introduction

1.1 What is research

My research has primarily been in optimization algorithms. These algorithms are used to find optimum or near optimum solutions to various problems. This project is specializing in genetic algorithms. Genetic algorithms use a biological approach to solving problems. They simulate a real life environment with solutions reproducing and mutating.

1.2 Why is research done?

There is constant research into using optimization algorithms to solve NP hard problems such as the Traveling Salesman Problem (TSP). Non deterministic Polynomial time hard problems come up often, and slightly more efficient solutions are very valuable. The TSP is often used in Computer Science and discreet mathematics because it is so easy to explain yet so difficult to solve. In this project I use genetic algorithms to find near optimal solutions to the TSP.

2 Background

2.1 Traveling Salesman Problem

The Traveling Salesman Problem (TSP) is a good problem because of its simplicity to explain yet its incredible difficulty to solve. The problem is outlined as thus: given a number of cities and the costs of traveling from any city to any other city, what is the cheapest roundtrip route that visits each city exactly once and then returns to the starting city? The solution space is $1/2(n-1)!$ for all $n$ greater than 2 where $n$ is the number of cities. Using brute force methodologies become difficult after the tour length is greater than 40 cities. Using dynamic programming, the run-time can be cut down to $O(n^2*2^n)$.

This runtime is still very large. Therefore it has become necessary to use search heuristics to find near optimum solutions in reasonable amounts of time.

2.2 Genetic Algorithms

Genetic algorithms have been proven to be effective at optimization and pattern finding. They have been used in the past to create more efficient boat designs and fiber optic cable. Genetic Algorithms are inspired by evolutionary biology and incorporate such concepts as inheritance, mutation, selection, crossover, and reproduction.

Pseudo-code for generic GA

1. Choose initial population
2. Evaluate fitness for each individual in the population
3. Repeat
   • Select the best individuals to reproduce
   • Breed new generations using crossover and mutation
   • Evaluate fitness of the offspring
   • Replace worst ranked part of population with offspring

2.3 Ant Colony Optimization

Ant Colony Optimization is a probabilistic technique for solving computational problems. It reduces problems to graphs which it then traverses using simulated ants. It was inspired by watching the manor in which ants search for food.
In the real world ants initially wander for food. When they find food they begin to lay down a pheromone trail. Other ants then find the trail and are able to find the food, instead of wandering randomly they eventually reinforce the trail. The longer it takes for an ant to come down a trail the more time the pheromone has to evaporate. Since a short path is faster it will get marched on more and create a positive feedback loop until it is the only path which still has pheromone. Ant Colony Optimization has been shown to be good when dealing with dynamically changing graphs. It has been shown to have practical applications in network routing and urban transportation systems.

3 Procedure and Methodology

For ease of coding I have broken my coding into two major segments, creating a prototype and expanding on my prototype.

3.1 Current Project

Currently I have finished my Genetic Algorithm which is able to find a near optimal solution to a 50 city TSP in around one minute. I am comparing how my Genetic Algorithm works with a greedy solution.

My Genetic Algorithm currently has the following components:

- Cycle Representation
- Double Point Crossover Reproduction
- Single Point Mutation
- Two Dimensional
- Roulette and Elitism Selection

My Ant Colony Optimization Algorithm is currently under construction and will hopefully be near completing in two weeks.

3.2 Extensions

My project is by no means complete simply because I have an effective, working prototype. I have a number of extensions which I will create and implement throughout the rest of the year to make my genetic algorithm more efficient so it can solve larger TSPs in less time. These include but are not limited to:

- Matrix Representation of Path
- Better Fitness Algorithms
- Intersection Detection
- Double point mutations
- Multi-Threading to Reduce Finding Local Optimum

4 Testing and Analysis

Currently I am implementing a basic automated testing suit to ensure that all of the functions of my many classes are working. I am considering switching to jUnit automated tests in the future for a more comprehensive testing suite.

To ensure the accuracy of my solutions to the TSPs currently I am checking them by eye. One of the beauties of the TSP is that the solution is obvious when one is looking at it. However, I will hopefully get pre-solved TSPs or brute force smaller TSPs to test my optimization genetic algorithms. Also since the main focus of my project is comparing global search heuristics the algorithms will check each other to see who finds the best solution in the smallest amount of time.

5 Expected Results

I hope that by implementing multiple global search heuristics I will be able to see which ones are optimized for the Traveling Salesman Problem. My Genetic Algorithm already finds better solutions in less time than the greedy algorithm and I hope that my Ant Colony Optimization will get comparable results. I will also try to prove that the Ant Colony Optimization is better for dynamically changing graphs. Through my project will find the differences between multiple general search heuristics.
Abstract

After seeing many traffic simulation web applets that work but have limited interaction on the part of the user, I wanted to create a traffic simulation that had an extensive user interface through which the user had high freedom and customizability options. A project in this field is called VISSIM, created by Tom Fotherby. This is an extensive project written in 11,000 lines of Java code to produce traffic simulation using a microscopic approach. The finished application should be able to at least somewhat realistically simulate real world behavior of traffic. Therefore, the primary goal of this project is to provide an accurate simulation of real world traffic behavior for multiple settings based on user-defined data.

Keywords: traffic modelling algorithms, microscopic traffic simulation approach.

1 Introduction

1.1 Scope of Study

This project will be more advanced than standard web applets, but it will not be a substitute for professional traffic simulations. The idea for the project is for it to be platform-independent, as long as the computer running the application has the JRE (Java Runtime Environment). This application will provide an extensive user interface that allows the user to change program variables such as number of cars and more. Therefore, this program will be useful for getting a general sense of what would happen in a hypothetical situation by setting the variables to reflect those in the real world.

Iterative development plan includes first developing a simple prototype or layout for the program. This involves simply creating the 800x600 application window, creating a JMenu bar at the top of the screen, and dividing the lower half into a JPanel and GUI functionality. The implementation will be done in steps to allow for testing simultaneous to development.

1.2 Expected results

By the end of the year, this application is expected to realistically simulate real-world traffic behavior for road networks defined by the user. The program should provide a default road network displayed at the left of the screen, but with GUI objects such as textboxes and checkboxes for which the user can modify the default road network to be similar to something found in the real world. While I do not expect this project to be as extensive as a professionally developed traffic simulation, it should be advanced enough to be of general use for modelling basic traffic situations in basic road networks.

I am desiring to learn further about the general subject of microscopic traffic simulation, using traffic algorithms that can emulate real world behavior. In addition, it will increase my proficiency in Java and using the swing API to generate effective GUI’s. This project should let us see what would happen in basic road network situations including some more complicated situations such as merging. This project will also be able to examine factors such as how car speed and velocity affects traffic.

1.3 Type of research

Use-inspired basic research would best describe my project. My project seeks to research the effectiveness of microscopic traffic simulation through use and testing in basic road networks. I pursue fundamental understanding of microscopic traffic simulation and the accompanying traffic algorithms, but the main motivation is to see the application of the concepts in an actual simulation program, which is the aim of my program.

2 Review of current literature and research

- VISSIM: Visual Traffic Simulation VISSIM is a project developed by Tom Fotherby that visually simulates traffic using the microscopic traffic simulation approach. His project is very extensive, cov-
ering 11,000 lines of Java code and even including a "design" mode for the user to define their own road networks. While the project is now discontinued, the project is open-source and downloadable via the web. This project effectively emulates real-world situations on simple road networks without accounting for person by person characteristics such as slow and fast drivers, drivers who keep a lot of distance or drivers who tailgate.

### 3 Procedures and Methodology

Resources required: Netbeans Java IDE, JGrasp, Java SE, Java JRE. Programming language: Java

The main procedure will include developing from a simple prototype, a 800x600 application window with the basic JMenu, JPanel and GUI options at the right. Functionality will be implemented step-by-step for the application. This allows for easy simultaneous testing of the program.

### 4 Background on microscopic traffic simulation

In contrast to macroscopic traffic simulation models, microscopic traffic simulation models simulate single vehicle-driver units, thus the dynamic variables of the models represent microscopic properties like the position and velocity of a single vehicle. In addition, the "cars" are generally represented by simple geometric shapes rather than actual 3D car objects or 2D car sprites. Allows for a large number of car objects to be on-screen at the same time and to be dynamically updated.

In addition, there is a third type of simulation model according to the level of detail on which the traffic system is based. This third type is called mesoscopic modelling and has a mixed fidelity as compared to high fidelity for microscopic modelling and low fidelity for macroscopic modelling. Microscopic represents the smallest amount of detail per individual car, but it is able to simulate much more cars and more efficiently than the other two types of simulation models.

### 5 Preliminary Testing

Input data will be mainly user-defined data instead of the default data, for example 20 cars, cars moving at 20mph and so on. In addition, the program will output data such as a measure of how "bad" the traffic is, depending on the total situation. For crash analysis, the program should eventually output dynamically individual and independent probabilities for each car currently being simulated within the traffic simulation.

Error analyses are both visually examining the program and comparing the data with those of real-world data or other established traffic simulation programs such as VISSIM. In addition, error analyses can be checked by checking the data output of my program. The data output will include collision coordinates + probability analyses.

To verify the program, I will use specific structural and functional testing to verify each part of the application is working properly, from the JMenu, JPanel and GUI options at the right. Eventually, dynamic testing may be used to test random inputs, but for now path and branch testing will be very effective in determining whether the application can successfully emulate real world behavior in a variety of situations.

- Basic requirements: 800x600, platform-independent, executable jar file
  - Advanced requirements: Accurate simulation of real-world traffic behavior, runs at a smooth frame rate (not more than 100 and not less than 30), able to run user-defined data on road networks, provides a usable crash analysis probability mechanic.

Traffic simulation algorithms with the microscopic traffic simulation approach: including algorithms to handle 1) Car speed, velocity, acceleration 2) Car breaking distance 3) Traffic light length, looping 4) Turning, switching lanes 5) Car reaction time

### 6 Analysis

By the end of the year, this application is expected to realistically simulate real-world traffic behavior for road networks defined by the user. The program should provide a default road network displayed at the left of the screen, but with GUI objects such as textboxes and checkboxes for which the user can modify the default road network to be similar to something found in the real world. While I do not expect this project to be as extensive as a professionally developed traffic simulation, it should be advanced enough to be of general use for modelling basic traffic situations in basic road networks.

Using the microscopic traffic simulation approach, my application should allow for upwards of 50 cars on screen at the same time, all being dynamically updated at the same time. The properties and variables defined for each individual
car will also update dynamically with user input change.

Screenshot of 2nd quarter prototype

References

http://www.tomfotherby.com/Contents/Education/Project/index.html, 2002

Development of a Fractal Dimension Calculator

Kelly Ran

Abstract

Fractal dimension is used as an index of complexity in many research applications. Currently, researchers calculate fractal dimension from raster graphics. This project introduces an alternative method by calculating fractal dimension from vector graphics. In addition, a display screen will show the user all of the steps in the calculation. Keywords: fractal dimension, box dimension, vector graphics, vector images

1 Introduction

The term "fractal" was first coined by Benoit Mandelbrot. Used to describe geometric figures that exhibit self-similarity, fractals are now used in myriad applications. Every fractal has a numeric fractal dimension that can be used to indicate how complex the fractal is. Multiple methods for calculating fractal dimension have been formulated. The box dimension method is most commonly used in research applications.

Raster graphics are images that assign numeric values to every pixel. The values determine the color of each pixel. Vector graphics contain object types like paths and shapes to show images.

In research applications, box dimension is calculated from raster image. This paper describes how to calculate box dimension from vector images. It also describes how to display the step-by-step calculations so that users can understand every step of the process.

2 Fractal Dimension

To calculate fractal dimension using the box dimension method, a square grid of size $s$ is superimposed over the image. $N(s)$, the number of grids that contain part of the image, is counted. Box dimension, $D$, can be calculated using the following formula:

$$D = \frac{\log N(s)}{\log \frac{1}{s}} \quad (1)$$

3 Vector Graphics

This project uses Scalable Vector Graphics (SVG), a standard vector graphics language and format. Based on XML, it stores paths and shapes instead of storing a value for every pixel of an image.

4 Processing

This project also uses the Processing Development Environment. Processing, a language based on Java, comes with libraries and methods that are useful for creating graphical displays. The Candy library allows users to import SVG files, embed them in graphical displays, and search through them to find information.

5 Previous Research

Fractal dimension is used in many research applications. Previous projects have focused on the correlation between fractals dimension and aspects of the natural world. For example, Corbitt and Garbary found a correlation between fractal dimension and brown algae development. They took photographs of brown algae in different stages of development, calculated the box dimension of the photographs. (Photographs are raster images.) They knew that as the algae developed and grew older, the algae became more complex in shape. Corbitt and Garbary concluded that as algae became more complex, the fractal dimension of the algae increased.

6 Fractal Dimension Calculator

6.1 Requirements

This fractal dimension calculator must use SVG file inputs. In order to be successful, it must calculate the box dimension of the SVG input. It must also show a display screen and let users see the calculations.
6.2 Design

Using the SVG language, images are created. These images include fractals and non-fractal shapes. In the Processing environment, the SVG file is uploaded. Using the get() method, the coordinates and attributes of the SVG file’s objects are imported into an array. The SVG image is displayed on the screen.

Now, the grid is superimposed. Starting with a large grid size, grid boxes are shown on the screen. Using a for-loop, the program goes through the object array and tallies how many grid boxes cover objects. Then, smaller grid sizes are used. A hash table keeps track of data: grid size, s, is stored in the keys, and number of grids, N(s), is stored in the values.

When the user indicates that he or she would like to stop seeing calculations, the value for D, box dimension, will be calculated using formula (1) and displayed.

6.3 Testing and Analysis

Testing the fractal dimension calculator will involve using different inputs. Many fractals have fractal dimensions that can be calculated by other methods, so those fractals will be programmed in SVG and then inputted into the Processing program. The box dimension output will be compared to the actual fractal dimension, and the error will be calculated.

Also, simple SVG files for lines and squares will be inputted to verify that the box dimension outputs are D=1 and D=2, respectively, because lines are 1-dimensional and squares are 2-dimensional.

7 Results and Discussion

Using the path and transformation attributes of the SVG language, creating fractals in SVG format has been achieved.

8 Recommendations

A useful extension of this project would be to create a function that uploads multiple SVG files at once and writes the corresponding box dimensions to an external file. This would be useful for researchers who need to find fractal dimension en masse. Another extension would be to create a function that calculates box dimension from raster images. The Processing program would time how long the raster image calculations would take, and then compare that with how long the SVG calculations would take.

9 Appendices

9.1 Appendix A: Code

st1.svg to make a Sierpinski Triangle

sc1.svg to make a Sierpinski Carpet

htree.svg to make an H-Tree

9.2 Appendix B: SVG Fractals Generated by Code

Figure 1: Sierpinski Triangle, 3 iterations.

Figure 2: Sierpinski Triangle, 6 iterations.
Figure 3: Sierpinski Carpet, 3 iterations.

Figure 4: H-Tree, 5 iterations.

10 Literature Cited

References


[5] Sutherland, S. (Tran.). (2002). Fractal Dimension (S. Sutherland, Tran.).

11 Acknowledgements

Thanks to Mr. Latimer for being my mentor and to my parents for being my parents.
Abstract

Physics simulations are often of single concepts or immune to user control. My project aims to change that by allowing users to create a situation and then simulating the behavior of objects in that situation. Users will create objects either through shape tools, then the program will convert them to polymorphic objects and run the simulation. Objects varying from the simple to complex will be modeled: single shapes or multiple shapes connected statically or with axles.

Keywords: physics, simulation, interactive, ASSIST

1 Introduction

The majority of my research was in physics simulation: how to do it accurately, what equations to use, and how to implement them. Using the equations and properties that I give objects, the program determines and shows the way that the objects behave. I started with basic equations and added more complex ones as the year progressed.

This project models projectile motion and interaction between simple and complex objects. I define simple objects as rectangles or circles, and complex objects as multiple simple objects connected by pins or axles. Interactions include collisions, friction, and rolling. Objects can also be anchored to the background to provide platforms or obstacles.

My goal was to create a program, usable by anyone, that would help the user to gain a better understanding of physical interactions by inputting any situation using an intuitive input system and viewing the behavior of the system. The process of creating the program would also help me to gain a better understanding of physics.

2 Background

A team from MIT created ASSIST: A Shrewd Sketch Interpretation and Simulation Tool which inspired this project. The program was created in order to give engineers a way to model systems in the early stages of design, when only an idea exists, before a traditional CAD program, which requires precision and planning, would be appropriate. The user draws a mechanical system on a smartboard, including an arrow for gravity. The “sketchpad” system then interprets the drawing. Certain symbols have special meanings: an x is an anchor, a small circle is a pivot. Finally, the interpreted drawing is fed into a commercial simulator. My project was inspired by ASSIST and aimed to be similar but with more focus on the physics rather than the sketching.

3 Testing and Analysis

The two main sections of my program are the simulation and the objects. The simulation is implemented both in the main program file - that is, the file containing the main timer - and the objects themselves. The main file contains an ArrayList of SimObjects and at each timer iteration it calls the step and draw functions of every object. Each object is an instance of a subclass of the abstract class SimObject. SimObject defines step, which updates the object’s position and velocity when it is passed a double value dt. It also includes a signature for the abstract method draw, which is implemented differently in each subclass. The subclasses are currently Rectangle and Circle.

Circles are easy to draw, but rectangles are more complicated because their rotation changes the way they must be displayed. When the rectangle is created, I determine the angle from the center to each corner. When drawing the rectangle, I add its rotation to the angles already found and multiply the sine and cosine of those by the distance from center to corner to determine where to draw the points.
Other polygons should be similar in implementation to the rectangle, as I treat it more like a set of points than as a rectangle. The method fillPolygon is used to display it.

Complex shapes will be implemented using pins and axles. Pins will connect two shapes so that they stay together in the same position. To achieve this I will create a ComplexObject class that will contain a list of shapes that combine to form it. It will calculate collisions for every object in it and apply forces to each object so that they move in unison. Axles will be more complicated; I will have to give each object independent motion while still keeping them attached to each other.

Collision detection has been the most complicated part of the project so far. It is easy to find when something is past a wall - check every corner to see if the x and y values are within an acceptable range. Determining whether an object is in another is more difficult. For circles, one must check if the x and y values are within an acceptable range. There is a collision when the following conditions are satisfied:

\[ B < P_y < T \]  \quad (12)

Wall collisions and object collisions are both resolved using similar equations. When an object collides with a wall[1]:

\[ \nu_{a2} = \nu_{a1} + \frac{j}{m_a} \]  \quad (13)
\[ \omega_{a2} = \omega_{a1} + \frac{(\frac{\tau_{ap} \times \bar{n}}{I_a})}{m_a} \]  \quad (14)

\[ j = \frac{-(1 + e)\tau_{ap1} \cdot \bar{n}}{1/m_a + (\frac{\tau_{ap} \times \bar{n}}{I_a})^2/I_b} \quad \quad (15) \]

When two objects collide, there are two more equations, and the final one changes [1]:

\[ \nu_{b2} = \nu_{b1} - \frac{j}{m_b} \]  \quad (16)
\[ \omega_{b2} = \omega_{b1} - \frac{(\frac{\tau_{bp} \times \bar{n}}{I_b})}{m_b} \]  \quad (17)

\[ j = \frac{-(1 + e)\tau_{ap1} \cdot \bar{n}}{1/m_a + 1/m_b + (\frac{\tau_{ap} \times \bar{n}}{I_a})^2/I_a + (\frac{\tau_{bp} \times \bar{n}}{I_b})^2/I_b} \]  \quad (18)

There are currently three ways to create objects - two for circles and one for rectangles. I tried to come up with as intuitive a way as possible so as to make working with my program easy and fluid. One way of inputting circles is to click where the center will be and drag to create a radius. The other way is to click an edge and drag the diameter. I implemented both because I think that the second is easier, but I have seen the first used before. It was harder to figure out a way to create rectangles because there are more variables than with circles.

To create a rectangle, one clicks where a corner will be, then clicks again for another corner and drags to finish the rectangle. After creating a shape, a dialog box appears to ask for the velocity and color of the object. I would like to create something that does not interrupt the flow as much, but I am not sure how to do so.

Input method selection is part of the GUI. The GUI is manifested in a menubar with the ubiquitous File, Edit, and Help menus, and two rows of buttons along the bottom. Although File, Edit, and Help are not the most descriptive names for menus, considering what my project does, I chose them because psychologically it would probably be more difficult or distressing for a user to have unfamiliar menus. They contain exit, reset, help, and about commands. The rows of buttons along the bottom are speed controls and input controls. The speed controls are fast rewind, rewind, pause, play, and fast forward. They work except when collisions are involved, but I can fix this by reversing parts
of the equations. The input method controls are the three I already mentioned, and will eventually include anchors, pins, axles, and other polygons.

4 Preliminary Results

My program accurately represents projectile motion and collisions with walls without regard to friction, and with an elasticity of one. While running, it may seem that it is not accurate, but that is because people are judging it with respect to their experiences, which take place in the real world, which has many more forces than my program currently simulates. Once I implement friction and find a good way to determine the elasticity value for each collision, my simulations will seem much more realistic.

References

Abstract

My main areas of interest with this project is parallel programming and its contribution to solving problems of graph theory. This project will eventually explore the Ear Decomposition Search and try to implement it on a Travelling Salesman Problem in an effort to find a connection.

Keywords: Parallel Programming, Travelling Salesman, Ear Decomposition, st-Numbering

1 Introduction - Problem Statement and Purpose

1.1 What is Sought

The purpose of this experiment is to find an st-Numbering system that will allow for points of a Travelling Salesman Problem to be added one at a time so as to minimize the distortion of the previous path. The Ear Decomposition Search (EDS) will be the method used in finding the st-Numbering of the graph. In order to use the EDS, knowledge of parallel programming is needed.

1.2 Why is it Sought

The two aspects of this project occur in very different fields of Computer Science, parallel computation and theory. Yet, they are connected by a common goal, an increase in run-time efficiency. The expected results of this project will provide three advancements in knowledge:

1. Research in the field of parallel programming
2. Research in the study of NP-Complete problems
3. Research in the applications of st-Numbering

2 Background

Parallel Programming and its Applications Parallel programming is in no way a new concept. Unfortunately, for the past fifty years more emphasis has been put in improving run-time for serial programs. Now that hardware has almost hit its serial limit, it it turning to parallel implementations; dual cores, for example, are becoming more popular. More research will hopefully be put into parallel programming in the near future.

When serial algorithms do not parallelize well, new approaches are needed to tackle problems. In the case of the Depth-First Search, it does not convert to parallel well. The Ear Decomposition Search was created as the parallel equivalent of the DFS.

One of the most common applications to the EDS is st-Numbering. st-Numbering is a method of organizing vertexes of a graph so that mathematical induction stands. For example, when using an st-Numbering for planarity, if the set of vertexes including the first vertex of each ear is planar, then, by induction, the graph is planar. subsection Travelling Salesman Problem

The Travelling Salesman Problem (TSP) is a well known NP-Complete problem. In its simplest form, a salesman needs to visit a certain number of cities and then return home. The TSP asks for the shortest path or cheapest cost the salesman should take. With a BigOh value of n!, the solution to problems with a large number of cities is almost impossible. The NP-Complete problem asks for an algorithm that runs in polynomial time that solves either the TSP or the many other NP-Complete problems.

3 Procedures

For the continuation of the calendar year, my main focus will be in completing the Informal Parallel Programming Course for High School Students. I will also continue my research of EDS and TSP. My first step will be to implement an Ear Decomposition on any bi-connected graph. Then, I will implement the EDS on a TSP data set. Finally, I
will need an st-Numbering system that acts as the most efficient in minimizing the previous path with each subsequent path.

3.1 Software

Computer language, libraries, and hardware I will be using:

1. C
2. XMT-C
3. OpenGL
4. 64-processor PRAM chip provided by the University of Maryland

4 Expected Results

By December 11, I expect to complete the Informal Parallel Programming Course. By the end of January, I expect to complete a working Ear Decomposition Search on a TSP data set. My program should find an st-Numbering that minimizes the changes from sub-Tours with the final tour (the solution tour). By minimizing the number of changes, the algorithm can safely eliminate non-optimal tours without checking them, in effect improving run-time. I consider a working algorithm, even if it may only be a heuristic, a successful project.
Projects

5th Period
Abstract

Artificial Intelligence has for long been an important aspect of computer science, but unfortunately artificial intelligence is usually computed from a single agent perspective or with multiple, but highly omniscient agents. I plan on creating an artificial intelligence engine, which works by having multiple agents, each with highly limited perspective. In order to solve tasks, they need to communicate their portions with each other through a network. Using that scheme, it will much more accurately simulate crowd dynamics, using particle swarm optimization to optimize the calculations.

Keywords: AI, PSO, C++, OOP, SDL

1 Introduction

The AI engine I’m programming is implemented through C++’s object orientation. I have programmed several classes which interact in order to make a completed end project. As my engine is an agent based networking engine, naturally the first two classes are the agent, Person, and network, PNetwork classes. The main program must include an array of Persons, which is passed to the PNetwork class on instantiation. Then the main program only needs to talk to the PNetwork, as its managing the list of people from instantiation on, and will take care of the movement of the Persons.

The Person and PNetwork also utilize another class I’ve written, the weightlist class. This class is a set of two array based, fixed size, looping lists; one for data, the other for the data’s relative weighting. The important feature of the weightlist that other prewritten container classes don’t offer is a summation function. This function effectively averages the data list, based on the relative weightings, and a decay weighting that favors the more recent entries in the list. This is crucial because the communication aspect of the PNetwork has to have a way of keeping track of each Person’s communications. Therefore each Person in the PNetwork is assigned a weightlist.

2 Background

Artificial Intelligence programming always requires a task at hand for functionality and relative significance. In my implementation, the initial task for the agents in my network based artificial intelligence engine will be target detection and convergence. The agents will have to locate and converge upon a random target (eventually to become a user controlled target) using methods streamlined with particle swarm optimization methods. Additionally, as they attempt to accomplish their motives, they should become increasingly more efficient at it. This would have to be as a result of the optimizations found in the paper by Kennedy and Eberhart.

3 Procedures and Methods

3.1 Overview

This program relies heavily on object oriented programming and function pointers, two fairly involved programming tasks. Early on I ran into a roadblock attempting to create two simultaneously co-referencing classes. I solved that issue through template classes. I will expect to have to solve many similar issues in the future in the same manner. To program this engine, along with the accompanying game (for graphical output reasons) I’ll need C++ (and therefore the g++ compiler) along with the SDL (software digital layer) libraries, for keyboard input and graphical output, and I’ll be using OOP programming (therefore the gcc compiler won’t be sufficient) and PSO for optimization.

The way the engine works is by: instantiate a PNetwork with a list of Persons, and run the PNet-
work in a non-terminating loop (except by exit). The PNetwork class has a step() method in which the communicator methods from each Person in its list are called. The communicator methods simply return whatever the Person can see that he wishes to inform the others about. Then those messages are added into weightlists associated with the other people in the list. When everyone has communicated what they have to communicate, the PNetwork calls the summation function on the weightlists and instructs the Persons the summation is relevant to, to head in that direction. That simple process can repeat endlessly, with obvious variations in the behavior created through different communicator methods (each Person only has a pointer to a function of the proper parameters and return type, it can be defined on instantiation to be whatever the programmer wishes) and different communication distances and message resilience (currently they decay in accuracy as a function of the distance they travel).

3.2 Testing

I’ve programmed this project so far with several debugging features and text based outputs for constant error checking. While for the final project these would be commented out for the final compilation, I plan to continue programming with those features to allow for ease of code writing and testing. Right now I’m using a series of testing shells to assess the resilience of my AI system. I have shells which print out pixels for each agent in the simulation, using SDL, which work for the regular PNetwork and ngon_world classes. These automated tests inform me whether the program is doing what it should be.

Eventually I will apply this AI engine to a game I’ve already programmed in Java. Using this game with enemy AI, there will be a very clear visual display as to whether or not the engine is effective and fast. If the game runs smoothly and is challenging, I’d consider it a success. That will be the final testing.

4 Expected Results

This project will greatly expand the field of Artificial Intelligence, with this much more modular and isolated AI engine. There are no hints given to the agents in this simulation, they must discover and solve their task themselves, with only their communications to guide them, which also get distorted. Essentially, by increasing the amount of communication, and decreasing the amount of intelligence of each agent, I will make a much more realistic search optimization, crowd dynamics AI engine.

This project requires plenty of additional research however. I could program the engine simply as written above, but my real intent is to create such an engine that can be used in real time environments, such as but not limited to, game environments. By implementing particle swarm optimization, which in and of itself will require much additional research, I should be able to accomplish the desired feat of real time run speed.

5 References

6 Appendixes
Abstract

Technology becomes more advanced and more accessible with every passing day. Education should be utilizing this technology boom in teaching current students. However, this does not seem to be the case. The goal of this project is to try and implement computer programming, through Scratch, as a tool for educating students in math and science topics. Computer science education at a younger age becomes more and more essential as computers become more advanced and more accessible with each passing day.

1 Introduction

The main question that this research project aims to answer is, "How young is too young to start teaching children how to program?" The goal is to establish a computer science program at Cardinal Forest Elementary School through the use of the MIT developed program 'Scratch.' Hopefully by watching how the students use the program I will be able to come up with an answer to the question. Despite the fact that I will be graduating at the end of this school year, I hope to continue my partnership with Mr. Allard and Cardinal Forest throughout my college years in order to ensure that the program continues, and to see the results of my initiative.

This paper will detail both how Mr. Allard and myself will teach the students, and how the students go about solving the problems that they are presented with. Mr. Allard and I will develop a curriculum based on the Virginia Standards of Learning (SOL) and Program of Studies (POS) standards for the students. The children that participate in the program are in first through sixth grade, with each grade having a similar curriculum scaled to the necessary level. Not all of the elementary school students will be participating in this project however. Mr. Allard has selected a diverse group of students for this initiative that he think would do best in and benefit from a program such as this.

2 Background

The task of educating the younger generations about programming has been attempted before. The first attempt to create a kid-friendly programming language was Logo, made by Wally Feurzeig and Seymour Papert. This programming language mainly involved telling a turtle how to move around in order to make various pictures with the turtle's "pen." Since then, multiple programming environments and languages have come about to try and engage not only youth but more specifically girls in computer science and programming such as: Squeak, Alice, and Scratch.

Despite the bountiful number of tools that modern technology gives us for teaching students, little progress has been made in teaching computer science at the elementary school level. The necessary technology is present in the schools, but it is only being used to reinforce outdated teaching methods. Currently, computers are mainly being used as a medium to transfer information, much like a television. Computers have so much more potential than that. They should be used as a universal construction material, not as a TV screen. Programs like Scratch enable kids to create whatever they want to all by themselves. Children learn better by immersing themselves in whatever they’re doing, rather than just listening to a teacher telling them what to do (Papert, 1993).

The goal for this project is to establish something akin to a Computer Clubhouse at Cardinal Forest Elementary School. The original Computer Clubhouse was started by the Massachusetts Institute of Technology in Boston in 1993 to "provide more young people with the opportunity to become digitally fluent." (Resnick, 2002) At these clubhouses, kids and older youth "become designers and creators with new digital technologies. Club-
house members use leading-edge software to create their own artwork, animations, simulations, multimedia presentation, musical compositions, websites, and robotic constructions.” (Resnick, 2002) I want to start a computer science program at Cardinal Forest where students can think for themselves and create whatever they can imagine.

3 Development Sections

3.1 Timeline

This past fall, I contacted the principals of the middle and elementary schools in the West Springfield pyramid inquiring about the possibility of starting a computer science program at their schools. Only one school replied (Cardinal Forest) and the principal referred me to Mr. Allard. October and November were spent sorting out which programming language to use and how we were going to use it. After experimenting with 'Squeak' and 'Alice' we decided on 'Scratch.' It was decided that the Scratch Lunch Bunch as it has come to be known would meet during Lunch on every Thursday. I personally teach the students every other Thursday.

3.2 Lessons

Each class lasts from 30-45 minutes, depending on the age of the students attending. Thus far, Mr. Allard or I instruct the students on what to do for a majority of the time, but this is not our end goal. As the year moves on, we will encourage students to explore the program independently and discover new methods and techniques on their own. We hope that this will not only teach the students how to take some initiative, but we think that they will gain a better, more solid, understanding of the program through individual exploration. Each lesson begins with the students signing in to the lab and quietly sitting down at their computers. Once everyone is seated, we begin teaching. By the end of the year, we hope to simply give students a project idea or category (i.e. celebrations or sports) and have them make a project all on their own.

3.3 Topics

The actual number of topics that can be covered will be dictated by how quickly the students move from one topic to the next. Thus far, the students have been exposed to the coordinate axes, degrees, if-then statements, and basic loops. Multiplication was also brought up in the loops discussion since the two subjects go hand in hand. We did this by first having the students tell their sprite to move in a square by telling it to move, then turn 90 degrees, move, turn 90 degrees, move, turn, move, turn. We told the students that this type of coding is inefficient and that there is a way to tell the sprite to do the same thing over and over again: loops. We instructed the kids to use the "repeat" loop that is found in Scratch. A square has four sides, so the sprite would have to do the move-¿turn sequence four times in a row. Four times two commands equals eight commands, which is what the students started out with.

3.4 Restrictions on Study

The most pressing restriction for this project is time. Seeing as how I am still a student, it is difficult for me to be at Cardinal Forest during the school day. This limits possible meeting times (when they don’t conflict with either Mr. Allard’s
schedule or my own) to before and after school. This of course carries with it the problem of transporting kids either to or from home, seeing as how elementary schools don’t have late buses. Despite these time restrictions, I have full faith that a computer club will be possible at Cardinal Forest and that the research project will be successful.

3.5 Reporting Results

To put data into a chart or graph for this project could be difficult, unless something of an assessment is offered to the kids at one time or another. Mr. Allard and I are reluctant to give such an assessment because we’re afraid it could discourage some of the kids from participating in the program. He and I will come up with a way to document the progress of the students. The data generated from this experiment will most likely be fairly subjective, based on my experiences with the elementary school students, and this would be rather difficult to simply place into a graph. The students were recently given a survey asking about their general knowledge of computer programs and computer programmers. The survey had three questions: “What computer programs do you use?” “What is a computer program?” and “What are computer programmers like?” There was an unforeseen problem in handing out this survey: many of the students (more or less from all of the grades) had trouble with the vocabulary. They had never heard of any software being called a "computer program" before, and this caused the survey to take much more time than it was meant to. I do not have the results from this survey yet.

4 Results and Discussion

As mentioned before, I hope that this research project will yield good results that would encourage the implementation of a simple computer science curriculum at the elementary school level. The earlier that kids can start to program and become interested in programming, the better. The computer has the potential to start a digital revolution in learning, not only in math and science but in English, and social studies as well. By the end of this initiative’s first year, I hope to show how the students have made progress in multiple school subjects as laid out by the SOL’s and that technology gives us better ways to teach the students of today.

References


TJHSST Senior Research Project
Development of a Generic Font OCR
Second Quarter Research Paper
2007-2008

Nathan Harmata

Abstract

OCR (Optical Character Recognition) is a very practical field of Computer Science. Since the late 1980’s, researchers have been developing systems to identify text from non electronic text sources, like pictures or papers. The use of OCR systems has spanned from making books in Braille to sorting mail by zip code by United States Post Office.

Keywords: OCR, Image Processing

1 Introduction

The goal of this project is to create an application that can read text from electronic picture files. One of the main focuses will be developing a generic way to recognize different fonts, rather than hardcoding in definitions for specific fonts. Although OCR is by no means a “new” field, it has still yet to be fully explored. There are very few OCR applications readily available to the public, and even the ones that are free of charge are lacking in performance and consistancy.

For now, an application that can read and recognize “simple” pictures, i.e. ones with minimal headers and only text, will be developed. If that is successful, more advanced techniques to handle headers and ’background noise,’ or erroneous markings in an image, will be used. The algorithms used to actually analyze text will be developed completely from scratch, and will be made to work irrespective of the font of the input image.

2 Background

OCR systems have been around since the late 1980’s. Still, they are not widely available or used by the public. The results from a review of the free ones on the Linux operating system are not very promising. [1] Although most of them had measured accuracies above 94 percent, that is not good enough. The one commercial product tested, Aspire OCR, was accurate only 91.5 percent of the time. The most likely industry standard, Tesseract, is also one of the oldest OCR systems. The review measured it to have an accuracy rate of 99 percent. Development on it started in 1985 and it is still used as the OCR engine for Ocropus, Google’s textual analysis application. It is unlikely that this project will be able to achieve similar success, but the goal is to be on par with the current OCR options.

3 Procedures

The current version works as follows:

The application accepts a picture file as input. The input file must be a png image of text. The current version accepts images of only text. Methods will eventually be implemented to deal with text that is actually part of a larger picture. [4] The picture is first boxed to remove excess whitespace along the border. Then, the program finds the locations of horizontal lines on whitespace inside the new image and pairs them so that paragraphs and the spaces between paragraphs become separated. A similar method is used to parse paragraphs into lines. Spaces between words are handled by comparing the size of an actual space to the average size of the spaces between letters. Lines are broken into words, which in turn are broken into letters. The idea of this is to then have individual pictures of each letter, which can be interpreted as a graph of pixels.

Each character is then compared to a database so that the best match can be determined. The database was generated from the results of a testing program (see Testing). It contains ‘definitions’ of each letter in the English alphabet per my own algorithm. Starting with an image of a single letter as mentioned above, it works as follows:

1. The image is broken into portions, referred to
Figure 1: Visualization of the 'Slope Field' algorithm. To the top is some input image. In the image on the bottom, the red lines represent the intermediate line segments and the blue lines represent the final line segments after execution of the algorithm.

as 'sectors', that pass the vertical line test. This 'Sector Parsing' is done by finding the locations of the optimal cuts.

2. Each sector image is converted into a two-dimensional array of pixels. Pixels that are whitespace and thus not text are removed, meaning that the array is comprised of only 1's and 0's.

3. Each of these arrays of pixels is then converted into a SlopeField. This is done by first averaging together horizontal groups of pixels into a single pixel, thus getting rid of unnecessary data. Then, starting with the lower left pixel, line segments between contiguous pixels are formed and their slopes calculated. Contiguous line segments with slopes similar in sign and magnitude are paired together, by treating keeping the starting point of the first one and the end point of the second one. The slope of this new line segment is recalculated, and the process continues. The result is a collection of line segments or, more simply, a collection of vertices that form the line segments.

Letters are defined by a 'Sector Vector,' which contains the number of sectors, the total number of line segments in all the sectors of the letter, and the sign of the slope of the first line segment. The comparison is performed by finding the element in the database whose distance to the 'Sector Vector' representation of the image is a minimum. A scalar is applied to certain attributes of the Sector Vector to give them more weight. For example, testing has shown that the number of sectors is a very good indicator of the value of a letter. The following expression is used to find the magnitude of the different vector between Sector Vectors A and B.

\[
\sqrt{\sum_{i=1}^{n} (\text{scalar}_i \times (\text{attribute}_i^A - \text{attribute}_i^B))^2}
\] (1)

3.1 Testing and Results

1. A user-friendly interface for viewing the pictures of parsed words has already been developed. This allows for manual visual detection of errors in letter parsing.

2. A generic testing program was developed to assess the results of the addition of new methods. It simply runs the current image transformation algorithm on every letter in the alphabet for several fonts, which were picked for their different attributes:

   (a) Arial
   (b) Comic Sans MS
   (c) Courier
   (d) Helvetica
   (e) Luxi Sans

The results were outputted to a file and a separate program analyzed them. Three relationships were determined:

1. The total frequency of each pattern (e.g. attributes of SectorVector) and the average frequency of that pattern in each font.
2. For each pattern, a list of matching letters.

3. For each letter, a frequency table of each pattern.

The following computer languages, algorithms and programs are being used.

### 3.2 Software

1. The OCR system is written entirely in Java.

2. Java’s ImageIO class is used for picture input and output.

3. Java’s BufferedImage class is used to handle pictures.

### 3.3 Algorithms/Programs

1. KolourPaint is being used to make picture files for input and to precisely view pictures.
Figure 6: Example input for OCR Manager.

2. My own algorithm which transforms an image of a letter into a collection of line segments (see Procedures) is used.

3. The current working version of the OCR system is called 'OCR Manager.' It uses the methods outlined in the 'Procedures' section and goes from the input of an image file to the output of all the possible matching words. For example, the input shown in Figure 6 generates the following output:

beak
boob
book
keek
kook

4 Conclusions

A lot of progress has been made since the first iteration of the OCR system. The original version was based off of direct comparisons to a cache, meaning that only text of the font that was cached could be read. The current, version, however has been designed to work with any font. This is done by comparisons to a cache of generic letter definitions, created by my own algorithm.

Most of the remaining work will be devoted to improving these methods. Perhaps the best way to do so is to either improve the current algorithms or to develop new ones, such that the relationship shown in Figure 4 is strengthened. This relationships measured how 'spread out' the results are; i.e. how different one letter is from another. The fewer letters in each group, the better, but at the same time, the more complex the comparison mechanism, the more devastating errors caused by 'noise' can be.

Further work will also have to be done to improve the current methods for the detection and removal of noise. There are various methods that can be used to accomplish this. [2] Overall, the successes the current version has had shows that, with improvement, it will be a viable way to implement an OCR system.

References


Abstract

My goal is to create a program that can solve the Traveling Salesman Problem, finding near-optimal solutions for any set of points. I will use genetic algorithms to try to find the optimal paths between the points. In the end, after I create a working algorithm that will find near optimal paths, I hope to create a graphic interface that will display the chosen points and the paths through those points as the algorithm runs.

Purpose

The main purpose of my project is to develop my own genetic algorithm that can hopefully find close to optimal solutions for the Traveling Salesman Problem. Once this is done I hope to modify the program to work for asymmetric problems and create a user interface that will graphically display the current problem and run the algorithm to find a solution.

This is a good problem to tackle because it is fairly complex and deals both with some complex algorithms and with some higher level math. By finding an efficient and optimal solution to the traveling salesman problem, it can be applied to the larger NP-complete field of optimization problems which can contribute to many fields of study. The TSP has been around for a long time, but more efficient programs for solving the TSPs are still being created. Many different algorithms have been used to attempt to solve TSPs, including heuristics, genetic algorithms, colony based simulations, and brute force. Heuristics are the best for finding ‘good’, but not optimal, paths fairly quickly, while genetic algorithms take longer but find more optimal paths.

The paper: "New Genetic Local Search Operators for the Traveling Salesman Problem" by Bernd Freisleben and Peter Merz details how a good way to create an algorithm for the Traveling Salesman Problem is to use a basic heuristic to find the initial pool of paths and then use the genetic algorithm on this pool of paths to find a near-optimal solution. I hope to build off of this approach by creating an algorithm that will work for both symmetric and asymmetric TSPs. Another approach that is detailed by Marco Dorigo and Luca Maria Gambardella in "Ant Colonies for the Traveling Salesman Problem" is to use a simulated ant colony to solve a TSP data set. While this is not the most efficient way of solving a TSP, it can find very near-optimal solutions. One of the most interesting articles that I found on the Traveling Salesman Problem is "Genetic Algorithms for the Traveling Salesman Problem: A Review of Representations and Operators". This article does a comparison of the different types of algorithms used to solve TSPs and their different way of representing the data. The question that I would like to answer through my project is what combination of algorithms can create the most efficient and optimal traveling salesman program.

Development

With my project, I would like to develop an efficient algorithm that can find near-optimal solutions for both symmetric and asymmetric traveling salesman problems and then incorporate it into a user interface that will run the algorithm and display the paths that the algorithm comes up with. My algorithm will be a mix of basic heuristics and the more complex genetic algorithms.

I began by creating a program that used a simple genetic algorithm that would reverse a section of a parent path which would then be replaced in the pool if it had a shorter path than the parent. I began testing this with data sets that can be found here: http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/. After finding that my solutions were off by multiple powers of ten, I discarded that algorithm and began a new one.
This new algorithm starts by creating an initial pool of fifty random, legal paths. For each iteration of the genetic algorithm it will then select two parent paths at random to create a child path from. All of the links between each point on the parent path are then compiled into one set of links. The program will then alternate choosing a link from each of the two parents to create the crossover. If the program gets stuck on a node and cannot create a legal link from the parent links, then a greedy algorithm takes over and completes the broken path.

During second quarter, I also created a heuristic to generate the initial pool of paths. I created the heuristic, hoping that it would produce better results by starting with a pool that isn’t random and it might even be faster. The heuristic I devised will first pick a random point out of all of the points the salesman must travel to. It then finds which two other points are the closest to that point and begins two paths starting at the first point, and going to each of the other two points. Then, for each of those two points, it finds the next two closest points, and creates two more new paths, thus doubling the number of paths being made. It continues doing this until there are enough points to fill the pool, at which point it will just continue by picking the next closest point, until a full traverse of the points is achieved. I will discuss how this heuristic did in my results section.

4 Results and Discussion

After testing my initial algorithm that reversed sections of the paths, I was not surprised to find that my solutions to data sets were multiple powers of ten off from the best known solutions. I knew that since my initial algorithm was based off of single parent genetics, it would not work very well.

I then created the genetic algorithm that I am currently using. When I first began testing this algorithm, my program would often fill up its pool with copies of the same path, which would prevent it from finding a solution any better than that one. In order to correct this I implemented a mutation method to free up the pool. This worked and my program ran pretty well. Using data set a280 from the TSPLIB website, the best solution that my program came up with was 2608.837612, which has an error of just 1.16 percent from the best known solution of 2579, with an average running time of about
2.15 seconds. Using the att48 data set, my program's best solution was 10820.248365, which has an error of just 1.81 percent from the best known solution of 10628, with an average running time of 3.52 seconds.

I then created my heuristic, hoping that it would produce better results by starting with a pool that isn't random, and possibly even be faster. When testing the heuristic program with the same data sets that I used to test the program with the randomly generated pool, I found that the solutions were slightly better, but the program took much longer to run. Using data set a280, the best solution that my program came up with was 2597.401845, which has an error of just .72 percent from the best known solution of 2579, with an average running time of about 5.03 seconds. Using the att48 data set, my program's best solution was 10751.542837, which has an error of just 1.16 percent from the best known solution of 10628, with an average running time of 7.31 seconds. Currently, I am not sure whether I should continue working with my heuristic program or with my randomly generated pool program, because although the heuristic program is slightly better, it takes much more time to run.

6 Appendices

6.1 An Overview of the Traveling Salesman Problem

The Traveling Salesman Problem is a problem in which a set of points is given and you want to find the shortest path that travels between each point once and then returns to the starting point. A symmetric problem is one in which the distance between towns A and B is the same as the distance between towns B and A. An Asymmetric problem is one in which the distance between towns A and B is different from the distance between towns B and A.

6.2 What is a Genetic Algorithm?

A Genetic Algorithm is a process for an algorithm that simulated genetics. First a pool of solutions is generated. Then for each generation of the program that is run, 2 of the solutions in the pool are chosen at random. These two solutions are then somehow combined to create a child solution. A fitness function is then used to determine whether the child solution is better than other solutions in the pool. If it is, then it will replace a solution in the pool. This process continues for many generations, until an optimal solution is found.

5 Bibliography


—University of Heidelberg Department of Computer Science. "TSPLIB". http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/

Abstract

Collision detection is a very useful concept, it is used in various applications from surgery to manufacturing to video game design. My project aims to create an efficient algorithm for detecting collisions so that it can be used in a gaming environment. The objects in collision will be simple solids, and multiple will be put in a space to monitor their interactions. The first step is simple 2D collisions followed by more complex 3D collisions.

Keywords: 3D, graphics, collision, collision detection

1 Introduction

The purpose of this project is to create an efficient algorithm for 3D collision detection. This project has value because there are many different applications for collision detection, and in game development, as with all other fields, efficiency is of extreme importance.

Collision detection is the concept of first detecting possible collisions, then contact, and then determining how to react to the collision. I intend to create an efficient algorithm that would detect collisions, so that the interactions of multiple solids could be modeled at once. The first step is to create a simple 2D algorithm that would model collisions as a prototype, followed by a simple 3D algorithm. This would then be optimized or redesigned, and then the number of solids in the given space would be increased, and the time taken and accuracy would be tested. The goal is to have the number of solids in space to be in the thousands, but the first benchmark would be in the hundreds.

2 Background

An important setting in which this would be used is in game development. In video games, it may be necessary for many objects to interact in space, and in video games, there can be no slowings as they are supposed to be a real-time simulation, and pauses for calculations cannot be accepted. Other applications include surgery, as simulations are used in the preparation, machining and animation.

A possibility for the algorithm would be raytracing, which uses vectors, or “rays” and uses them to detect possible collisions. Others include using bounding solids or using a simple point in polygon test, which is similar to raytracing.

3 Development

This project is an effort to create a fast and efficient collision detection algorithm. Success is considered a working algorithm that can successfully detect collisions for one hundred solids (although one thousand would be preferable). Anything less would be considered a failure.

The language used is C using OpenGL, because C is a powerful language, and OpenGL is an easily accessible graphics library.

The workplan for the project is as follows: write a 2D algorithm, then write a 3D algorithm, then optimize the 3D algorithm or rewrite it to meet my time constraints.

So far the 2D and 3D algorithms have been completed, and the next stage is to optimize the 3D algorithm. Unfortunately the current algorithm is not very robust and only works with certain solids. This problem will have to be remedied before the project can continue.
4 Expected Results

I would expect the results to be a 100% success in respect to all collisions and contact made, and the speed in which the program executes should not experience any noticeable slowages. This means that in the end, there would be a large number of solids, hopefully at least one thousand would be able to interact seamlessly.

5 Conclusions

To come after project has been finished

6 References

1. The sweep plane algorithm for global collision detection with workpiece geometry update for five-axis NC machining - T.D. Tang, Erik L.J. Bohez and Pisut Koomsap

2. Fast continuous collision detection among deformable models using graphics processors - Naga K. Govindaraju, Ilknur Kabul, Ming C. Lin and Dinesh Manocha

3. Deformable free-space tilings for kinetic collision detection - Pankaj K Agarwal, Julien Basch, Leonidas J Guibas, John Hershberger, Li Zhang

7 Appendixes
Prisoner’s Dilemma is a non-zero sum game that is applicable to game theory. The game is played between two participants or 'prisoners' and the goal is to maximize their own individual payoff at the end of the game. It is normally iterated to play through a number of rounds to determine a winner, while all the prisoners do is determine what their decisions will be with the goal of maximum payoff.

This relatively simplistic game has been through numerous variations. While the classic game itself has only two participants, some have more than two, perhaps "N" participants. My goal is not much different from creating a variation of Prisoner's Dilemma, but I'd like to expand further on the idea of cooperation. Prisoner’s Dilemma reveals a potential that can be applied to cooperation as a whole. Since the two prisoners both affect each other’s decisions, they not only would have to anticipate their opponent’s moves, but also 'cooperate' if they want to maximize their own payoff. However, this is only one possible situation out of a countless many.

Therefore, I would like to create Prisoner’s Dilemma to implement a number of parameters including the use of "N" prisoners, enabling cooperation to be known amongst others, and initiating a form of "morality". Alongside that, I would also like to find a general solution to certain scenarios.

Keywords: Prisoner’s Dilemma, Agent Based Modeling.
I want to do this project because I want to better understand the psychology of cooperation and see what types of scenarios favor certain strategies for defecting and cooperating. I also want to understand this on a larger, broader scope over the traditional two person Prisoner’s Dilemma.

1.3 Type of research
This type of research falls under applied research, although it has some aspects of user inspired research.

2 Background and review of current literature and research
I have looked into Robert Axelrod’s ‘Evolution of Cooperation’ to gain an understanding of how Prisoner’s Dilemma operates and understanding the optimal strategy tit for tat. This has been my main source of research and literature review as of this time, and I plan to continue to refer to the text for aid and reference. Prisoner’s Dilemma and the psychology of cooperation are explored very well here, therefore it is an excellent text to use as a reference and an aid.

Alongside this, I have also looked into a paper that ran a similar experiment to my own. This project enabled the participants to use a variety of strategies in several environments, as well as the ability to ‘evolve’ into a culture, allowing their strategy to change. Their goal was to find an overall optimal strategy for the given environments, and they determined that a participant must look back about six turns to make the optimal move and in the end, earn a high payout.

3 Procedures and Methodology
A visual will be constructed to show the current payouts of the prisoners playing and their choices made per turn. It will be designed to show statistics of current players, their standing strategies, and their payoff at the specific point in the game.

Testing and analysis is done by running the program and determining outputs based on certain strategies. Since strategies have now become available, the testing will come in the form of pitting a prisoner with a specific strategy against another prisoner with a specific strategy. To ensure that the results are projected correctly, the results gained will be compared to those from previous runs as well as those stated from sites on the internet.

Analysis will be done by comparing end payouts of each prisoner to their opponents. At the end of each game, the payoffs of each prisoner will be compared to determine which strategy works best for the game. I also plan to monitor payoffs during ‘mid game’ to see which strategies could possibly work best in other situations.

The testing that will be used for verification is by dynamic testing.

Requirements and specifications are currently being discussed.

Algorithms I intend to use are variations of Tit for Tat and a variation of a N-Participant Prisoner’s Dilemma program that was run on Netlogo. The project has the participants moving about freely in a universe, but the interactions between two individual participants are done through a game of Prisoner’s Dilemma.

4 Expected Results
The results I obtain from this project will be not only a simulation that can aid in just about any Prisoner’s Dilemma scenario and the real world. With the dilemma of cooperation, on whether you should cooperate or defect, being predominant in just about every point in life, I believe this can help determine what strategy will bring about the best output. The payouts given determined by strategy will give an idea of what kinds of strategies to follow for certain scenarios.

I believe this can be a contribution to seniors for next year’s tech. lab.

5 Sources
Pathfinding Algorithms for Mutating Graphs
TJHSST Computer Systems Lab
2007-2008

Haitao Mao

1 Abstract

Consider a map of an unknown place represented as a graph, where vertices represent landmarks and edges represent connections between landmarks. You have current information on whether each edge is traversible, as well past data about the availability of each connection. You have a preset destination that you want to reach as fast as possible. Pathfinding algorithms for static graphs involve computing the whole path from start to destination, but if the graph is rapidly changing, say due to some extreme environmental condition, then calculating the whole path in the beginning will not be feasible. The purpose of this project is to design and compare different pathfinding algorithms for a graph whose structure mutates to a significant extent. Algorithms may involve probabilistic theory, dynamic programming, heuristics, genetic programming, and variations of standard shortest-path algorithms such as Dijkstra’s algorithm.

2 Introduction

The problem statement is as follows: given an initial graph structure of a mutating graph, a start vertex, an end vertex, and an edge history for every pair of vertices, develop an algorithm to travel from the start vertex to the end vertex. The mutating graph will be implemented in timesteps. After each move, each edge will either stay the same or be toggled by some random function of the current state of the graph. The plan is to create a sturdy algorithm for the general case of the problem, as well as variations for specific cases where the main algorithm would not be as effective. Algorithms will be compared and analyzed to determine the circumstances for which each one is best. This project will involve both theory and actual programming.

3 Background Literature

There are little to no studies available concerning mutating graphs, so research has been focused on graph theory in general as well as the more specific topic of dynamic graphs, which may change in structure. General shortest path and flow algorithms have been reviewed. Dynamic graph algorithms and query/update algorithms have been reviewed lightly. The results from this project are expected to be completely new and original.

4 Theory and Algorithms

Define randomized distance as the distance to destination node taking the possibility of graph mutation into account. For example, a vertex with two unit length paths leading to the destination will be closer in this sense than a vertex with only one. We use steady-state convergence and methods from numerical analysis to set up a system of equations we want the randomized distances to satisfy, and solve the system. We use dynamic programming to approximate distance to heuristically closer points first, then base calculations for farther vertices on these approximations. We use the previous states of the graph: we can use this data to develop a hashmap to approximate future mutations. The hashmap stores each mutating as a mapping from the original state to the new state, and then calculates the probability of toggling states. Then, that probability is used to calculate the probability that an edge will exist in any number of timesteps. We use genetic programming to find optimal values for algorithm-specific variables, such as probability estimate multipliers and heuristic functions. We focus on sparse graphs, graphs where the number of edges is significantly less than the square of the number of vertices. The edge weights are limited to positive doubles so mutation will be somewhat controlled; edge weights that are too large will never be traversed anyway.

Currently, the algorithm proceeds chronologically, then for each vertex, it calculates the optimal
vertex in the previous time step that could have led to this vertex. It uses the history hash map to predict the graph structure at that timestep, and uses an approximation error to weight lower timesteps. Then, it backtracks to find the best vertex after the first timestep to visit. This is the main body of the working Java implementation of this algorithm:

```java
for(int v=0; v<vertices; v++) prevvals[v] = inf;
prevvals[curvertex] = 0;
for(int t=0; t<tlimit; t++)
{
for(int v=0; v<vertices; v++)
{
curvals[v] = inf;
for(int e=0; e<adjlist[v].size(); e++)
{
Edge E = (Edge)adjlist[v].get(e);
if(prevvals[E.getVertex(v)] >= inf) continue;
if(x < curvals[v])
{
curvals[v] = x;
bestprev[v][t] = E;
}
}
}
prevvals[v] = curvals[v];
if(curvals[vend] < inf)
{
if(curvals[vend] < bestend || bestend < 0)
{
bestendtime = t;
bestend = curvals[vend];
}
}
else if(t == tlimit-1) System.out.println("Time limit is insufficient for the width of this graph");
int btrack = vend;
for(int t = bestendtime; t > 0; t--)
{
System.out.println(t + " " + btrack);
btrack = bestprev[btrack][t].getVertex(btrack);
}
return bestprev[btrack][0];
```

5 Testing

The testing interface as well as the algorithms themselves will be written in Java. Since graphs are difficult to develop graphics for, output will be limited to textual lists and charts. Testing will be done by generating graph structures and initial weights and devising a system for the random edge weight mutation. Then, repeated simulations will be run and the algorithms will be scored based on their performance for various types of starting parameters.

First, the algorithms will be tested for functionality and stability by examining its pathfinding in the interface and seeing if it behaves as expected. Then, algorithms will be tested for efficiency through random and user-specified initial states. Algorithms will be compared based on how fast they can find their destination, runtime complexity, and memory usage. If the algorithms take parameters, then genetic algorithms can be used to find optimal values for the parameters.

Second quarter was devoted solely to building algorithms and developing theory for solving the problem. Testing and varying will begin third quarter. The results should be ready by the end of third quarter and fourth quarter will be focused on analyzing and improving the project and writing it up.

6 Expected Results

Results will consist of the efficiency, complexity, and stability of the algorithms tested. Results will be presented in charts, data tables, qualitative statements, and possibly graphics. Applications of the results are undetermined as this point, since this is not a commonly trod subfield of graph theory. Robots may be able to apply the algorithms in natural or man-modeled environments. The graph may be able to simulate a transportation network in order to find paths for pioneers. The randomly mutating edge weights may represent an unknown cause of change in an environment, even if there is a systematic pattern to the change. The project may be useful for applications further into the future, or may spark further development in the area which will lead to results that may be put into practice.

7 Literature Cited

References

TJHSST Senior Research Project
Sugarscape: An Application of Agent Based Modeling
2007-2008

Andy Menke

Abstract

Sugarscape was a complex program created to simulate human culture and society through the interactions of agents that travel around the world (known as a scape) collecting sugar, which they need to survive. Sugarscape allows social scientists to actually set up experiments and test hypotheses, which is normally extremely difficult. Sugarscape is one implementation of agent based modeling, which is the idea that many complex phenomena can be explained by the interactions between rather simpler agents. I am in the process of recreating Sugarscape entirely in the programming language Java and extending it, while allowing for relatively simple modification by others. I have implemented simple agent functionality and have begun working on making the agents live and die, which will help them act much more like actual living things.

1 Background

1.1 Agent Based Modeling

Agent based modeling grew out the idea of cellular automata, and may be considered a more complex form of these. A cellular automaton is a grid of cells, each of which can be in one of several states. Which state each cell is in is determined by local rules, which usually rely on what states neighboring cells are in to determine a given cell’s new state. One well known cellular automaton is Conway’s Game of Life, where cells have only two states. In agent based models, there are relatively simple rules that govern the behavior of agents; from the interactions of these agents, which may or may not be based on a grid, complex behaviors emerge. One of the earlier agent based models was Boids, which simulated the behavior of flocking birds. In general, each agent tries to avoid crowding the other agents, while staying near them and flying in the average direction of nearby agents. From these simple rules, flocks of agents that travel across the “map” arise.

1.2 Potential Problems

One problem that agent based modelling in the social sciences suffers from is its opacity. It is extremely difficult to tell from simply observing an agent based model what its behavior is or will be. Also, it is sometimes difficult to tell whether or not results are really meaningful. The results may be simply caused by the specific input parameters and not be meaningful. Another large problem with agent based societies is that it is very difficult to match results to the real world. A pattern observed in the model may be completely hidden by other factors in the real world. However, one can still learn about rules that apply to societies generally, and not just human societies (Srbljinovic).

1.3 Sugarscape

Sugarscape is one of the most complex agent based models that has ever been created. It is meant to model simple human societies and make sociology more of a “hard” science, like chemistry or physics. The basic underlying structure of Sugarscape is the scape, which consists in its simplest form of a grid of cells, each of which contains a certain amount of sugar. Agents travel around the map, collecting the sugar that they need to survive and interacting with each other. More complex versions of Sugarscape also include spice, which the agents can trade among themselves. Agents can have children together, fight with each other, and transmit their cultural attributes.

Sugarscape can be extended to study many different aspects of sociology. Flentge, Polani, and Uthmann extended Sugarscapes to allow agents to claim plots of land as their own and transmit memes that determine how agents act about the land claims of others. I hope to create a version of Sugarscape that will be easily extensible.
2 Testing

As I recreate piece by piece the work that was done in *Growing Artificial Societies*, my program should have similar results to those in the book. The results will not be exactly the same, because Sugarscape is not entirely deterministic (the order in which the agents move is random), but results will be similar no matter what random movement order is selected.

3 Procedures

3.1 Software

My program will be completely written in Java. Agent based modeling is best implemented in an object oriented programming language, and my program will take full advantage of Java for easy extensibility.

3.2 Rules

These are some of the basic rules that agents and the environment follow.

1. Sugarscape Growback Rule G(A): At each location on the scape, sugar grows back at the rate of A units every unit of time until the amount of sugar is the maximum allowed at that location.

2. Agent Movement Rule M: Each agent looks as far as it can in the four permissible directions: north, south, east and west. The agent then moves to the closest unoccupied location with a maximal amount of sugar and collects the sugar at that location.

I am working on modifications to these rules and further rules that will allow agents to act more as actual living beings. In particular, they will live and die (rather than being immortal) and breed. This will cause evolution to occur, generally leading to higher vision and lower metabolisms.

4 Expected Results

My program does not have a certain result that should come out of it. I hope to create a program that will be useful to other students and researchers working on problems related to sociology. My first major goal is to recreate the results that were obtained by Epstein and Axtell in *Growing Artificial Societies*. After that, there are no real precedents for what I will be doing; I will have to see what happens.

By the end of the year I hope to have a project that is powerful enough and easy enough to use to be released onto the Internet for others to use. I am considering releasing my project under the GNU General Public License, which allows others to use and make changes to the code.

4.1 Sample Run

I do not know how to insert pictures into a LaTeX document, so I will be doing this next quarter. Coming soon!

References


Abstract

Conway’s Game of Life is a set of rules in a two-dimensional cellular automata grid. Specifically, the Game of Life allows cells to have two states and provides them with eight neighbors. The rules state that between one generation and the next, live cells with two or three live neighbors live on, and dead cells with exactly three live neighbors come to life. This ruleset was specifically chosen by John Conway for the ability to create stable patterns as well as the difficulty of creating patterns which grow infinitely. This difficulty was rather quickly overcome by Bill Gosper’s glider gun, which opened up the ability to create binary computational devices such as logic gates. This project endeavors to first create a Game of Life interface that is suitable for the creation of large, complex patterns in the Game of Life, then use this functionality to create such a computational device in the Game of Life.

1 Introduction

1.1 Scope of Study

The first portion of this project involves the design and coding of a high functioning Game of Life interface. This interface then consists of two parts: a set of programs intended to allow users to create and run patterns in the Game of Life, and a set of programs which to allow users to search for patterns which exhibit certain end behaviors. The second part of this project involves using the previously created interface to create a computation device in the Game of Life, and explore the time and space limitations of such a computational device. This second part of the project is what necessiates the first part; it would be virtually impossible to create a computational device in the Game of Life without first having created an extremely high functioning interface and then creating the search programs. Specifically, the second objective of this project is to develop a calculator implemented with the Game of Life. This will require the development of algorithms useable and relatively efficient in the Game of Life. After a four function calculator is created, functions such as exponentiation will be added, though one major failing of any Game of Life implementation is the difficulty of representing numbers not in the whole number set. This is possible, however, and will be another extension of the computing device.

1.2 Expected results

The prospective result of this project is the development of two flexible programs, one for constructing and running designs in the Game of Life the other for searching for reactions and oscillators in the Game of Life world, and a design within the Game of Life which will take specific inputs and return the desired output. In doing this project, I will develop a flexible Game of Life interface, which will be the major new contribution to the field. The underlying ideas are not new - one man spent a number of years developing a Turing Machine extensible to universality in the Game of Life - but the completeness of the interface is unparalleled. It is not feasible to design something as large as a Turing Machine in the course of this project, though it will be necessary to create designs for specific purposes which may be new.

1.3 Type of research

This project involves pure applied research. None of the problems that this project attempts to address have not been tackled before and the trick will to find, using computer searches, patterns which have
the functionalities that will be required. This will all be new to me, but it is the nature of cellular automata that fundamental understanding is impossible to achieve: cellular automata are notable precisely for the ability to defy prediction.

2 Background

As previously mentioned, a man by the name of Paul Rendell spent a number of years developing a finite Turing machine extensible to universality in Conway’s Game of Life. His project is the extreme of complexity, but others have created logic gates, most notably Andrew Adamatzky’s LogiCell, presented in his Collision Based Computing. With respect to Rendell, two rather large extensions are possible on his work. Firstly, he created his Turing machine with extensibility to universality specifically in mind, meaning one could use his Turing machine as a template to create a universal Turing machine. In addition to that, Rendell is currently working on creating a stack cell generator, which would generate tape for his Turing machine, effectively giving it infinite tape. This project differs from the other projects in that it endeavors to create a product, a calculator, which must have a useable user interface. My approach differs from that of Adamatzky’s because I will endeavor to create one multipurpose design which will perform all of the functions and give understandable outputs, as opposed to creating a number of different circuits, the outputs of which would hinge on the state of a specific cell at some arbitrary time.

3 Development

The current edition of my program functions to run any 2 dimensional cellular automata setup using von Neumann neighborhoods. The inputs may be either clicked into the grid or loaded from a text file of on and off states. In the interface, a number of useful functions have been implemented, including the selection and running of sub-grids, copying and pasting rectangular areas of the grid, clearing selected areas, and the aforementioned saving and loading of patterns. The program is highly efficient, running at up to 200Hz in the Systems Lab computers. In order to complete my task I will continue by writing a program to search for useful patterns and reactions, and put everything together. During the third quarter, I will focus on the development of the search programs. Analysis for the functioning of my interface and search programs will be done by evaluating the speed of the programs. For the computational device, analysis will be performed by taking into consideration the space and time requirements for performing any function. Testing my program will be done mainly with the search function, which will provide me with patterns for use in my project. The purpose for using a search program is to come up with patterns that will do exactly what I want, and theoretically render program testing unnecessary. However, when I have created the calculator, testing will be rather simple; every inputs will by quickly verifiable with an electronic calculator. The design will necessarily be compartmentalized, so I can only test my program by testing specific parts of it. The major requirement that I will impose upon my program is that it work. I have no idea how much space it will take to make the patterns necessary for computation, and no idea how much time it will take to perform computation.

4 Results

At the end of this project, I can expect a functioning computational device which computes solely in the Game of Life. My analysis of this calculator will have two dimensions - time and space. I can optimize the number of generations it takes to perform any computation and reduce the amount of space it takes to do so.

4.1 Discussion

The first purpose of this project was to create a Game of Life interface that exhibits efficiency and flexibility. Although the program was written in Java, which to some extent limits the efficiency of the interface, the program is able to run at up to 200Hz while displaying every change in state. For the flexibility aspect, the interface excels; the following functions have been implemented: the selection and running of sub-grids, copying and pasting rectangular areas of the grid, clearing selected areas, and the saving and loading of patterns. In addition to loading patterns from a text file to paste onto a grid, it is possible to load a full size grid.

4.2 Conclusion

coming when I have concluded the project

4.3 Recommendations

coming when there’s stuff to recommend
5 Appendices

none yet

References


[2] Collision Based Computing, by Andrew Adamatzky
Abstract

The goal of this project is to create a three dimensional game in which most of the games content is generated procedurally (as the game is played). Referential transparency and local randomization are important for a realistic but reliable procedural generation algorithm. Such techniques as fractal geometry, multivariable algebra, and statistical analysis can be utilized to generate terrain, texture, character models and behaviors, obstacles, and other game objects, and can be modified to best complement a users playing style to create a more fulfilling, possibly endless game.

Keywords: procedural generation, midpoint displacement, referential transparency, procedural modeling

1 Introduction

Exploration into the topic will hopefully expand techniques of procedural generation into new areas, such as the random generation of 3D characters and obstacles. Common applications of random generation techniques will also be implemented, such as generating random 3D terrain and textures using fractal geometry for local detail and exploring the possibility of multivariable functions for referential transparency. Results will hopefully allow procedural generation to be used in a larger variety of circumstances, allowing for larger and more player-friendly video games. Procedural generation techniques will be applied to a 3D lion killing game. After initial development of the game, algorithms will be added to generate terrain and background procedurally. Later, as time allows, a variety of more complex implementations of procedural generation may be pursued, such as creation of enemies, obstacles, items, etc. The exact nature of more sophisticated investigations cannot be determined until more research is done on current generation abilities.

2 Background

Many techniques are out there for creating random terrain, which seems to be the most common use of procedural generation. Fractal geometry is widely used in such algorithms. Similar techniques are commonly used to create random textures, such as cloudy skies and ground. The unreleased game, Spore, is expected to be groundbreaking in the area of procedural generation, using procedural algorithms to create 3D creature models and animations. The use of 3D equations to model terrain seems to be seldom use and research is lacking, but the speed and potential of terrain functions has drawn me to the use of multi-variable equations. Hopefully my program will be completed implementing procedural generation in a new way, paving the way for further testing and experimentation. As of now, most of my research energies have been spent learning OpenGL and its GLUT library, but I've found a few articles on gameprogrammer.com and the Intel website.

3 Development

Using OpenGL and Python (and the OpenGL binding for Python, PyOpenGL), the first step will be to create a generic 3D game (a camel jousting game). A basic framework has been created for an interactive game environment with enemies and bombs. After attempting the use of fractal geometry to generate global terrain patterns, this has been determined to be too slow (almost 10 frames per second) to generate a minimum area. Multivariable equations may be better suited to generate terrain, with perhaps implementation of fractal patterns for local detail. The ultimate goal is to create a game world
generated entirely procedurally, as the game is run. The intermediate and more manageable goal is to utilize procedural techniques to create terrain and texture, along with one or two less conventional applications. Testing wouldn’t be clear cut for such a program. I need to make sure what’s generated is both random and realistic, which are not easily quantifiable measurements. Human testing would be most effective. Examination of different terrain equations is done using a Java program that can generate an interactive height map.

4 Expected Results and Conclusions

In the end, even if my ultimate goals are not realized, I will at least have contributed minor tweaks and ideas to the field of procedural generation, and hopefully even applied procedural generation to an entirely new area, paving the way to a wider range of applications. Results can be presented in several ways. Comparison of various techniques would be effective. File size is also a good measurement for how much of a video game is procedurally generated (the smaller the file, the more game content is generated during game play). Even if my contributions are minor and don’t meet my expectations, hopefully they will build upon current techniques and allow later programmers to further build upon my findings.

4.1 Literature Cited/Appendices

Coming soon.
Interactive Geometry in 3D

Jacob Welsh
TJHSST Computer Systems Lab Senior Research Project
2007-2008

Abstract

The goal of this project is to write a program that allows its user to create and manipulate a complex system of geometric objects in space. From a few basic object types, interesting and useful constructions can be built. This could be useful for education, mathematical or scientific research or visualization, or just for fun.

Keywords: Euclidean geometry, human-computer interaction, educational computing, scientific visualization

1 Background

For a while there has been software for computer assisted design (CAD), which utilizes a few basic shapes and techniques such as snapping and numeric entry to create precise, polished diagrams of a product that can then be used in its manufacturing.

A similar sort of program is used for 3D modeling, in which the user constructs polygon meshes in three dimensions: freehand; with snapping; and numerically. My program aims to be more focused on geometric objects and dynamic preservation of their relationships as some are manipulated. The leading example of this is a commercial program called The Geometer’s Sketchpad. Its interface is rather clunky, and it is limited to two dimensions. However, the fact that it is possible to build primitive 3D constructions in it illustrates the power behind the idea of geometric construction. The basic philosophy for the user interface of my program comes from the modeling program Blender and the text editor VI.

A somewhat different and interesting approach is taken by the SKETCH project of Zeleznik, Hernand and Hughes of Brown University from the mid-1990’s, and the commercial program SketchUp. With these programs, the goal is quick, informal visualization of a scene from the user’s imagination, much like a pencil-and-paper sketch in three dimensions. SKETCH is particularly noteworthy for making extensive use of mouse gestures for determining how to interpret the drawn lines. My program will aim for a simultaneous use of mouse and keyboard for maximum flexibility and efficiency.

2 Project Development

In the first quarter I worked on the basic structure of the program. A variety of functions were developed abstracting common graphics and mathematical routines. SDL and OpenGL were used as the underlying graphics libraries, but the core routines of the program are ignorant of that, calling routines that deal with geometric objects – points, lines, and the space that contains them. The original conception was that two parallel display backends would be implemented: one using OpenGL for full 3D drawing and performance, and software-only SDL routines to offer more limited support for computers lacking OpenGL. This approach was soon abandoned due to the complexity and confusion it added to the prototype program, but may eventually be added back. Another of my early ideas was that all objects in the scene would be linked in a multidirectional tree, which would allow only the necessary dependent objects to be recalculated when their parents moved. Again, this approach had to be greatly simplified for the sake of getting a working prototype off the ground.

Second quarter saw the various components of the program come together in a functional way. After extensively considering the data structures representing the geometry, I settled on a linked list of all the objects in the scene, with pointers to parent objects when necessary for correct drawing and calculation. This linked list is not actually global to the program, but rather contained in a “space” superstructure. In addition to providing a handy place for storing information like 3D projection parameters, a space completely contains a scene. This allows future features such as multiple spaces displayed in separate tabs or panes. With the data structures layed out properly, I created wrapper functions for their memory management and linking, enabling core code to be easy to write as well as comprehend. From the programmer’s
To resolve this doubt I employed some field theory. A point can be thought of as emitting a spherical field, decreasing in strength with distance from the point. Similarly, a line segment emits a cylindrical field with spherical end caps, again representing the shortest distance to a given point in space. Thus, for a certain location of the mouse pointer, the desired selection was the object whose field was strongest at that location. This has a flaw though; consider a line segment with endpoints. The distance to the segment is often equal to the distance to one endpoint. This can be resolved by scaling down the strength of the segment’s field. Distant locations will select one of the endpoints, but sufficiently close locations can still select the segment itself.

3 Results

The current program can interact by creating a chain of linked points and line segments as the user clicks on the screen. The point closest to the mouse pointer is highlighted, to illustrate the selection algorithm. While the current demonstration appears in two dimensions, all the code is scalable to three, as will become evident once the user interface is developed enough to perform rotations of the space.

The most interesting aspect of the program, however, will come with the implementation of constrained objects, such as a point lying on a line segment or point of intersection, and the addition of circles. With this, principles of Euclidean geometry can be dynamically illustrated, and more sophisticated features will follow.
TJHSST Computer Systems Lab Senior Research Project
Development of a German-English Translator
2007-2008
Felix Zhang

Abstract

Machine language translation as it stands today relies primarily on rule-based methods, which use a direct dictionary translation and at best attempts to rearrange the words in a sentence to follow the translation language’s grammar rules to allow for better parsing on the part of the user. This project seeks to implement a rule-based translation from German to English, for users who are only fluent in one of the languages. For more flexibility, the program will implement limited statistical techniques to determine part of speech and morphological information.

Keywords: computational linguistics, machine translation

1 Introduction - Elaboration on the problem statement, purpose, and project scope

A perfect machine translation of one language to another has never been achieved, because not all language expressions used by humans are grammatically perfect. It is also infeasible experimentally to code in every single grammar rule of a language. However, even a basic program that translates the basic idea of a sentence is helpful for understanding a text in a given language.

1.1 Scope of Study

I will focus on a rule-based translation system, because of time and resource constraints. I will start with part of speech tagging and lemmatization, and then progress to coding in actual grammar rules so that sentences can be parsed correctly, so that my program can handle more complex sentences as I embed more rules. I will also expand the program to incorporate limited statistical methods, including part of speech tagging and linguistic property tagging. At best, the program should be able to translate virtually any grammatically correct sentence, and find some way to resolve ambiguities.

1.2 Purpose

The goal of my project is to use rule-based methods input to provide a translation from German in to English, or vice versa, for users who only speak one of the languages. Though the translation may be simple, the program still aids a user in that it provides a grammatically correct translation, which facilitates understanding of even primitive translations. Basic translations of short passages are especially helpful for users reading less formal text, as sentence structures tend to be less complex.

2 Background and review of current literature and research

Rule-based translation is the oldest form of language processing. A bilingual dictionary is required for word-for-word lookup, and grammar rules for both the original and target language must be hardcoded in to structure the output sentence and create a grammatical translation. Most online translators currently are based off of SYSTRAN, a commercial rule-based translation system.

The more modern technique, statistical machine translation, is the most-studied branch of computational linguistics, but also the hardest to implement. Statistical methods require a parallel bilingual corpus, which the program reads to "learn" the language, determining the probability that a word translates to something in a certain context using Bayes Theorem:
They can also be used to determine linguistic properties, such as part-of-speech and tense. Usually, statistical methods are more accurate when the corpus used is larger (Germann, 2001). Statistical methods are considerably more flexible than rule-based translation, because they are essentially language-independent. Google Translate, which has access to several terabytes of text data for training, currently is developing beta versions of Arabic and Chinese translators based on statistical methods. Most research is being done with much more funding and resources than my project, and is thus much more advanced than my scope.

3 Development

The main components to a rule-based translator are a bilingual dictionary, a part of speech tagger, a morphological analyzer that can identify linguistic properties of words, a lemmatizer to break a word down to its root, an inflection tool, and a parse tree.

3.1 Dictionary

The dictionary stores a German word, its part of speech, its English translation, and any other data relevant to its part of speech, for example, for nouns, it also lists its plural form and gender. A large dictionary would be impractical for testing purposes, so I only include pronoun forms, conjunctions, and articles, with only a few nouns and verbs. These entries are stored in a hashtable, with German words as keys and English translations as values.

3.2 Part of speech tagging

The program first attempts to tag words in the input sentence using the freely available TIGER corpus, which consists of 700,000 German tokens, with each token manually assigned a part of speech. For large, full sentences, the program stores the entire corpus into a hashtable. Each unique word in the corpus serves as a key, while each table value is a list of tuples. Each tuple represents a different part of speech assigned to the word in the corpus. The first element in the tuple is the part of speech, while the second is a number, indicating the frequency of the tag’s occurrence. For single words and short phrases, it is more efficient to search for the single word in the corpus, and incrementing a separate counter for the occurrences of each different part of speech assigned to it. When a word, usually a noun or verb, is unable to be looked up in the corpus, a rule-based system is used as backoff. These rules are specific to the language being translated. For example, if a word is in between an article and a noun, it will be tagged as an adjective.

3.3 Morphological Analysis

Morphological analysis would use definite articles, suffixes, and adjective endings to determine linguistic properties such as gender, case, tense and person. It generates possible pairs of gender and case for nouns, and tense and conjugation for verbs. Two separate sets of pairs are generated for articles and modifiers, and the final list of possibilities is derived from the intersection of these two sets. To reduce ambiguity, a method for noun-verb agreement is used to determine the subject of the sentence. This information is used for lemmatization.

Morphological analysis can also be implemented statistically. Since each token in the TIGER Corpus is also assigned linguistic information such as gender, case, and number, the likelihood of a word having certain linguistic properties can be calculated. The simplest calculation would be for gender, since singular words will not change gender in different contexts.

3.4 Noun-verb agreement

Since each word will often generate several different possibilities during morphological analysis, a method for noun-verb agreement is used. The properties of the nouns nearest to the verb in the sentence are crosschecked with the properties of the verb, according to conjugation. A singular noun, if next to a singular third-person verb, will most likely be the subject of the sentence. This method helps to disambiguate verbs and nouns, by reducing the possibilities of gender, case, tense, and person.

3.5 Lemmatizer

The lemmatizer takes information from the morphological analysis and breaks a word down into its root form. For nouns, this means that plural nouns should be reduced to singular form, and suffixes resulting from different grammatical cases should be removed. When the program encounters a word that may be plural, it attempts to remove any of the common verb endings from the word: -e, -en, -er, -ern, and -s. For verbs, any ending from conjugation or tense should be removed. The program takes the few possible conjugation endings, ”-e”, ”-st”, ”-t”, and ”-en”, removes them, and adds ”-en” to the root to render the infinitive form of the word. The prefix for past-tense verbs, ”ge-”, is also searched for and removed. This saves considerable space in
the dictionary, as I do not have to code in every inflected form of every word.

3.6 Parse tree

The parse tree arranges the sentence based on dependency grammar. Verbs connect from the subject to the direct object, and articles and adjectives are nodes of nouns. In translation, this tree must be rearranged to accommodate the target language’s grammar.

3.7 Inflection

Since the dictionary lookup will only produce the root form of the translated word, a simple inflection tool is used to conjugate words, once translated into English. Inflection requires the information from the morphological analysis, which it then uses to add endings to words. Words marked as plural add an ”-s” or ”-es” to the end, as do singular verbs, depending on whether the root word ends in a consonant or a vowel. Also taken into account are common ending changes, such as words ending in ”-y” turning into ”-ies” in the plural.

4 Testing

Testing is conducted through input of sentences with new features. To test my lemmatizing component, I would input various inflected forms of a word to check the uniformity of the program’s output. To test part of speech tagging, two versions of a corpus are needed, one tagged and one untagged. The program attempts to tag all words in the untagged corpus, which is then checked against the manually tagged corpus for accuracy. Varying sentence structures can also serve as a functional test to check the validity of newly coded grammar rules in the parse tree.

5 Results

My program is able to translate a simple German or English sentence into the other language, provided the word is known in the lexicon. A statistical tagger correctly resolves most ambiguities in words. The project fulfills its purpose as a simple translator with basic grammar rules and basic statistical techniques, but would need an implementation of more advanced statistical methods to attain more flexibility in sentence structure parsing.

5.1 Word ambiguity

In German, many words can be taken to very different meanings depending on the contexts. For example, the German pronoun ”sie” can be translated to ”she”, ”her”, ”they”, ”them”, or ”you”. Though the program does attempt to resolve as many ambiguities as possible using noun-verb agreement, there still exist cases wherein even a native human speaker of German would have trouble disambiguating, such as a sentence in which both nouns could possibly be the subject.

5.2 Encoding Problems

A characteristic unique to the German language is the use of special characters in its alphabet, such as diacritic marks. Due to program constraints, these characters can not be expressed directly during input, instead substituting them for their closest equivalents: ö is expressed as ”oe”, while ß is expressed as ”ss”. An issue with the corpus lay in the corpus compilers’ attempt to encode the special characters, which ended up as garbled ASCII code when the corpus was read into the program.

5.3 Corpus Size

Though a larger corpus typically allows for greater accuracy in tagging, file size can be a constraint in many cases. The TIGER Corpus, consisting of 700,000 lines, is 42 megabytes in size, making it impractical for web-based or portable use. The amount of time spent by the program while going through the corpus also presents a problem of convenience and efficiency.

5.4 Stem changes

In general, most inflected verbs in German add a suffix, depending on its conjugation - first person singular adds an ”-e”, second person singular adds an ”-st”, and third person singular adds ”-t”. However, for several exceptions in German, the root word itself alters slightly in singular conjugations. For example, the verb ”lesen”, which means ”to read”, has a vowel change when conjugated in the third person singular, ”er liest”, as opposed to the expected ”er lest”. Only certain verbs follow this rule, which means the program cannot simply change the vowel stem when it encounters such a conjugation, but the verbs that express this quality are too commonly encountered to simply disregard. A way around this problem is to include an indicator in the dictionary entry for the word, noting that the verb is irregularly conjugated.
Similarly, German verbs are divided into "strong" verbs and "weak" verbs. Weak verbs follow a common pattern in the present perfect tense, adding a "ge-" prefix and a "-t" suffix. The program’s morphological analysis easily detects weak verbs. Strong verbs, however, have no set pattern when in the past tense, including many vowel changes. For strong verbs, the only way to resolve the problem is by manually including the past tense form for each strong verb in the dictionary.

5.5 Statistical accuracy
According to Charniak (1997), when assigning part-of-speech statistically, the accuracy of tagging should approach 90 percent when each word is simply assigned its most frequently occurring tag. Running the part-of-speech tagger on the sample corpus confirms this, yielding accuracy of around 90 percent.

References
Projects

6th Period
Study of the Evolution of Organism Combination
2006-2007

Nicholas Brown
TJHSST
Alexandria, Virginia

Abstract
Simple single celled organisms face evolutionary pressures in a similar manner to multi-cellular organisms. In order for multi-cellular life to evolve the combination of cells to form a multi-cellular organisms must be advantagous to the individual cells. The objective of this study is to create a simple fluid environment and populate it with rudamentary predator and photosynthetic organisms.

Keywords: Multiagent, Simulated Evolution

1 Introduction
Algae is one of the simplest multicellular life forms that exists in nature. The photosynthetic agents in this simulation are designed to be similar to a single independant Algae cell and will hopefully evolve into a basic Algae like grouping.

2 Background
One of the most controversial and most studied areas of science to day is evolution. One side is attempting to disprove it and the other side seeks for it to be recognized as the well verified theory that it is. One of the ways to gather evidence in this debate is through the use of simulations like this one. I am seeking to model the evolution of single cell organisms into basic multicellular ones with a single cell type and a self maintaining and regenerating structure.

3 Defining an Organism Abstractly
The abstract definition of an organism is simply a numerical summary through variables and equations of the characteristics of that organism. For example the color of an organism is modeled through an equation representing the frequencies of light that it reflects and the amounts of these reflections. There are a large number of other variables that represent things such as organelles in a cell which themselves may be bacteria that entered the cell and then mutated. The difficult part of this is determining the correct level of detail to attempt to represent that strikes a balance between simplicity and accuracy.

3.1 Defining the World Abstractly
The world that the creatures inhabit must be very simple out of necessity because it is the single most resource intensive object. The world is defined by the variables shown below.

```cpp
World::World(int sx, int sy, int sz, int l, bool b, int il, int ip, int miC, int maC, double opc)
{
    int SizeX = sx;
    int SizeY = sy;
    int SizeZ = sz;
    int light = l;
    bool loop = b;
    int initPhoto = il;
    int initPred = ip;
    int minCurrentSpeed = miC;
    int maxCurrentSpeed = maC;
    double opacity = opc;

    int[][][] environment = int[SizeX][SizeY][SizeY];
    ...
```

The SizeX, SizeY, and SizeZ variables define the size of the environment. The light variable defines the base light levels and the opacity defines how much of this light is lost with each level. The Current speed variables define the variations in the water current speeds and the init**** vaiables define the initial populations of creatures.

4 Results and Discussion
So far there isn’t any meaningfull output since I will need to write a short program to sift through the outputs and graph them or otherwise display them.

Bibs attached
Abstract

The world is becoming better interconnected. As more and more people in developing countries seek to live in economically secure ones, less and less people stay in their own. This constantly changing flux of movement highlights just how important understanding the dynamics of human population is. This project attempts to analyze and understand the growths of a population and the migrations of people across the world. Through understanding how human populations develop, we can predict changes in the future. Keywords: Population model, Logistics curve

1 Introduction

1.1 What is so important about population models?

The human population of the world is now at 6 billion and counting. It is constantly growing, constantly moving. To even try to use human power to analyze all of this data would require thousands of people and thousands of hours of man power to complete. By using computers, we can drastically cut down on the man power needed.

1.2 Why does it matter?

This project can be useful for a great variety of problems. Most prominently, the US takes a census report every ten years. But every decade in between, the census department uses the data gathered to estimate population values. A dynamic model such as the one this project would achieve would be invaluable in assisting their efforts.

2 Background

Similar research:

1. Surface modelling of human population distribution in China

2. Human Population Dynamics Revisited with the Logistic Model: How Much Can Be Modeled and Predicted?

3. A Stochastic Population Model Related to Human Populations

3 Procedure

This project works by calculating a population growth rate value using population data for a certain group. First, it starts from the states level. The growth rate is calculated for historically rich
states such as Virginia and New York. Then, it moves on to the entire US, in which it obtains the growth rate for each and every state. It takes those rates, displays a graphical representation of the growing population with it, and calculates the growth rate of population for the entire country. Finally, it moves on to the population of the entire world, including this time, the immigration and emigration of people. At this level, it calculates the world’s growth level, and shows its corresponding graphical representation.

4 Preliminary Testing

Testing at a basic level is done by comparing my project’s predicted values with those of the US census. The most recent US census estimates are of the year 2005. My predicted values for that year compared with the census’ should be relatively close.

5 Expected Results

I expect my project to currently be able to correctly predict future population values. It should also be able to display the value graphically, showing an animated change over time.

References


[2] Tian Xiang Yuea, Ying An Wanga, Ji Yuan Liua, Shi Peng Chena, Dong Sheng Qina, Xiang Zheng Denga, Ming Liang Liua, Yong Zhong Ti ana, Bian Ping Sub, Surface modelling of human population distribution in China, Institute of Geographical Sciences and Natural Resources Research, Chinese Academy of Sciences, 917 Building, Datun, Anhai, Beijing 100101, China, and College of Science, Xian University of Architecture and Technology, Xian 710055, China.


Abstract

This project studies artificial societies, especially the Sugarscape and the Schelling segregation model. To implement the Sugarscape, a display of the sugar-filled environment with agents is outputted, the simulation allows agents to harvest sugar, consume sugar, die of starvation, migrate during plagues, reproduce, and combat each other and allows the environment to grow back at a given rate and undergo plagues. To implement the Schelling segregation model, two or three distinct groups of agents are added to the environment with a preference for neighbors of their own kind to determine the effects of the individual preferences on the society at large. The reasons these two projects are being implemented is because while both are often compared, the two models in their original forms have not been combined and analyzed in a single simulation. In addition to displaying the environment graphs showing the population growth and wealth distribution are displayed. These graphs analyze what is occurring in the simulation. The program code is broken up into files: a main file, an environment file, an agent file, a location file, a display file, and a simulation file. The conclusions show that the model conforms to Axtell and Epstein’s models in the areas which were implemented. But more importantly, it shows that the simulation conforms to real world phenomena reasonably well.

1 Introduction

The program implements several aspects of Axtell and Epstein’s Sugarscape model. An environment with locations holding various amounts of sugar, which grows back over time is populated by a heterogeneous group of agents, different with respect to vision, metabolism (rate at which sugar is consumed), starting wealth, and age limit. The agents move to the locations with the highest concentration of sugar. Over time agents die and new agents are added to the environment according to a logistic function. Two graphs are displayed: one showing population growth over time and the other showing wealth distribution. The information shown by these two graphs will be analyzed.

2 Background

As of yet the Sugarscape society has not been implemented in Ruby and it would be valuable for this code to be available because of the scope of the Sugarscape research. Sugarscape has inspired further research concerning agent-based modeling and artificial societies. The Schelling segregation model was one of the first artificial societies to be implemented on a computer and has defined the area of study. The combination of these two models can provide valuable insight into human culture. Perhaps 3 different groups could be put into the Sugarscape instead of the usual two different groups. Lastly, combat between different groups will be implemented, as this has not yet been done by Tony Bigbee at George Mason.

Growing Artificial Societies: Social Sciences from the Bottom Up written by Joshua M. Epstein and Robert Axtell and Micromotives and Macrobehavior by Thomas Schelling define Sugarscape and the Schelling segregation model. Tony Bigbee from George Mason University has written the Sugarscape in Java and his code will be used for reference along with the first book primarily. In the book by Axtell and Epstein Schelling’s segregation model is mentioned and the Sugarscape is built with two separate groups (tribes) which combat against each other. The results should mirror those of the Sugarscape models in Growing Artificial Societies. However, once Schelling segregation is implemented with possibly more than two different colored populations the results will differ. In all likelihood only two groups will survive in the long run. The final results will be presented with screenshots of the running program along with graphs of relationships of variables. It will perform like previous Sugarscape models. Growing Artificial Societies and Micromo-
tives and Macrobehavior are two books which are used as references to develop this project. The article “Seeing Around Corners” shows how both the Schelling segregation model and the Sugarscape compare to various other artificial societies in the field of generative social sciences. “The Theoretical Basics of Popular Inequality Measures” examines various ways of determining inequality in a population. The measures used are: simple range, the McLoone index, the coefficient of variation, the Gini coefficient, and Theil’s T statistic. Gigliotta’s article “Groups of Agents with a Leader” examines how a leader affects a group of agents attempting to reach a goal location. The leader was effective in small groups without communication, and especially effective when he had increased vision.

3 Development

I. Theory. The algorithm driving the move method of the agents is at the core of the simulation. The agents look out in the four cardinal directions as far as their vision allows and move one square in the direction of the closest location with the most sugar. If more than one location is optimum, a random direction is chosen. See the section on agent movement in Appendix A. Agents are added to the environment according to a logistic function which follows the form of: \( P \cdot (1 - \frac{L}{K}) \) where \( P \) is the current population level and \( K \) is the carrying capacity. The section on population growth in Appendix A shows how the population growth is graphed. The inequality of the population is found using the Gini coefficient. The Gini coefficient is calculated according to the formula: \( 1 - 2 \cdot L \) where \( L \) is the area under the Lorenz curve, which is calculated using trapezoidal Reimann sums. The section on wealth distribution in Appendix A shows how the Gini coefficient is calculated and how the Lorenz curve is graphed.

II. Design Criteria. The goal of the project is to accurately represent the models it is implementing. It follows the Sugarscape design from Growing Artificial Societies by Axtell and Epstein and the Schelling segregation design from Schelling’s book Micromotives and Macrobehavior. The agents and the environment behave as they should with respect to the aspect implemented so far. The Schelling segregation model will accurately represent Schelling’s model as best as possible, but will not be perfect because concessions will need to be made to allow it to run in the Sugarscape. The information shown in the graphs and the display of the environment will be compared to the results found by the authors.

III. Materials. The program code was written in Ruby (see http://ruby-lang.org/). Tk toolkit is used for the GUI representation and graphics in the program. A text file which represents the maximum capacities of sugar in various locations in the environment was used from GMU’s Tony Bigbee’s files (he wrote a Java version).

IV. Procedures. Currently the program displays the environment, and has the agents move and harvest sugar. The display draws each location in the matrix using a circle whose radius increases based on the amount of sugar at that location. The display draws the agents as a red circle with the same radius as a location with the maximum amount of sugar. The display also shows the current time step. The GUI window has a frame containing the canvas and buttons to play, pause, and step the simulation and to quit the program. The agents themselves choose the closest location with the greatest amount of sugar. If more than one location matches these requirements, one of them is randomly chosen. Then the agent harvests the sugar and consumes from his own supply of sugar. At each time step the sugar in the environment grows back by one. The program begins with a small number of agents and adds to the population using a logarithmic function so that it reaches carrying capacity. Modifications in the individual agents include an improved move method, a random age limit, and a variable for red or blue color. The GUI window has been modified to include buttons to change the graph and change the refresh rate. There is an input box to change the refresh rate in the display of the environment and of the graphs. The two graphs which are now displayed are the population growth over time, and the percent of total wealth over the percent of the population (Lorenz curve). To get the population graph, it keeps track of the length of the array of agents at each time step in the simulation file and cycles through the array of population values in the display file. To get the wealth graph, it cycles through the array of agents and stores the wealth of each individual agent. Then it sorts this array and cycles through it keeping a running total to determine percents.

4 Results

It has been determined that the program meets the design criteria in the areas in which it was implemented. The graphs are what answer many of the experimental questions. Descriptions of the population growth graph refer to the section on population growth in Appendix B. In general it follows the shape of logistic graphs which are proven to be a fairly accurate representation of population growth.
Growth is slow when the population is close to zero and close to the carrying capacity, and growth is highest at half of the carrying capacity. The few anomalies reveal certain aspects of the simulation. The initial portion of slow growth is smaller than the final portion because the population begins with three individuals instead of one (but starting with one agent would not completely remedy this). The oscillations near carrying capacity come from the age limit of agents. It takes longer for the population to decrease due to dead agents than it does for it to react to the added agents. The oscillations decrease over time and will eventually disappear. At about half of carrying capacity the line begins to become jagged instead of fairly straight like it was earlier in the simulation. This is a result of the heterogeneous population. In the beginning even agents with low vision and high metabolism (less fit agents) have room to survive in the regions of abundant sugar. As the environment fills up only better fit agents can survive on the fringes, areas with less sugar, so many added agents die quickly. The effects are even more pronounced as population approaches carrying capacity. Descriptions of population inequality refer to the graph in the wealth distribution section of Appendix B. At first a bar graph was used to represent wealth distribution, but it was replaced with the Lorenz curve. Both conform to the graphs in Axtell and Epstein's book. They show that there are very few wealthy agents (agents with a lot of harvested sugar stored) and many poor agents. In this sense the population is pretty unequal. The Gini coefficient is a numerical representation of this phenomenon. A coefficient of zero represents perfect equality and one represents perfect inequality (one agent has all the wealth). The number is just over .5 showing that the Sugarscape population is closer to perfect inequality than to perfect equality.

5 Further Research

Further research could include implementing other aspects of the Sugarscape, as described by Axtell and Epstein. Possible topics include reproduction or the trade of spice. In addition, other studies of artificial societies (like Schelling’s segregation model) could be analyzed using the Sugarscape as the base environment. Changing the range of values in the heterogeneous aspects of the agents yields different results in the graphs. This could be attempted to be quantified. Combat could be implemented, a desire expressed by Tony Bigbee. Genocide, as described in “Seeing Around Corners” could be implemented. The manner in which agents are added to the environment could be done according to other functions to show other phenomena, like exponential growth. In addition to changing the environment, the method of determining social equality could be determined using some of the different methods described in “The Theoretical Basics of Popular Inequality Measures.” Lastly, the Sugarscape could be implemented in other languages, like assembly for example.

Bibliography


Appendices

Appendice A

Code

agent move method

```python
def move2
choices = []
4.times {choices << [@@env[@posY][@posX],-1,0]}
choices[0] = nil if @posY+1 >= @@env.length or
@@env[@posY+1][@posX].hasAgent != -1
choices[1] = nil if @posY-1 < 0 or @@env[@posY-1][@posX].hasAgent != -1
choices[2] = nil if @posX+1 < 0 or @@env[@posX+1][@posY].hasAgent != -1
choices[3] = nil if @posX-1 < 0 or @@env[@posX-1][@posY].hasAgent != -1
return if choices == [nil,nil,nil,nil]

for k in 1..@vision do
  if choices[0] != nil and @posY+k < @@env.length and
    @@env[@posY+k][@posX].sugarquant > choices[0][1]
    choices[0] = [@@env[@posY+k][@posX].sugarquant, @posX+1, choices[0][1], k]
  else
    choices[0] = [@@env[@posY+k][@posX].sugarquant, @posX, choices[0][1]]
  end

  if choices[1] != nil and @posY-k >= 0 and
    @@env[@posY-k][@posX].sugarquant > choices[1][1]
    choices[1] = [@@env[@posY-k][@posX].sugarquant, @posX-1, choices[1][1], k]
  else
    choices[1] = [@@env[@posY-k][@posX].sugarquant, @posX, choices[1][1]]
  end

  if choices[2] != nil and @posX+k < @@env.length and
    @@env[@posX+k][@posY].sugarquant > choices[2][1]
    choices[2] = [@@env[@posX+k][@posY].sugarquant, @posX+k, choices[2][1], k]
  else
    choices[2] = [@@env[@posX+k][@posY].sugarquant, @posX, choices[2][1]]
  end

  if choices[3] != nil and @posX-k >= 0 and
    @@env[@posX-k][@posY].sugarquant > choices[3][1]
    choices[3] = [@@env[@posX-k][@posY].sugarquant, @posX-k, choices[3][1], k]
  else
    choices[3] = [@@env[@posX-k][@posY].sugarquant, @posX, choices[3][1]]
  end

for k in 1..@vision do
  if choices[0] != nil and @posY+k < @@env.length and
    @@env[@posY+k][@posX].sugarquant > choices[0][1]
    choices[0] = [@@env[@posY+k][@posX].sugarquant, @posX+1, choices[0][1], k]
  else
    choices[0] = [@@env[@posY+k][@posX].sugarquant, @posX, choices[0][1]]
  end

  if choices[1] != nil and @posY-k >= 0 and
    @@env[@posY-k][@posX].sugarquant > choices[1][1]
    choices[1] = [@@env[@posY-k][@posX].sugarquant, @posX-1, choices[1][1], k]
  else
    choices[1] = [@@env[@posY-k][@posX].sugarquant, @posX, choices[1][1]]
  end

  if choices[2] != nil and @posX+k < @@env.length and
    @@env[@posX+k][@posY].sugarquant > choices[2][1]
    choices[2] = [@@env[@posX+k][@posY].sugarquant, @posX+k, choices[2][1], k]
  else
    choices[2] = [@@env[@posX+k][@posY].sugarquant, @posX, choices[2][1]]
  end

  if choices[3] != nil and @posX-k >= 0 and
    @@env[@posX-k][@posY].sugarquant > choices[3][1]
    choices[3] = [@@env[@posX-k][@posY].sugarquant, @posX-k, choices[3][1], k]
  else
    choices[3] = [@@env[@posX-k][@posY].sugarquant, @posX, choices[3][1]]
  end
```

```ruby
```
choices[2] = [@@env[@posY][@posX+1],@@env[@posY][@posX+k].sugarquant,k]
end

if choices[3] != nil and @posX-k >= 0 and
@@env[@posY][@posX-k].sugarquant > choices[3][1]
choices[3] = [@@env[@posY][@posX-1],@@env[@posY][@posX-k].sugarquant,k]
end
end
choices = choices.compact
choices.sort! { |x,y| y[1] <=> x[1]}

while choices[-1][1] != choices[0][1] and choices[-1][2] != choices[0][2]
choices.pop
end

i = rand(choices.length)
return if choices[i][0] == nil
@posX = choices[i][0].posX
@posY = choices[i][0].posY

population graphing method

def drawPop
$graph.delete(:all)
Tkcline.new($graph,40,40,40,$w-40)
Tkcline.new($graph,40,$w-40,$w-40,$w-40)
Tkctext.new($graph,$w/2,30,:text=>"Population levels over time",
:font=>['Helvetica',15,'bold'])
Tkctext.new($graph,10,$w/2,:text=>"% of Population",
:font=>['Helvetica',10,'bold'])
Tkctext.new($graph,$w/2,:text=>"Time",
:font=>['Helvetica',10,'bold'])
for n in 1...popLength do
break if getPop[n] == nil
Tkcline.new($graph,50+(n-1)*400/$popLength,$w-40-count[0]*y,40+n*x,$w-40-(w-100)*count[0]-y)
giniAr += (count[0] + 0.5 * (count[1]-count[0]))*dx*dy
end
Tkctext.new($graph,30,$w-30,:text => "0%")
Tkctext.new($graph,440,$w-25,:text => "100% (#{$env.agents.length})")
Tkctext.new($graph,40,15,:text => "100%")
Tkctext.new($graph,40,30,:text => "(<#{totalW})")
Tkctext.new($graph,150,150,:text=>'Gini coefficient: \
%f' % (1.0-2*giniAr),
:font=>['Helvetica',10,'bold'])
end

Lorenz curve graphing method

def drawWealth
$graph.delete(:all)
Tkcline.new($graph,40,40,40,$w-40,:arrow=>:first)
Tkcline.new($graph,40,$w-40,$w-40,$w-40,:arrow=>:last)
Tkctext.new($graph,$w/2,30,:text=>"Wealth Distribution",
:font=>['Helvetica',15,'bold'])
Tkctext.new($graph,40,30,:text=>"(#{totalW})")
Tkctext.new($graph,40,15,:text=>'Gini coefficient: \
%f' % (1.0-2*giniAr),
:font=>['Helvetica',10,'bold'])
end
Appendix B
Graphs
population growth

wealth distribution
1 Abstract

The goal of this project is to be able to resize an image without distorting any important aspects of the image. Commons methods of resizing, including cropping and scaling, remove or distort some of the image and are thus undesirable. By finding the least important pixels and removing them, this dynamic resizing can be possible. These can be found by finding the change of intensity of each pixel to the next and taking away the ones with a very low change. Using this method, humans should be unable to tell if an image has been altered.

2 Introduction

Currently on the web, there is such a thing as dynamic text formatting. For instance, when you resize a web browser window, the text in it will adjust itself to fit inside the window while still being readable. There is nothing like this for images however. My goal in this project is to be able to change the dimensions of an image without losing important content, such as the dimensions of the focus of these pictures.

3 Background

Edge detection is being researched heavily in modern times. Many teams are trying to allow computers to see and identify objects. But there is also much research being conducted about images and modifying them. There is one project called PhotoSynth that is trying to take a large amount of images from the web, and from them, create a 3D model of whatever the images are of. There is also another project that is very similar to what I am trying to do, although I have some ideas for my project that they have not yet implemented.

4 Development

I will be using C for all of my programming. In order to resize the images, the program will first convert the image to grayscale by averaging the red, green and blue values for every pixel. In order to find the least important pixels (the ones that should be removed to harm the image the least), the gradient magnitude function will be placed on the image. What this function finds is the rate of change of intensity of the grayscale image (or the first derivative of it). From this gradient image, the program will find the path of pixels from one side of the image to the other that is the least important by using the cumulative sums of the gradient image. This will be determined by the smallest changes in intensity. These pixels will be removed and this process repeated until the image has the desired dimensions.

5 Results

As of now, the program is capable of shrinking the image in both the horizontal and vertical directions. With the newest method of finding the optimal pixels to remove (the cumulative sums of the gradient image) the final images are usually very realistic. It is difficult to notice that it has been modified at first glance, which is the goal of the project.
Agent Based Simulation, Negotiation, and Strategy Optimization of Monopoly

Nicholas Loffredo

Abstract

Computers have a difficult time performing common human tasks, such as learning a language well enough to be able to "talk" intelligently with someone/something. Monopoly, one of the most well known and understood board games in the United States, if not the world, provides a good environment to see whether or not a computer can "learn" to negotiate through a number of strategies. It is much simpler than negotiating in the real world, due to the simplified environment, yet complex enough that it may be useful as an example of computer negotiation. By creating a Monopoly simulation with computer agents playing the game, it can be used as a test bed for these computer negotiations. The methods used in this test bed, if it works, could then be applied to more complex computer negotiation. The agents can be given aggressiveness values for different negotiation techniques, such as price "stubbornness" when selling or buying properties from other agents. The results from running the simulation hundreds of times can then be graphed to show which strategies are the optimal strategies for agents.

1 Introduction

Computers currently are unable to perform common human tasks such as understanding a language well enough to speak it and effectively communicate. A good example of this is negotiation. Most humans are able to negotiate with one another for various goods. Computers, on the other hand, can't. If computers were able to negotiate effectively, they could be used in many situations that currently require people such as in diplomacy, selling/buying goods, trading goods, or just negotiating with other people in general. More importantly, it would allow people to instruct a robot/computer to negotiate using certain items and to meet certain goals, instead of hiring people to do it. These computers would be resistant to common human flaws, such as anger or impatience.

Making a computer than can negotiate effectively in a limited environment is a first step towards being able to negotiate in a more complex one. The game of Monopoly is simple enough that negotiation should be able to be implemented within a year, yet complex enough that the method used to achieve the results may be able to be applied towards real negotiation. By making a working simulation of Monopoly, a negotiation capability can be implemented for computer agents that will play the game.

Fundamentally, the system must simulate all the rules of Monopoly. Agents must be able to move around the board based on the dice roll, be able to buy titles they land on, and buy houses on monopolies they own. Additionally, they should be able to sell houses and mortgage properties. When an agent lands on a Chance or Community Chest square, they should receive the top card from a deck which was randomly sorted before the game, and do whatever the card says. Furthermore, to explore the research areas contained herein, agents should also be able to negotiate with players. In particular, these negotiations will be based on aggressiveness levels. For example, how far an agent is willing to drop/raise his initial price in order to complete a negotiation.

2 Background

Surprisingly, not much research has been done into making agents for Monopoly that can learn the optimal strategy for negotiation. One of the few existing simulations is one that determines the probability of landing on each square in Monopoly. This can be used to see if my Monopoly simulation results correlate to theirs, and determine whether or not the simulation works correctly. Fortunately, there has been research done into reinforcement learning, which is effectively how the agents will learn. Reinforcement learning is when agents take a number of actions over a course of a game, and then are basically told that they did well (when they won) or they did poorly (when they lost). Based on this feedback, each agent will try to change its aggressiveness values (which may involve different strategies) to find the winning values. Also, agents may learn from the strategies other agents used, and whether they won
or lost, to determine how their aggressiveness levels should change, which varies from the traditional approach slightly. There is not a state-of-the-art reinforcement learning program yet, however. Everything I Need to Know About Business I Learned from Monopoly, which discusses various strategies for Monopoly, can be used to see if agents develop the strategies the book discusses.

3 Procedures

3.1 Preliminary Testing and Results

The simulation was tested by adding features that allow the program to play multiple games in one run, display the number of wins of each agent, and display the aggressiveness levels of each agent. Both agents initially had a 50 percent chance to take actions. One agent, however, was allowed to learn, whereas the other would not change. Doing this would show whether or not the agent was actually evolving to beat the other agent - i.e., if his win ratio over a large number of games was greater than 50 percent. After running the simulation about 100 times, the learning agent had roughly a 60 percent chance of winning. This shows that the agent was learning. The likely reason that the chance of winning was not higher is that the agent was not able to trade properties - therefore, whether or not the agent wins is highly influenced by whether or not he lands on enough properties first to be able to buy a monopoly.

3.2 Software

Java is the programming language being used.

4 Schedule

In the first quarter of the research project, the focus will be on designing the Monopoly environment with all the correct rules.

The second quarter will focus on the researching, planning for, and initial designing of the negotiation capability of the agents. (e.g., based on an array of aggressiveness values for different items).

In the third quarter, the design will be completed, as well as the implementation and testing. At this point, experiments will be designed and executed, and the results will be analyzed. Conclusions will be drawn, and recommendations will be made. Graphs will be used to display the results of how often agents win with various strategies.

5 Development

In the first quarter of the project, I developed the monopoly simulation program, finishing all features of monopoly (agents 'roll' dice, buy properties, etc.) except for auctions, trading properties, buying and selling houses, and mortgaging properties. I also created an interface to display the monopoly board.

In the second quarter of the project, I finished implementing buying and selling houses, and mortgaging/unmortgaging properties. After that I created and implemented aggressiveness levels which are used to determine how often an agent will take certain actions for each property group - i.e. buying properties, buying/selling houses, or mortgaging/unmortgaging properties. When that was finished, the learning algorithm for the agents was implemented so that agents became able to learn over time which values are optimal for different actions on each property. A display was then created to show the aggressiveness levels and wins of each agent.

6 Bibliography


The Applications of Image Processing Techniques to Sign Language Recognition Through a Web Camera Interface

Byron Hood

Abstract

Sign language recognition is the first step in a long road towards natural language processing, or the ability for a computer to “understand” naturally spoken (or signed) language. Such an invention would drastically lessen the amount of time required for user-computer interaction, by as much as a factor of two. This project explores using image processing techniques such as edge detection, line detection, and line interpretation to identify sign language as it is performed, using frame-based input from an average web camera (“webcam”). When research and programming is complete, it is expected that the program will be able to identify the sign language gestures for alphanumeric characters with a high degree of accuracy, in real time.

1 Introduction

1.1 Purpose

The purpose of this project is to provide an interface for people who use sign language for everyday communication with others (referred to as “sign speakers”) to enter input into a computer using their native form of communication. Such input could theoretically increase their input speed twofold (see the explanation under Background) or possibly even more. In addition, such interpretation of sign language gestures and hand positions is a first step towards fulfilling a dream of human-computer interaction nearly as old as the machines themselves: natural language processing, or the concept of a computer “understanding” naturally spoken (or in this case, signed) language and performing actions based on the gestures or speech interpreted.

1.2 Scope

This project will require some research into sign language hand positions and gestures, but more specific and deeper research is necessary into “image parsing” techniques. These include edge detection, line-finding, and methods of line classification. Additionally, as the program will attempt to insert characters into the computer’s input system, this will require some heavy digging into either windowing system input code (so that the program can feed into X windowing system input) or operating system-level input (so that the program can effectively emulate a keyboard or other such input device). Further, considering that data concerning hand positions will be stored in XML files to be read and parsed during the running time, to ease the process of editing hand positions, and also to make reading the said data simple when starting up; yet the project will also need some basic research into XML parsing schemes and methods of storing the XML data in memory.

To limit the overall complexity of the program, the final application will recognize and interpret only alphanumeric characters. The further this program goes in terms of recognizing additional characters, the more hand positions necessary to differentiate all of these different symbols. If the program continues to expand, the hand positions, by necessity, will become closer to each other and distinctions between different characters are made smaller and more difficult to determine. This, in turn leads to a higher rate of error; error that increases exponentially as one adds symbols to be recognized. A more practical approach, therefore, is to designate a reasonable number of different characters, and then to extend this basic program later on as image processing techniques improve and the rate of error decreases.

2 Background

In today’s society, people with auditory and locutory disorders usually opt to communicate using sign language, a silent variation on the local language which uses body language: gestures, mouthing, and hand/finger positions, instead of spoken words. Through extensive practice and use (as normal people might gain extensive practice speaking their native language), many sign speakers are capable of “speaking” as fast as non-impaired people speak orally, about 200 words per minute in normal conversation[2].
If the average word is taken to be six letters long and one accounts for a slight speedup due to the short amount of time required to communicate a single letter, spelling a word out letter by letter will likely reduce speed by a factor of four for both sign and oral speakers. Nonetheless, this is a hefty 50 words per minute, and compares quite favorably to average typing speeds. According to Karat, et. al. the average computer user can type 33 words per minute while copying text, and this drops to a mere 19 when composing[3]. Therefore, an average computer user will spend from two to three seconds typing any given word, whilst a sign speaker (could he or she sign into a computer), would spend half of that time. Finally, the “QWERTY” keyboard was designed expressly for the purpose of (slowing) typists down (this is a throwback to the days of typewriters, to help prevent jams). Therefore, a person signing has a double advantage over a person typing: first of all, they are not inhibited by the popular keyboard, and secondly, they sign faster than the average person types.

While extensive and highly specific research has been done in the field of computer vision (as shown by the sheer number of books available on the subject), little has been devoted to the recognition of sign language, and only one study[7] has considered using a webcam-computer setup (most other modern research explores using a specialized glove to transmit data back to the computer). This is for a combination of reasons: first of all, processor power was formerly far too expensive and not powerful to process a multitude of images (with a high enough resolution to distinguish sophisticated shapes such as the human hand) in anything close to real time. Additionally, the keyboard has been—and remains—an effective, flexible, cheap, and easily extensible tool for computer input. Finally there is as of yet little demand for such a novel mechanism of input.

2.1 Methods & Concepts

2.1.1 Edge detection

“Edge detection” is the process of highlighting differences in pixel intensity and pixel color over an image. It follows that an “edge” in this context is a small area of an image where either pixel intensity or pixel color is changing rapidly. At the very beginning of the research associated with this project, I analyzed various methods of doing this, listed below.

**Horizontal differencing** This method, along with vertical differencing, is the fastest and the least computationally intensive. However, the speed comes at a cost: this method is highly inaccurate and only manages to find edges with any degree of accuracy when these edges are near to vertical, or, rather, when (part of) the image is changing rapidly from left to right. The equation for this method is

\[ V = |P_b - P_t| \]  

where \( V \) is the value of the computed pixel in a new image which contains the outline of the image being subjected to edge detection, \( P_b \) is the value of the pixel below the current pixel, and \( P_t \) is the value of the pixel below the current pixel. This equation is applied to every pixel in the image, not including the top and bottom rows. If the differences are in a vertical direction, then this method will not recognize them. I quickly discarded this method of edge detection because a very important part of the edges formed by the outline of a hand would be missed by this method.

**Vertical differencing** This method of edge detection is nearly identical to horizontal differencing, explained above, except that this method registers changes in a vertical perspective. In the same way that horizontal difference’s great weakness is missing any changes which occur vertically, this method does not record any edges which occur horizontally. And just as horizontal differencing is inadequate for the purposes of detecting the edges in a hand, vertical differencing misses crucial horizontal differences. The equations are also very similar:

\[ V = |P_t - P_r| \]  

where \( V \) is again the value of the computed pixel in a new image which contains the outline of the image being subjected to edge detection, \( P_r \) is the value of the pixel to the right of the current pixel, and \( P_t \) is the value of the pixel to the left of the current pixel. I discarded this relatively quickly as well for the same reasons as I discarded horizontal differencing.

**Robert’s Cross** Although strikingly simple, the Robert’s Cross method delivers good results for edge detection. Although not superb, because it misses the finest details, it finds all necessary lines. Plus, it minimizes the amount of noise from the joints of fingers and the joints between the fingers and the palm. The equation for Robert’s Cross, applied to every pixel, is:

\[ R = \sqrt{(P_t - P_b)^2 + (P_t - P_l)^2} \]  

Where \( V \) is the value and \( P_t, P_b, P_l, \) and \( P_r \) are the pixels above, below, to the left, and to the right, respectively, of the pixel on which the edge detection
is being performed. I did not immediately discard this method as it performed reasonably well. My final decision was for this method because it is the best balance between detecting too little and too much, and also was not overly intensive on the processor.

**Sobel’s Operator**  This method performed even better than Robert’s Cross in terms of finding the edges in an image. While Robert’s Cross might find faint traces of small, non-distinct edges, Sobel’s operator would highlight those strongly and find edge where the eye couldn’t have found a difference. This precision was made possible by the following operations, assuming the pattern

\[
\begin{array}{ccc}
    a & b & c \\
    d & e & f \\
    g & h & i
\end{array}
\]

around pixel \(e\). The general idea of the equations is that they account for all eight of the neighbors of each pixel, while Robert’s Cross accounts for only four, and horizontal and vertical differencing two.

\[
V = \sqrt{(c + 2f + i - a - 2d - g)^2 + (a + 2b + c - g - 2h - i)^2}
\]

where \(V\) is the final value of the pixel. I eventually chose against this method, and elected to continue with Robert’s Cross, for two reasons. First of all, this method is more computationally intensive, and requires more memory accesses. While over a single pixel the difference is negligible, the difference over 307,200 pixels (the number of pixels in a 640x480 image) is far greater. Secondly, this method highlights too much detail, bringing out parts of the hand that I would prefer not to be visible in an image of edges.

2.2 Prior Research in this Field

As very little substantial work (outside of Kraiss and Zieren’s research[7]) has been done in this field, I am pretty much a pioneer and I must decide what path to follow on my own with little outside guidance from previous products and plans. This adds a new element of interest for me: success means that my program is one of the first of its kind in the world. Although others have performed studies and have even programmed sign language recognition systems, they have all used some form of aid to recognize the hand: they have used a mechanical glove, or a colored glove, or a very specialized background. I intend to write this with no such requirements.

3 Development

3.1 Requirements and Limitations

A part of this project is to provide a relatively portable interface for human-computer interaction with a webcam. Therefore, the requirements of this program are rather basic. All that is necessary is a webcam—the basis of the application—and the associated drivers, a computer with Linux installed, and finally Video 4 Linux. To compile from source, a C compiler is also necessary.

In terms of sign language recognition, the boundaries of this program will exist in terms of letters “understood” and accuracy. To simplify matters, the first program will only deal with alphanumeric characters, to provide a large enough distinction between letters to minimize some of the factors that might otherwise impede position recognition. In addition, the program will also have a limited quantity of time in which to analyze each frame, ranging from \(\frac{1}{4}\) to \(\frac{1}{2}\) of a second. The goal of acting in real time precludes any deep analysis of each image, and so therefore the program will inherently be somewhat inaccurate (although this may not be as much of a disadvantage as it seems; people make many mistakes at their keyboards as well).

3.2 Plan for Development

Originally, my plan was to program in the order of most testable programs first: first edge detection,
then line-finding, then a line-interpreting AI, and finally an image-capture program to “grab” frames from a webcam. I soon found out, however, that this was impractical because the only good way to test a line-finding AI is to use a variety of images representing a variety of letters (otherwise I might just end up fine-tuning to a specific letter or image thereof and breaking compatibility with other letters). The best way to generate a multitude of realistic images is to capture them from a webcam; therefore my plan changed. Instead, I am working on perfecting a capture mechanism first, and then line finding and interpretation (I have already completed the edge detection phase). In brief, my plan is to program in the order of execution in the planned final product.

3.3 Testing and Analysis

The plan for testing my program(s) is rather straightforward: I will use a Python script to run each program several times and report the results and timing. Afterwards, I will inspect the image results from each portion (except the line-interpreting AI) to ensure that it is correct. In each circumstance, I will test ordinary conditions/images, boundary conditions/images, and images or conditions which should be discarded. For example, some very basic testing of four algorithms yielded these times:

This graph shows that the total processing time per frame is currently around 280ms, a little bit higher than the ceiling of 250ms per frame so that I can interpret sign language in real time at an acceptable pace. However, this calculation does no include line interpretation, so this figure of 280ms is subject to increase.

References


C to Java Programming Language Translation

Jamie McAtamney

Abstract

With the modern emphasis on program portability and the new need to run programs on multiple computers in networks or over the Internet, it would be very useful for C programmers to be able to translate either legacy or newly-written C programs into Java to make them more portable; however, currently translation by hand is seen as too tedious and time-consuming, while computer algorithms to do so are not very accurate. A combination of keyword search/replace and algorithms to translate C structs to Java classes and C “include” modules to Java “import” modules can help alleviate or solve the problem of tedious or inaccurate translations specifically between C and Java.

1 Introduction

Both the C programming language and the Java programming language are versatile, powerful, and popular among programmers. C is commonly used when creating operating systems, network interfaces, and other programs which require the ability to manipulate memory usage, binary data, and similar low-level constructs. Java has two major advantages over C, however. The first is its modularity, as it is capable of being used on any platform and any operating system, while implementations of C are platform-specific and must be recompiled or sometimes rewritten when moved from one computer to another. The second is the fact that coding in Java is easier for the programmer than coding in C, as details such as memory usage and data size are not handled by the programmer but by the Java Virtual Machine. For these reasons, translation of programs from C into Java are most beneficial when programs are required to run under different operating systems or machine specifications, when a less-experienced programmer needs to modify a program originally written in C, or a combination of the two, though these are by no means the only scenarios under which translation would be beneficial.

Shifting from the programmer’s perspective to a consideration of program functionality, there are three major groups of programs that benefit from translation from C to Java. First are “legacy” programs that were originally written in C to take advantage of its higher execution speed; as modern computers have more memory and run faster than those of even a few years before, these “legacy” programs would gain more from added portability than they would from remaining in C. Second are programs wherein the majority of the code implements simple algorithms such as string tokenization, data storage and manipulation, and the like; Java already has several implementations of algorithms such as these built into it, so code could be simplified and shortened. Third are programs that will be used either over a network or the Internet; while C has methods for sending and receiving information between different computers, any programs that require a user interface on the other end of transmissions would benefit greatly from Java’s portability and its already-implemented applet system.

2 Background

While the differences among programming languages have been studied extensively in comparative languages courses and otherwise, little progress has been made in the area of automated programming language translation. One company, Jazillian, provides translations among a limited number of languages for a fee, but significant client involvement is required to tailor the algorithm to the program’s intended use. The problems involved with automated translation occur because programming languages are too dissimilar for direct word-for-word translation. Python and Ruby, for example, do not declare variables and use indentation instead of braces—“{” and “}”—to denote blocks of text, in comparison to the C and Java methods of declaring variables and separating code. Even syntactically similar languages such as C and Java have differences that make simple search-and-replace difficult. For example, while the C char arrays have an analog in Java Strings, because they are two different data structures the methods for accessing them are very different, and this discrepancy must be taken into account. A related difficulty is C’s use of pointers: A “string” in C is not simply an array of chars, it is a pointer to an array of chars—expressed as “char*”—which means that string comparison methods, string search methods, and the like are required; one cannot simply copy, compare, or otherwise manipulate strings in the same way.
one may manipulate ints or chars.

Computer scientists A.A. Terekhov and C. Verhoef have written a paper entitled “The Realities of Language Conversions” which describes many of the various problems facing automated language translation systems and upon which I am basing my preliminary work. They describe in detail the some of the differences in structure, implementation, and syntax I mentioned above, and after I progress to the stages where I will have to deal with each problem I will be able to discuss the difficulties in more detail in later versions of this paper. Computer scientist Lili Qiu has written a paper entitled “Programming Language Translation” which both describes in greater detail the rationale behind translation of lower-level languages into higher-level languages, with the specific example given of translating C to C++, and some of the differences between procedural languages (like C) and object-oriented languages (like Java).
Analysis of spectro-temporal receptive fields in an auditory neural network

Madhav Nandipati

Abstract

Neural networks have been utilized for a vast range of applications, including computational biology. But the realism of these models remains in question. In the models of the auditory cortex, for example, the properties of neuronal populations are hard to fully characterize with traditional methods such as tuning curves. Spectro-temporal receptive fields (STRFs), which describe neurons in both the spectral and temporal domains, have been obtained in a variety of animals, but have not been adequately studied in computational models. The aim of this project is to address these issues by generating the spectro-temporal receptive fields on a basic, neural network model of the early auditory processing stages in the brain. This novel use of STRFs is also evaluated against tuning curves in describing the properties of the neural network.

Keywords: neural network, auditory processing, spectro-temporal receptive field (STRF)

1 Introduction

Neural networks are utilized for a wide range of applications in both mathematics and science, and recent efforts have been made to mimic the processing that occurs in the brain. Although many of these models revealed how well we understand the brain, the realism of such models remains in question. In this paper, I discuss the use of receptive fields in a neural network and application of receptive fields to assess the validity of computational models.

The brain is a remarkably complex organ that is responsible for the intricate processing of tactile and abstract information. The hominine capability of sight has already been widely studied and well documented in research and modeling. Yet the intrigue of sound and the auditory processing in the brain remain is still surprisingly new territory. Realistic models of the auditory cortex will give scientists the tools to better understand and hopefully mimic these complex functions in artificial systems.

Spectro-temporal receptive fields (STRFs) are visual descriptions of the linear properties of auditory neuronal populations. STRFs accurately describe both the spectral (frequency) and temporal (time) components of neuronal responses. With receptive fields, computational models can be studied in greater detail. The computer-generated STRFs are hypothesized to be able to evaluate the realism of auditory processing models of the brain. To this effect, I have employed the use of two newly developed, neural network models of the brain as the preferred way to model early auditory processing stages and generate STRFs. One model is uses a linear transform scheme and the other model has a simple representation of memory. The connection factors, or weights, between different neurons determine how the neurons respond to auditory stimuli. These weights were fashioned through an unsupervised training algorithm using natural stimuli. Using this training procedure, the neural network extracted some of the statistical regularities of the natural world. Subsequently, complex, moving ripple stimuli were used in the model to obtain the receptive fields of the neuronal populations.

The resulting receptive fields illustrate the properties of the neural network. By analyzing the properties of the model, the validity of the neural network can be determined. To investigate this, other neural networks were created using the untrained and trained weights. The different weight matrices serve as different levels of realism that can easily be implemented in the same model. The comparison of the receptive fields from the two different models can be used to evaluate the validity of the neural network.

As scientists strive to develop ever more realistic, computational models of the brain, the detail and elegance of receptive fields will equip scientists to better evaluate the models against their biological counterparts. Computational models, with the use of receptive fields, can be extended to better purposes. Neural networks of the brain will help scientists understand ourselves and our capabilities and aid doctors in complex medical pathologies.

2 Background

2.1 Layout of the ear

The ear is the earliest stage of auditory processing in the brain. The ear is divided into three main
areas: the outer ear, the middle ear, and the inner ear. The general purpose of the outer and middle ear is to convey and amplify the mechanical vibrations in the air (sound) to the inner ear. Transduction, the process of converting mechanical signals into electrical potentials, takes place in the inner ear. The vibrations in the inner ear selectively cause hair cells along the basilar membrane in the cochlea to move. The movements of the hair cells allow electrical potentials to travel to the auditory nerve and become processed by the brain.

Hair cells are theorized to be frequency-selective. Not all hair cells respond to all aspects of a sound stimulus. Specific frequencies activate specific areas of the basilar membrane, information which is relayed to higher levels of auditory processing. Many neural networks of the auditory cortex take advantage of this phenomenon by representing sound stimuli as spectrograms, distributions of frequency vs. time. Computational neurons in models are connected to a particular frequency range in order to most realistically depict frequency-selectivity.

### 2.2 Oja’s rule

Unsupervised training methods allow neural network models to dynamically modify their own weighted connections between the units, analogous to the changes in synaptic plasticity between real neurons. The simplest form of unsupervised training is based on the Hebbian learning rule. Dr. Hebb hypothesized that if two neurons are simultaneously active, the connection between them should be strengthened. As a mathematical equation, Hebb’s rule can be represented as:

\[
\Delta_t w_{ij} = \epsilon x_i y_j
\]

where \( \Delta_t w_{ij} \) represents weight change between two units, \( \epsilon \) is the learning rate, and \( x_i \) and \( y_j \) are the activation values of the pre-synaptic and post-synaptic neurons, respectively. Hebb’s rule is a concise, albeit very limited, simplification of the synaptic plasticity of neurons.

Hebb’s rule is inherently unstable. One overwhelming problem with Hebb’s rule is that the weights approach infinity after repeated training sets. A modified version of Hebb’s rule, Oja’s rule, fixes this problem by normalizing the weights during each update by subtracting a portion of the existing weight away from the weight change. Oja’s rule can be shown as:

\[
\Delta_t w_{ij} = \epsilon (x_i y_j - y_j^2 w_{ij})
\]

where \( \Delta_t w_{ij} \) represents weight change between two units, \( w_{ij} \) is the current weight, \( \epsilon \) is the learning rate, and \( x_i \) and \( y_j \) are the activation values of the pre-synaptic and post-synaptic neurons, respectively. The learning rate is a key parameter that dictates how quickly the weights are updated. While a very small learning rate will cause the weights to change slowly over the training set, a sufficiently large learning rate will cause the weights to oscillate. In this paper, the learning rate was arbitrarily chosen in order to avoid these two issues.

Oja’s learning rule performs principal components analysis (PCA). PCA determines where the greatest variability of data lies and Oja’s rule extracts the first principal component of the data. PCA allows neural networks to meaningfully represent and interpret the input data.

### 2.3 Spectro-temporal receptive fields (STRFs)

STRFs represent the linear properties of primary auditory processing neurons in many types of animals. STRFs are generated by collecting a neuron’s responses to different moving ripple stimuli. Since these stimuli are approximate components of complex sounds, the STRFs characterize the neuron response to spectro-temporally rich sound stimuli.

The STRF depicts a unit’s impulse response characterizations at frequency-time points. The STRF plots describe the neuronal response in both spectral and temporal terms, and so are more useful than traditional methods of describing neurons such as tuning curves. Tuning curves only depict a neuron’s spectral properties. Although STRFs cannot fully capture the properties of an auditory neuron, they are useful descriptions of the cells. Since STRFs can approximate the linear properties of a neuron in both frequency and time, STRFs have been used to predict the outputs of neurons, further validating the utility of receptive fields in the auditory world.

### 3 Methods

#### 3.1 Neural network

Neurons in the primary auditory cortex have been generally characterized as linear. The neural network employed in this paper is a two-layer, linear transform model. The first layer of the network is the input layer, composed of 129 units. The input to the model are represented as spectrograms in order to account for frequency-selectivity in the basilar membrane. This method of input is the most realistic way of depicting auditory stimuli in the lower levels of processing. The input vector is a single timestep of the spectrogram of the input sound stimulus. Each vector represented a tempo-
ral window timestep of approximately 12 ms. The frequencies were evenly spaced by 43 Hz and ranged from 0 to 5512 Hz.

Each neuron in the second layer of the model is linked to 30 input units through the weighted connections. The neurons in the second layer have a 11 input unit window overlap to the two contiguous neurons. In total, the second layer consisted of 6 neurons. The second layer is simply the matrix multiplication of the first layer and the second, or:

\[ y_j = \sum_{i=1}^{\text{w}_{ij} x_i} \]

where \( y_j \) is a second layer neuron, \( \text{w}_{ij} \) is the weights between the inputs and second layer neuron, and \( x_i \) is the first layer neuron. The weights between the first and second layers of the model were originally set at zero-centered, normally distributed, random values. Though the weight values become modified by training using Oja’s Rule, the actual connections between the computational units will not change.Though 6 neurons in the second layer may seem small, this allows each neuron to observe a larger frequency range, making the outputs of each individual neuron more meaningful. Each neuron is in the second layer is permanently connected to a set of input neurons.

Animals do not have an unlimited ability to hear all frequencies. In this paper, the neural network has a range of hearing from 214 to 5512 Hz. Although animals may not have the same range of hearing as the neural network has, the neural network artificially simulates this limitation to more accurately match the real world. Therefore, the neural network has no weighted connections between the second layer and the first 5 input units. The upper range of hearing was determined by the sampling rate of the sound stimuli, as the Nyquist Theorem shows that the highest frequency for a sound waveform can accurately capture is half the sampling frequency.

\[ \lambda \]

\[ \text{t} \]

\[ \text{w}_{ij} \]

\[ \text{x}_i \]

\[ \text{y}_j \]

\[ \text{w}_{ij} \]

\[ \text{x}_i \]

\[ 1 \]

\[ t \]

\[ \text{w}_{ij} \]

\[ \text{x}_i \]

\[ 1 \]

\[ t \]

\[ \text{w}_{ij} \]

\[ \text{x}_i \]

\[ 1 \]

\[ t \]

\[ \text{w}_{ij} \]

\[ \text{x}_i \]

\[ 1 \]

\[ t \]

\[ \text{w}_{ij} \]

\[ \text{x}_i \]

\[ 1 \]

\[ t \]

\[ \text{w}_{ij} \]

\[ \text{x}_i \]

\[ 1 \]

\[ t \]

\[ \text{w}_{ij} \]

\[ \text{x}_i \]

\[ 1 \]

\[ t \]

\[ \text{w}_{ij} \]

\[ \text{x}_i \]

\[ 1 \]

\[ t \]

\[ \text{w}_{ij} \]

\[ \text{x}_i \]

\[ 1 \]

\[ t \]

\[ \text{w}_{ij} \]

\[ \text{x}_i \]

\[ 1 \]

\[ t \]

\[ \text{w}_{ij} \]

\[ \text{x}_i \]

\[ 1 \]

\[ t \]

\[ \text{w}_{ij} \]

\[ \text{x}_i \]

\[ 1 \]

\[ t \]

\[ \text{w}_{ij} \]

\[ \text{x}_i \]

\[ 1 \]

\[ t \]

\[ \text{w}_{ij} \]

\[ \text{x}_i \]

\[ 1 \]

\[ t \]

\[ \text{w}_{ij} \]

\[ \text{x}_i \]

\[ 1 \]

\[ t \]

\[ \text{w}_{ij} \]

\[ \text{x}_i \]

\[ 1 \]

\[ t \]

\[ \text{w}_{ij} \]

\[ \text{x}_i \]

\[ 1 \]

\[ t \]

\[ \text{w}_{ij} \]

\[ \text{x}_i \]

\[ 1 \]

\[ t \]

\[ \text{w}_{ij} \]

\[ \text{x}_i \]

\[ 1 \]

\[ t \]

\[ \text{w}_{ij} \]

\[ \text{x}_i \]

\[ 1 \]

\[ t \]

\[ \text{w}_{ij} \]

\[ \text{x}_i \]

\[ 1 \]

\[ t \]

\[ \text{w}_{ij} \]

\[ \text{x}_i \]

\[ 1 \]

\[ t \]

\[ \text{w}_{ij} \]

\[ \text{x}_i \]

\[ 1 \]

\[ t \]

\[ \text{w}_{ij} \]

\[ \text{x}_i \]

\[ 1 \]

\[ t \]

\[ \text{w}_{ij} \]

\[ \text{x}_i \]

\[ 1 \]

\[ t \]

\[ \text{w}_{ij} \]

\[ \text{x}_i \]

\[ 1 \]

\[ t \]

\[ \text{w}_{ij} \]

\[ \text{x}_i \]

\[ 1 \]

\[ t \]

\[ \text{w}_{ij} \]

\[ \text{x}_i \]

\[ 1 \]

\[ t \]

\[ \text{w}_{ij} \]

\[ \text{x}_i \]

\[ 1 \]

\[ t \]

\[ \text{w}_{ij} \]

\[ \text{x}_i \]

\[ 1 \]

\[ t \]

\[ \text{w}_{ij} \]

\[ \text{x}_i \]

\[ 1 \]

\[ t \]

\[ \text{w}_{ij} \]

\[ \text{x}_i \]

\[ 1 \]

\[ t \]

\[ \text{w}_{ij} \]

\[ \text{x}_i \]

\[ 1 \]

\[ t \]

\[ \text{w}_{ij} \]

\[ \text{x}_i \]

\[ 1 \]

\[ t \]

\[ \text{w}_{ij} \]

\[ \text{x}_i \]

\[ 1 \]

\[ t \]

\[ \text{w}_{ij} \]

\[ \text{x}_i \]

\[ 1 \]

\[ t \]

\[ \text{w}_{ij} \]

\[ \text{x}_i \]

\[ 1 \]

\[ t \]

\[ \text{w}_{ij} \]

\[ \text{x}_i \]

\[ 1 \]

\[ t \]

\[ \text{w}_{ij} \]

\[ \text{x}_i \]

\[ 1 \]

\[ t \]

\[ \text{w}_{ij} \]

\[ \text{x}_i \]

\[ 1 \]

\[ t \]

\[ \text{w}_{ij} \]

\[ \text{x}_i \]

\[ 1 \]

\[ t \]

\[ \text{w}_{ij} \]

\[ \text{x}_i \]

\[ 1 \]

\[ t \]

\[ \text{w}_{ij} \]

\[ \text{x}_i \]

\[ 1 \]

\[ t \]

\[ \text{w}_{ij} \]

\[ \text{x}_i \]

\[ 1 \]

\[ t \]

\[ \text{w}_{ij} \]

\[ \text{x}_i \]

\[ 1 \]

\[ t \]

\[ \text{w}_{ij} \]

\[ \text{x}_i \]

\[ 1 \]

\[ t \]

\[ \text{w}_{ij} \]

\[ \text{x}_i \]

\[ 1 \]
ordinate in response to the various environmental sounds. Therefore, the additive inverse of the weights of those specific neurons were used in order for the neurons to best correspond to the natural auditory scenes.

3.4 Constructing stimuli

3.4.1 Moving ripples

The moving ripple stimuli are complex, broadband noises that are used to determine the specrotetimal receptive fields (STRFs) of neuronal populations. These stimuli analogous to the Gabor patches in the visual domain, both of which have been tested in many previous studies. The moving ripples were made within the same range of frequencies as the natural stimuli used in the training in order for the units in the neural network to respond to their best frequency (BF). The spectral envelope of the noise was then modulated linearly. The ripple equation, intensity at specific time-frequency points, is given as:

\[ S(t, x) = 1 + \Delta A \times \sin[2\pi(\omega t + \Omega x) + \Phi] \] (5)

where \( S(t, x) \) is intensity, \( t \) is time, \( x = \log_2(F/F_0) \) where \( x \) is the logarithmic frequency axis, \( F_0 \) is the baseline frequency, \( F \) is the frequency, \( \Delta A \) is modulation depth, \( \omega \) is the ripple velocity (Hz), \( \Omega \) is the ripple frequency (cycles/octave), and \( \Phi \) is the phase shift (radians). The stimuli were generated in Matlab using a program designed by Powen Ru*. These ripple stimuli were varied across two parameters separately, the ripple velocity (Hz) and the ripple frequency (cycles/octave). The ripple velocity was varied from -40 to 40 Hz in steps of 4 Hz and the ripple frequency was varied from -4.6 to 3.4 in steps of 0.4 cycles/octave. The raster outputs of the units to the different moving ripples were computed. The transfer function (TF) is a broad characterization of a unit’s responses to the ripple stimuli and is defined by:

\[ TF(\omega, \Omega) = M(\omega, \Omega) \times exp[i \times \Phi(\omega, \Omega)] \] (6)

where \( M(\omega, \Omega) \) is the response magnitude, \( \Phi(\omega, \Omega) \) is the response phase, and \( i = \sqrt{-1} \). In order to construct the TF, the magnitude and phase of the raster responses were calculated by performing a Fourier transform. The second half of the Fourier transform was discarded because it provides redundant information. The magnitude was the maximum value of the transform, and the phase was extracted from the unwrapped angle at that point.

The transfer function is assumed to have conjugate symmetry. A two-dimensional inverse Fourier transform function was performed on the transfer function in order to generate the desired STRF.

3.4.2 Tuning curve tones

Tuning curves have been used extensively in both biological and computational applications. Tuning curves allow scientists to quantitatively analyze the frequencies at which a specific auditory neuron responds best to. The firing rates of the neurons are collected in response to pure tones varied across the frequency domain. The neurons respond with the greatest intensity to tones that match their BF, and with decreasing intensity to tones away from their BF. The plots of these rates against the frequency of tone generate the tuning curves. In this paper, the tones that were used to construct the tuning curves were generated in Matlab. These tones were 1 second long, and sampled at identical settings to the environmental stimuli, and were subsequently converted to spectrograms to become the input of the neural network. The frequency of the tones were varied from 10 to 5490 Hz in steps of 40 Hz. The responses of the computational neurons to these tones were collected. These responses were plotted in an intensity v. time plot, and the peak of the plotted curve denotes the BF of the neuron.

4 Results and Discussion

4.1 Receptive fields

The receptive fields were constructed after probing the neural network with the moving ripple stimuli. The moving ripple stimuli were varied across the rippled velocity (Hz) and ripple frequency (cycles/octave) parameters. The ripple velocity was varied from -40 to 40 Hz in increments of 4 Hz and the ripple frequency was varied from -4.6 to 3.4 cycles/octaves in increments of 0.4 cycles/octave. In total, * ripple stimuli were used to obtain the receptive fields. The other parameters were held constant for all ripple stimuli, shown on table *.

The receptive fields for the six different computational neurons are plotted in figure *. The abscissa represents time after stimulus onset and the ordinate represents frequency. The bright* area of the graph shows where neuron responds most intensely to. The dark* area shows where neuron does not respond to, or responds very weakly to. The width of the receptive field shows how long the neuron responds to complex stimuli and the length of the receptive field shows the frequency range at which the neuron responds to.

4.1.1 Original model

The STRFs were generated with the neural network without any memory. In the spectral domain of the receptive fields, the graphs show that the neurons
are responding to distinct frequency ranges. As the neurons are connected to higher frequency input units, the receptive fields show that the neuron also responds to higher frequencies. This result agrees with the hypothesized outcome and is graphed in figure *. The temporal component of the neurons is constant for all units. Since the neural network is linear, it does not respond to the stimuli over time and so the computational neurons all have the same temporal properties. The temporal domain of the graph is very narrow because the neurons are only responding to one timestep at a time. Additionally, the bright area of intense response is near zero on the time axis. This means the neurons respond immediately after the onset of the stimulus.

The best frequency (BF) can be obtained from the STRFs. The spectral component of the maximum value of the STRF represents the frequency at which a neuron responds best to. The BF for all neurons are graphed in figure *.

4.1.2 Tuning curves

After collecting the responses of the neurons to the pure tones, the tuning curves were obtained. The graphs of the tuning curves of all the computational neurons are shown on figure *, in intensity v. frequency. Although biological neurons can only respond at one fixed strength (all-or-none principle), the intensity of the response can be quantized as the firing rate, or how many times a neuron fires per time unit. The intensity of the response in the neural network is analogous to the firing rate in biological neurons. The maximum of the tuning curve is the best frequency (BF) of the neuron; the neuron responds most strongly to frequencies near the BF. The tuning curves, similar to the receptive fields, show that the neurons respond to specific frequency ranges. The BF’s from the STRFs is closely correlated (r=???) with the BF’s from the tuning curves. The tuning curves, though, do not give any indication of how the computational neurons are responding over time.

4.1.3 Model with memory

The receptive fields were also generated from the neural network with a simplistic version of memory. The spectral component of the STRFs from the model with memory is similar to the model without memory. The neurons are still connected to the same frequency ranges, so these STRFs show that the neurons are still responding to the same frequency ranges. Now, the STRFs show that the model with memory is responding over time, evidenced by the repeating areas of high intensity in the images. The subsequent areas of activation in the STRFs are less intense than the original area because of the exponential decay factors that scale the time-delayed inputs.

4.2 Accuracy of models via STRFs

Receptive fields have been used in this project to establish the linear properties of neural networks. Another goal of the project was to help scientists determine the accuracy of neural networks. To this effect, the neural network was modified with different levels of realism by changing the connective pathways between the input and output layer and the values of those connections. The original neural network (NN1) employed structured connections between the input and output layers with weights trained using Oja’s Rule. A more unrealistic neural network (NN2) used the same structure, but with untrained weights. The most unrealistic network (NN3) used random connections between the input and the output layers with untrained weights. The receptive fields from each of these networks is shown in figure *. The STRFs show not only visually depict the unrealistic nature of the networks but can also guide scientists in determining the nature of the problems.

The STRFs, though, show nothing new compared to the tuning curves. The tuning curves from each of the neural networks is shown in figure *. The tuning curves show as well how the varying models are unrealistic. But if the temporal-coded model is used, STRFs have the advantage over the tuning curves. For instance, if the neural network with memory is coded with responding to time-delayed inputs with increasing strength, then the STRFs would be able to show the inaccuracies while the tuning curves show no difference. In figure *, the areas of high intensity occur near the end of the receptive field, showing that the time-delayed inputs are more strongly responded to than the current input. On the other hand, the tuning curves look nearly identical in both cases. The STRFs are able to demonstrate the accuracy of neural networks in both the spectral and temporal domains.

4.3 Limitations

In this paper, environmental stimuli were used to modify the weights between the computational neurons. The environmental stimuli, though, are not perfect. For example, the sounds from the Internet and CDs were recorded using different equipment. The quality of the sound varied between the different sources. In addition, the content of the sound varied. Some sounds included large segments of background noise in between bird calls. Some sounds had more background noise than oth-
ers had. Some of the differences between the sounds were minimized through resampling and other modifications in order for the input to the neural model to be fairly consistent.

This neural network was constructed to evaluate STRFs in computational models, and so it is only a basic model. The memory coding for the model adds another dimension to the model, but it is not the most accurate method of simulating time-delayed inputs. The neural network also does not model lateral inhibition, the process through which neurons can suppress the activity of other neurons. Lateral inhibition is derived from neurotransmitters such as GABA that inhibit the post-synaptic neuron.

The use of the STRFs has been realistic in this project, with identical use of rippled stimuli to generate the receptive fields. The major differences are the parameters at which the moving ripples were varied across. But this limitation does not pose much of an importance because the STRFs, even across different ripple parameters, are stable and maintain a similar structure.

5 Conclusion

Receptive fields have been extensively analyzed in both animals and neural networks in the visual domain. In the auditory domain, spectro-temporal receptive fields (STRFs) describe both the spectral and temporal aspects of a neuron. The STRFs have been used in many types of animals, but have not been explored in a computational model. This project describes the construction of a neural network of basic auditory processing and the subsequent testing using STRFs. STRFs can be utilized to analyze the properties of computational models of auditory processing in a visual manner. The receptive fields describe how a neuron would generally respond to both the spectral and temporal properties of sound stimuli. This characterization of the neural network can then assist scientists in determining the accuracy of their models, without reading lines of code or examining multiple outputs from each individual neuron. The STRFs from the auditory network would be able to show if a model is responding to the correctly to time-delayed inputs and frequency ranges in just a few graphs. Tuning curves are also able to depict the realism of models, but not with both frequency and time information. In this way, the simpler tuning curves do not have the same ability of receptive fields.

Neural networks and receptive fields also have potential to help doctors in the medical field. The underlying purpose of neural networks is not just to help scientists understand ourselves better but also put that knowledge to use in helping us. The most prevalent auditory disorder is hearing loss, and neural networks may be able to aid doctors determine the cause and spread of hearing loss. This model can approximate the auditory processing, and so can act as a pseudo-subject in an experimental study for hearing loss. Scientists would be able to retrain the weights of the model using Oja’s Rule by using stimuli such as loud music and other destructive noises, and then generate the STRFs with the new weights. If the weighted connection becomes greater than an arbitrary threshold, then the weight is set to zero, modeling the loss to hear certain frequencies. By examining the receptive fields, scientists can tell which parts of the ear are affected by the sounds and how both the spectral and temporal properties changed because of the hearing loss. This neural network would even be able to track the spread of hearing loss, and help determine whether the loss to hear certain frequencies can further contribute to hearing loss. Scientists would be exploring the results to this kind of study with a neural network without actually exposing humans to these potentially dangerous sounds.

Spectro-temporal receptive fields are powerful tools that have now been studied in computational models for the first time. This project demonstrates the flexibility and utility of STRFs in visually describing the properties of neurons in an auditory model and determining the accuracy of the model. This analysis suggests that hearing loss can further be studied without affecting humans in an experimental study.
Abstract

In order to better understand chemistry, chemists create 3D models of molecules. In a large introductory chemistry class, physical models are not viable because the supplies needed to give each student the opportunity to create even simple molecules are too costly. Also, software available online can also be costly. The goal of my research project is to create a program that will allow users to generate accurate 3D models of simple molecules (i.e. not macromolecules). Therefore, my project could help students in introductory chemistry courses better understand the geometry of different molecules.

1 Introduction

The goal of my project is to create a free program that will allow users, such as introductory chemistry students, to easily and intuitively create models of molecules. While running the program, users will be able to create atoms and bonds where they want them. Once they have everything created, with just a click of the mouse, my program will position the atoms correctly. The user will then be able to export the model so that it can be easily imported at a later time. All of the models will also be rotatable and zoomable at all times, allowing for a better understanding of the molecular geometry.

The first step of my project was creating the graphics for my program. The second step was creating an easy and intuitive user interface. Through mainly inputs from the mouse (and some keyboard input) users can create atoms and bonds, select and delete atoms and bonds, import and export models, draw single, double and triple bonds, choose which element they want to draw, and position the atoms where they want. (For atomic radius values, I used empirical values derived by J.C. Slater and published in The Journal of Chemical Physics) With these features, users will be able to easily create the models.

2 Background

A lot of research has been done on modeling molecules and the techniques to do so now have become quite advanced. Pharmaceutical companies have invested thousands of dollars in programs to predict orientations of new complex molecules. Large databases have been created, storing the orientations of thousands of molecules. Chemists have modeled everything from RNA to inorganic crystals. These programs have become increasingly more accurate over the years. However, the cost of these programs has also increased, and now very few people have access to them. For a beginning chemistry student there are not many options to play around with molecules and learn their different geometries. Buying physical modeling sets for everyone in an introductory chemistry course is costly. Also, software on the Internet can also be costly, sometimes a couple hundred dollars. Therefore, even today, a free program to model simple molecules would be helpful.

3 Development

The first step to my project was creating the graphics for my models. The graphics for my project are a much simplified version of real molecules: spheres represent atoms and cylinders represent bonds. Nevertheless, the results are sufficient.

The second step of my project was creating an easy and intuitive user interface. Through mainly inputs from the mouse (and some keyboard input) users can create atoms and bonds, select and delete atoms and bonds, import and export models, draw single, double and triple bonds, choose which element they want to draw, and position the atoms where they want. (For atomic radius values, I used empirical values derived by J.C. Slater and published in The Journal of Chemical Physics) With these features, users will be able to easily create the models.

The third step of my project is creating the algorithm that will correctly orient the atoms in the molecule. Since there are many variables accounting for the actual orientation of the atoms and because I only have a limited amount of time, I will need to limit the number of accounted variables.
With my set number of accounted variables, I will use an A.I. algorithm (hill-climbing) to correctly orient the atoms in the molecule. The first variable that I will account for will be the polarity of the bond. Once I get that working, I will continue to add more variables, creating a better model.

Dynamic testing will not work for my project because an atom can not bond with any random atom. Therefore, I will mostly use specific structural and functional testing and path and branch testing.

4 Testing

Testing the graphic part of my program was simple; all I had to do was run it and see if my program created the intended model.

Testing the user interface part of my program was a little more complex. I had to think of everything that a user could do and account for the inputs. However, my program runs as intended.

Now that I have all of the graphics and user interface done for my project, I will now be able to start working on my algorithm to correctly orient the atoms in the molecule. I will first start by only taking one variable into account: the polarity of the bond. Once I get that working, I will then add another. Once that one is working, I will start on another. Through this process, my program will start to produce accurate representations of different molecules.

In order to see if my program is constructing accurate models, I will compare my models with the actual/accepted models. By doing this, I will also be able to do error analyses. In order to thoroughly test my program, I will test many different molecules and compare my results with the actual orientations.

5 Results

The purpose of my project is to create a free program that will allow users to easily and intuitively create models of molecules. Then, after the user finished creating the molecule, the program will correctly position the atoms. My project currently allows users to create molecules. A user can also rotate the model and zoom it in and out. These functions help users get a better picture of the molecule. However, I still need to work on the algorithm to correctly position the atoms. This will be the focus of the third quarter.

References


Advanced Automobile Recognition Through the Use of Image Processing Techniques

Andrew W. Stebbins
TJHSST
Alexandria, Virginia

Abstract

Many law enforcement agencies have recently shown interest in automated automobile recognition and tracking technologies such as license plate reading or GPS tracking. However, some criminals may drive vehicles that have false license plates or are not equipped with GPS tracking devices, making the pursuit of such vehicles difficult. This project aims to create a computer vision system capable of taking real-time input from a static camera and identifying passing cars by make and model in order to assist law enforcement agencies in the tracking of suspect or stolen vehicles. Vehicle identification is accomplished using a combination of old and new algorithmic image processing techniques.

Keywords: Canny edge detector, Hough transform, artificial neural network

1 Introduction

Many law enforcement agencies, especially ones in large metropolitan areas, are faced with difficulties when tasked with finding one specific car in a city of thousands. For example, a police officer may receive breaking news of a robbery underway, arrive late at the scene, and then have to chase the getaway car provided only with a witness’s visual description of the vehicle. Existing car-tracking technologies such as License Plate Recognition (LPR) would fail in this case, as the officer does not know the license plate number of the vehicle driven by the suspects who he is attempting to apprehend. It is in cases like these that an automatic visual automobile recognition system may prove useful. Thus, this project is primarily aimed towards assisting law enforcement agencies with chasing down criminals or recovering stolen cars.

2 Background

Several computer systems currently exist for the tracking of military and civilian automobiles via License Plate Recognition (LPR) or GPS technology. Such systems are in use by law enforcement entities such as US Customs and Border Protection[1] and UK police[2], and have proved very effective in catching criminals. However, these systems fail when an automobile has fake or no plates, and no GPS tracking device, and is able to avoid recognition. The new system outlined in this paper, on the other hand, is able to alert law enforcement officers of the presence of any specific type of vehicle regardless of whether or not it is equipped with GPS or the proper license plates, assisting in situations such as when an all-points bulletin is put out for a certain vehicle based only on a visual description. In addition, some systems already exist[3] that can automatically recognize military vehicles such as tanks by their color, size, geometric description. However, in the course of my preliminary research I found no complete existing systems capable of the automated, advanced (ie make and model) recognition of everyday civilian vehicles such as cars, small trucks, semis, etc. The inner workings of my system will be similar to that of the existing systems for the automated detection of military vehicles, in that it will define a certain set of characteristics for comparison, extract those characteristics from the image of an unknown vehicle, and search amongst a list of characteristics known to belong to certain vehicles for a possible match. The primary difference between these types of systems and my own is that mine will be much more precise in terms of characteristics such as size and shape, and select possible matches from a much more diverse database.

3 Region of Interest Identification

This part of my program has not yet been implemented, but will utilize movement tracking to identify areas of an image that are in motion in order to differentiate a moving car from static background clutter. The purpose of identifying the region of interest in an image that contains a car is to remove background clutter and make it easier for the object classification algorithm to correctly recog-
nize key features and important points or vertices. Movement tracking is a commonly used and reasonably accurate method for obtaining such a region of interest[5].

4 Object Classification

This part of my program has not yet been fully implemented, but will focus on recognizing and classifying key distinctive features from the image of an automobile. Such features may include color, size, shape, and the location of key points or vertices. Currently, I have implemented edge, line, and vertex detection, underlying building blocks for any advanced feature recognition algorithm. I have looked into neural networks as an intuitive, learning-based method of automobile classification; my conclusion was that they would not be appropriate for an object categorization task of such complexity. The set of training example images would need to be immense for a neural network-based automobile recognition program to attain any acceptable level of accuracy.

5 Results and Discussion

Unfortunately, my project does not yet exist in the form of a single program able to be tested on its ability to detect and recognize automobiles in video segments. The various components of my program currently perform as expected, accurately detecting lines, recognizing handwritten characters, and displaying video input. The frame rate of my streaming input viewer is somewhat low, a problem which will have to be addressed before accurate motion detection can be attempted. I hope to be able to test an assembled version of my final program on short video segments of cars in motion by the end of the third quarter.

References


[2] “Number plate scanners hit crime.” BBC
   New. 27 October 2007
   http://news.bbc.co.uk/2/hi/uk_news/england/berkshire/4553469.stm

   http://people.csail.mit.edu/dalleyg/presentations/MSThesisPresentation.ppt
   (October 27, 2007)

   http://www.eecs.berkeley.edu/Research/Projects/CS/vision/shape/nips04.pdf (October 27, 2007)

   http://www.cse.ucsd.edu/classes/fa05/cse252c/dlagnekov_belongie.pdf (January 2, 2007)
Cayley graphs formed by conjugate generating sets of $S_n$

Jacob Steinhardt

Abstract

We investigate subsets of the symmetric group with structure similar to that of a graph. The “trees” of these subsets correspond to minimal conjugate generating sets of the symmetric group. There are two main theorems in this paper. The first is a characterization of minimal conjugate generating sets of $S_n$. The second is a generalization of a result due to Feng characterizing the automorphism groups of the Cayley graphs formed by certain generating sets composed of cycles. We compute the full automorphism groups subject to a weak condition and conjecture that the characterization still holds without the condition. We also present some computational results in relation to Hamiltonicity of Cayley graphs, including those elements not fixed by the permutation. Given a set $S \subset G$, the Cayley graph $\Gamma = Cay(G, S)$ is defined as follows: each vertex is an element of $G$, and two vertices $g, h \in V(\Gamma)$ are adjacent if $gh^{-1} \in S$ or $hg^{-1} \in S$.

Key words: Cayley graph, automorphism, transposition, cycle, conjugate, Hamiltonian cycle.

1 Terminology

In this paper, we will let $N$ denote the set $\{1, 2, \ldots, n\}$. $S_n$ will denote the symmetric group acting on $n$ elements with canonical action on $N$. $A_n$ will denote the alternating group acting on $N$. We will use the notation $(a_1 a_2 \ldots a_k)(b_1 b_2 \ldots b_k)\ldots$ to express a permutation as a product of disjoint cycles. By the support of a permutation we will mean those elements not fixed by the permutation. Given permutations $\sigma, \tau, \sigma \tau$ denotes $\tau \circ \sigma$.

Given a multiset $A = (a_1, \ldots, a_k)$ with $a_i \geq 2$, we define its extended conjugacy class in $S_n$ to be the set of all permutations such that, when decomposed into disjoint cycles, contain cycles of lengths $a_1, \ldots, a_k$, and no others. We denote it by $C(A)$.

Given a set $S \subset N$, define the subsymmetric group of $S$ as the set of all permutations in $S_n$ that fix all elements outside of $S$. Define the subalternating group of $S$ as the set of all even permutations in $S_n$ (i.e., the permutations in $A_n$) that fix all elements outside of $S$. A semisymmetric group of $S$ is defined as a subgroup of $S_n$ that stabilizes $S$ whose restriction to $S$ forms a symmetric group acting on $S$. A semialternating group of $S$ is defined similarly.

Given a graph $\Gamma$, we let $V(\Gamma)$ and $E(\Gamma)$ denote the vertices and edges of $\Gamma$, respectively. An Eulerian path is a walk in $\Gamma$ that traverses each edge exactly once. It is called an Eulerian cycle if the first and last vertices in the walk are the same. Given a group $G$ and a set $S \subset G$, the Cayley graph $\Gamma = Cay(G, S)$ is defined as follows: each vertex is an element of $G$, and two vertices $g, h \in V(\Gamma)$ are adjacent if $gh^{-1} \in S$ or $hg^{-1} \in S$.

2 Motivation and Overview

Cayley graphs are of general interest in the field of Algebraic Graph Theory and also have certain properties desirable in practical applications. We present here a brief survey of some of the broader results and conjectures surrounding Cayley graphs. Godsil and Royle [8] provide a useful overview of work on graphs with transitive permutation groups in general, which we partially reproduce here. First, all Cayley graphs are vertex-transitive since the mapping $\phi_g(x) = xg$ is an automorphism for all $g \in G$. As such, there is always a representation of $G$ in $\text{Aut}(Cay(G, S))$, denoted $R(G)$. $R(G)$ acts not only transitively but regularly on the vertices of $Cay(G, S)$. Sabidussi has shown that the converse of this is true, namely that if a Cayley graph of $G$ if and only if $\text{Aut}(\Gamma)$ contains a subgroup isomorphic to $G$ that acts regularly on $V(\Gamma)$ [20].

Minimally generated Cayley graphs have provably maximal vertex connectivity [8], which points to uses in practical applications. Specifically, Cayley graphs have been used to create networks with small diameter and valency and high connectivity for uses in parallel processing, and Schreier coset graphs, a generalization of Cayley graphs, have been used to solve certain routing problems [3]. See also [2] and [21].

One tempting conjecture related to Cayley graphs is that every Cayley graph has a Hamiltonian cycle, first given by Strasser in 1959 [19]. For more information, see [4], [13], and [17]. While we offer some computational ideas in relation to this conjecture, the focus of this paper is on another difficult problem in Algebraic Graph Theory, that of characterizing the...
automorphism groups of Cayley graphs. We have a poor understanding of the automorphism groups of Cayley graphs, though there are some notable exceptions (see below). These groups are fundamental as the most natural algebraic structure to associate with an arbitrary highly symmetric graph.

Note that a graph can be defined as a collection of vertices and edges. Two vertices are adjacent if there exists an edge connecting them, and two vertices \( v_1 \) and \( v_2 \) are connected if there exists a sequence of adjacent vertices containing \( v_1 \) and \( v_2 \). On the other hand, consider the following definition: Given a collection of vertices \( V \) and a collection of edges \( E \), we can let each element of \( E \) act on \( V \) as a transposition swapping the two vertices on which \( E \) is incident. If we then let multiplication in \( E \) extend through the definitions of a group action, \( E \) generates a subgroup of the symmetric group acting on \( V \) (we denote this subgroup as \( < E > \)). Then we say that \( v_1 \) and \( v_2 \) are adjacent if \( (v_1, v_2) \in E \), and that \( v_1 \) and \( v_2 \) are connected if \( (v_1, v_2) \in < E > \). Additionally, connected components correspond to orbits of \( V \) under \( E \). A tree is a minimal generating set of \( S \) consisting only of transpositions (thus the fact that trees have \( n-1 \) vertices corresponds to the fact that it takes \( n-1 \) transpositions to generate \( S_n \)). It is easily verified that these definitions are equivalent.

The algebraic properties of the related Cayley graphs of trees in the above definition are well-understood. We know in particular that these graphs are Hamiltonian [12]. Furthermore, in 2003 Feng [5] generalized a result by Godsil [8] that fully characterizes the automorphism groups of these graphs.

### 3 C-trees

#### 3.1 Definitions

The above definition of a graph in terms of transpositions can be generalized. Given a collection of vertices (which, from now on, for convenience, will without loss of generality be \( N \)), and a set \( T \subset S_n \) in which all elements of \( T \) are conjugate (say with conjugacy class \( C \)), then we can define elementary notions in a \( C \)-graph as follows. \( v_1, v_2 \in N \) are adjacent if they have the same orbit under a single element of \( T \). They are semi-connected if they have the same orbit under \( T \), and connected if \( (v_1, v_2) \in < T > \) (it is then easy to verify that semi-connectivity and connectivity are equivalence relations). Connected components correspond to subsymmetric groups of \( < T > \). A tree is a minimal generating set of \( S_n \) with all elements lying in \( C \). It is natural to ask why we add the somewhat artificial-looking stipulation that all elements of \( T \) belong to the same conjugacy class.

The main reason is that this stipulation is inherent in the construction of a normal graph, where all edges are transpositions. Additionally, without this restriction we get the result that a tree, under our fairly intuitive definition, almost always has 2 edges since \( (1 2) \) and \( (2 3 4 \ldots n) \) generate \( S_n \).

In this paper we will characterize \( C \)-trees and study some of their properties, including a generalization of Feng’s result. However, we will still use the language of graphs for the sake of intuition. For approaches to extending the above intuitive generalization to a well-structured system, see the concluding section on open problems.

With \( C \)-trees defined, we introduce some more terminology associated to them. A set \( T \subset S_n \) is said to be semi-connected if \( N \) has a single orbit under \( T \) (i.e. all elements of \( N \) are semi-connected). We call it split if the intersection of the supports of any two elements of \( T \) has size at most one. Note that if \( T \) generates \( S_n \) then it must be connected.

Given a multiset \( A \) and integer \( n \), we define \( f(A, n) \) to be the infimum of \( |G| \) across all \( G \subset C(A) \) that generate \( S_n \). We aim to find \( f(A, n) \) for every \( A \) for sufficiently large \( n \). Let \( c(A) \) be defined as

\[
|A| \sum_{i=1}^{a_i} a_i - 1
\]

We aim to prove that there exists a function \( X_0(A) \) such that, for \( n \geq X_0(A) \), \( f(A, n) \) is equal to

\[
\left\lfloor \frac{n-1}{c(A)} \right\rfloor
\]

(1)

when \( c(A) \) is odd, and \( \infty \) otherwise. When \( c(A) \) is odd, then \( A \) defines the conjugacy class of an even permutation and so \( f(A, n) \) is obviously \( \infty \) (because it is impossible to generate any odd permutations). Note further that \( f(A, n) \) is necessarily at least the value given by (1), as \( c(A) \) counts the number of transpositions necessary to generate an element of \( C(A) \), and so if it was smaller then it would be possible to generate \( S_n \) with less than \( n-1 \) transpositions. Another way to see this is that no potential generating sets can be semi-connected, and thus cannot generate \( S_n \).

#### 3.2 Cycles

We study first the case of a single \( k \)-cycle, i.e. \(|A| = 1 \) and \( a_1 = k \). We will give explicit generators for \( S_{2k-1} \):
Proposition 3.1  The set \(\{1, 2, \ldots, k\}, (k, k+1, \ldots, 2k-1)\) generates \(S_{2k-1}\) when \(k\) is even.

Proof  We construct something similar to a semisymmetric group of \(\{1, 2, \ldots, k\}\), except with the elements lying in the positions \(\{k, k+1, \ldots, 2k-1\}\). From this we will construct a subsemisymmetric group of \(\{1, 2, \ldots, k\}\), which will finally allow us to construct the entire symmetric group \(S_{2k-1}\).

Lemma 3.2  We can place the elements \(\{1, 2, \ldots, k\}\) in any order in positions \(\{k, k+1, \ldots, 2k-1\}\) as long as we allow the other elements to move arbitrarily (even for odd \(k\)).

Proof  Let \(\sigma = (1, 2, \ldots, k)\) and \(\tau = (k, k+1, \ldots, 2k-1)\). If we allow the first \(k-1\) elements to be arbitrary, we can set the last \(k\) elements to any permutation \(\pi = (\pi_1, \pi_2, \ldots, \pi_k)\) of \(\{1, 2, \ldots, k\}\) as follows: Rotate \(\pi_k\) to the \(k\)th position using \(\sigma\), then apply \(\tau\) once. Now rotate \(\pi_{k-1}\) to the \(k\)th position (again with \(\sigma\)), and apply \(\tau\) again. Continue this process until we have put all \(k\) of the desired elements into place. For example, to make the last 4 elements (2, 4, 1, 3), we would apply \(\sigma^1 \tau \sigma^1 \tau \sigma^2\). Each successive set of applications of \(\sigma\) moves the next desired element into place (3, then 1, then 2).

Lemma 3.3  For \(k\) even, the \(k\)-cycles generate \(S_n\) for \(n \geq k\). Moreover, for \(k\) odd, the \(k\)-cycles generate \(A_n\).

Proof  Let \(U\) be the set of all \(k\)-cycles. \(U\) is closed under conjugation, so \(\langle U \rangle\) is normal in \(S_n\). Thus \(\langle U \rangle\) is either \(\langle e \rangle\), \(D_4\), \(A_n\), or \(S_n\), where by \(D_4\) we mean the normal subgroup of \(A_4\) isomorphic to the dihedral group of order 4. It can’t be \(\langle e \rangle\) because \(U\) is non-trivial, and it can’t be \(D_4\) because \(D_4\) contains no cycles. Thus it is \(A_n\) or \(S_n\). If \(k\) is odd, it must be \(A_n\) because \(U\) consists of only even permutations. If \(k\) is even, it must be \(S_n\) since \(U\) contains an odd permutation.

Lemma 3.4  We can generate the subsymmetric group on \(\{1, 2, \ldots, k\}\) when \(k\) is even.

Proof  Take some such permutation \(\pi\) generated in the manner of Lemma 3.2, and consider \(\pi \pi^{-1}\). This creates an arbitrary \(k\)-cycle among the first \(k\) elements while fixing the last \(k-1\) elements. Then by Lemma 3.3 we can generate the subsymmetric group on \(\{1, 2, \ldots, k\}\).

Now, to generate an arbitrary permutation \(\pi = (\pi_1, \ldots, \pi_{2k-1})\) in \(S_{2k-1}\), first use \(\tau\) to move \(\{\pi_{k+1}, \ldots, \pi_{2k-1}\}\) to the first \(k-1\) elements of the set. We can do this by moving each one to the \(k\)th position, then, since we can generate any permutation among the first \(k\) elements (recall that we can do this by Lemma 3.4), move it to an arbitrary place among the first \(k-1\) elements in which we haven’t already put anything with this process. Next apply a permutation that puts \(\{\pi_{k+1}, \ldots, \pi_{2k-1}\}\) in the proper order (though leaving them in the positions \(\{k, k+1, \ldots, 2k-1\}\)). We can then move them to their proper locations with \(\tau \pi^{-1}\). Now that the last \(k-1\) elements are in place, we can apply whatever permutation is necessary to put the first \(k\) elements in place. We can thus generate an arbitrary permutation and therefore \(S_{2k-1}\). This completes Proposition 3.1.

Proposition 3.5  \(f((k), n(k-1) + 1) = n\) for \(n \geq 2\) and \(k\) even.

Proof  This follows by induction on \(n\). Proposition 3.1 proves the base case of \(n = 2\). The inductive step is completed by the following easily verified lemma:

Lemma 3.6  The subsymmetric group on \(S\), together with the cycle \((a_1, \ldots, a_n)\), generates the subsymmetric group on \(S \cup \{a_1, \ldots, a_n\}\) provided that \(S \cap \{a_1, \ldots, a_n\} \neq \emptyset\) and \(S \notin \{a_1, \ldots, a_n\}\).

Corollary 3.7  \(f((k), n) = \left\lfloor \frac{n-1}{k-1} \right\rfloor\) for \(n \geq 2k - 1\).

Proof  Take the construction for when \(\frac{n-1}{k-1}\) is an integer (i.e. that given above in Proposition 3.5). Then, to extend the formula to arbitrary \(n\), add the \(k\)-cycle \((n-k+1, n-k+2, \ldots, n)\) and apply Lemma 3.6. This completes the claimed characterization of \(f(A, n)\) for cycles.

3.3 Products of Transpositions

Having proven our result for cycles, we would like to extend it to more complex permutations. We will start with the simplest of these, i.e. products of disjoint transpositions. We call a permutation basic if it is of this form, and denote \(B_k = (2, 2, \ldots, 2)\) (\(k\) twos).

Proposition 3.8  \(f(B_k, n) = \left\lfloor \frac{n-1}{k} \right\rfloor\) for \(n \geq k(2k + 1) + 1\) and \(k\) odd.

Proof  Our goal will be to write \(2k+1\) explicit generators for \(S_{k(2k+1)+1}\). Then we can easily proceed by induction as before. Such generators must form a semi-connected set. However, we would also like the set to be split so that only one cycle interacts at a time when multiplying permutations. To do this, we will find an Eulerian cycle of \(K_{2k+1}\), which will allow us to create a connected and split set.
First note that every vertex of $K_{2n+1}$ has even degree (in particular, degree $2n$), so that $K_{2n+1}$ has an Eulerian cycle. We now construct generators from this cycle. We start with an example, then give a general method. Consider this cycle. We start with an example, then give an Eulerian cycle. We now construct generators from a group (in particular, degree 2).

\[ g_1 = (1\ 2)(5\ 6)(12\ 13) \quad g_2 = (2\ 3)(9\ 10)(19\ 20) \]
\[ g_3 = (6\ 7)(16\ 17)(20\ 21) \quad g_4 = (3\ 4)(13\ 14)(17\ 18) \]
\[ g_5 = (10\ 11)(14\ 15)(21\ 22) \quad g_6 = (7\ 8)(11\ 12)(18\ 19) \quad g_7 = (4\ 5)(8\ 9)(15\ 16) \]

Note that if we follow the path of permutations containing $(1,2)$, $(2,3)$, $(3,4), \ldots, (21,22)$, then we get $g_1, g_2, g_4, g_7, \ldots, g_3, g_5$, i.e. exactly the constructed Eulerian cycle (with the exception of the final vertex). This is the general method in which we will construct our generators. Specifically, we place the transposition $(i\ i+1)$ in the generator corresponding to the $i$th vertex visited in the cycle. Note that the properties of an Eulerian cycle guarantee that these generators will be semi-connected and split. The semi-connected part is obvious, whereas the split part is a consequence of the fact that every edge is traversed exactly once, which corresponds to the fact that each pair of generators move at most one common element. We now show this construction generates $S_{h(2k+1)+1}$.

**Theorem 3.9** If $T \subset C(B_k)$ is semi-connected and split, then $\langle T \rangle = S_n$ or $A_n$, depending on whether $k$ is even or odd.

**Proof** We call two generators adjacent if they move a common element. Consider two adjacent generators, $g_i$ and $g_j$, and consider $g_i g_j g_i g_j$. All transpositions are applied twice in this case and therefore cancel, except for the two transpositions that act on the same element, which multiply to a three cycle. So, in the above example, $g_5 g_7 g_5 g_7 = (14\ 15)(15\ 16)(14\ 15)(15\ 16) = (14\ 16\ 15)^2 = (14\ 15\ 16)$. In this manner, we generate all 3-cycles of the form $(i\ i+1\ i+2)$. We wish to show that these generate $A_n$. From this, we would be done, since any odd permutation then allows us to generate $S_n$. In fact, it is convenient for later purposes to prove a slightly stronger result:

**Lemma 3.10** When $n$ is odd, the subalternating group on $S$, together with the cycle $(a_1, \ldots, a_n)$, generates the subalternating group on $S \cup \{a_1, \ldots, a_n\}$ provided that $S \cap \{a_1, \ldots, a_n\} \neq \emptyset$ and that $|S \cap \{a_1, \ldots, a_n\}| \leq |S| - 2$. When $n$ is even, it generates the entire subsymmetric group.

**Proof** Like Lemma 3.6, the proof is easy enough to omit. The only important detail to note is that we get $\frac{|S \cap \{a_1, \ldots, a_n\}|}{2}$ distinct permutations, which must be the alternating group when $n$ is odd or must generate the symmetric group by Lagrange’s theorem when $n$ is even.

In particular, a 3-cycle looks like $A_3$, so the given 3-cycles indeed generate the alternating group (they are semi-connected since $T$ was semi-connected), and we are done with Theorem 3.9.

Our proof of the remainder of Proposition 3.8 (i.e. the induction and extension to cases when $c(A)$ does not divide the $n−1$) follows in exactly the same manner as that of Proposition 3.1, and so we omit it, instead referring the reader to Proposition 3.5 and Corollary 3.7. The only necessary modification is that we must deal with each of the cycles in the added permutation one at a time in our inductive step.

### 3.4 The General Case

We would next like a general criterion for connectedness. We present it here:   

**Definition** A set $T \subset C(A)$ is called balanced if it is possible to divide the set of orbits of elements of $T$ into disjoint sets $S_1, S_2, \ldots, S_{|A|}$ such that all orbits in $S_i$ have the same size and each element of $S_i$ overlaps with at least one other element of $S_i$. We will denote the size of the orbits in $S_i$ by $|a_i|$.

**Theorem 3.11** All semi-connected, split, balanced sets in the extended conjugacy class of an odd permutation generate $S_n$.

**Proof** We proceed by induction on two quantities: first $|A|$, then the number of occurrences of 2 in $A$. Note that we have already proven the base cases of this induction in Corollary 3.7 and Proposition 3.8.

We call two permutations $i$-adjacent if they both have orbits in $S_i$. Pick $i$ such that $a_i$ is maximal, and consider any two $i$-adjacent permutations, $\sigma$ and $\tau$, with orbits $\sigma S_i$ and $\tau S_i$. By the same argument as Proposition 3.1, these generate the semialternating group on the elements moved by the two identified cycles (moving $2a_i − 1$ elements in total). Thus in
particular, by Chebyshev’s Theorem [18], there exists a prime strictly between \(a_i\) and \(2a_i\), and the semialternating group contains a cycle of this length, call it \(p\).

Consider each cycle of this length in our semialternating group, and apply it \(\text{lcm}_{a \in A} a\) times. Since \(p\) is prime and \(a_j < p\) for each \(j\), we end up with a \(p\)-cycle, which we can then apply some number of times to get back to our original \(p\)-cycle. Note, however, that all other elements that were moved contained only cycles of length \(a_j\) for some \(j\), and so were all cancelled out by the above repeated application. Thus we are left only with the actual \(p\)-cycle. Repeating this for all such \(p\)-cycles in the semialternating group gives us all actual \(p\)-cycles, i.e. those living in the associated subalternating group. Thus, by the same arguments as in Lemma 3.3, they generate the entire subalternating group. If \(a_i\) is odd, then \(\sigma_i\) and \(\tau\) both live in this subalternating group, and so we can take \(\sigma_i\tau^{-1}\) and \(\tau\sigma_i^{-1}\). Taking \(\sigma_i\tau^{-1}\) for all \(\sigma \in T\) (we can do this since \(T\) is balanced) gives us a semi-connected, split, balanced set with strictly less orbits in each permutation, so that we can apply the inductive step to generate the subsymmetric group on the elements moved by these new permutations. Then, by adding \(\tau\sigma_i\) for each \(\sigma \in T\), by Lemma 3.5 we can generate the entire symmetric group, and we are done.

On the other hand, if \(a_i\) is even, then we can only cancel \(\sigma_i\) down to a transposition. However, this gives us an extended conjugacy class with the same number of orbits, but with strictly more occurrences of 2 in \(A\) than before. Thus we can apply the inductive step in the same manner as above, and are once again done. Note that in both cases we attained elements in the new extended conjugacy class by multiplying elements in the old extended conjugacy class by an even permutation. This shows that the new extended conjugacy class does indeed correspond to an odd permutation.

We are now ready to prove our major contention:

**Theorem 3.12** Let \(A\) be a multiset of size \(k\). Then there exists some \(X_0(A)\) such that \(f(A, n) = \lceil \frac{n-1}{\Phi_k} \rceil\) when \(n \geq X_0\). Furthermore, \(X_0((k)) \leq 2k - 1\), \(X_0(\langle k \rangle) \leq \frac{(2k+1)^2}{2} + 1\), and \(X_0(A) \leq c(A)\Phi_2(A)\Phi_k(2|A|) + 1\), where \(\Phi_k\) denotes the \(k\)th cyclotomic polynomial.

**Proof** Note that the first two bounds have already been proven. For the final case, we again use Eulerian cycles, this time with the goal of creating a semi-connected, split, and balanced set. In particular, we find a prime congruent to 1 mod 2. \(k\). We know that such a prime exists that is less than \(\Phi_{2k}(2k)\) (proof sketch in appendix).

Take such a prime, \(p\), fitting the properties described above. If \(\frac{p-1}{2} = kn\), then we will work in the extended conjugacy class that is equivalent to \(n\) copies of \(A\), then use this to move down to \(A\) itself. We construct an Eulerian cycle for \(K_p\) as follows. The edges (mod \(p\)) will be

\[1, 2, \ldots, p, 3p, 4p, \ldots, 2p, 3, \ldots, 3p, \ldots, \frac{p-1}{2}, p-1, \ldots, \frac{p(p-1)}{2}\]

So, for example, if \(p = 7\) then we have (in the case of 2-cycles) the associated generators

\[(1\ 2)(11\ 12)(19\ 20)\ (2\ 3)(8\ 9)(17\ 18)\]
\[(3\ 4)(12\ 13)(15\ 16)\ (4\ 5)(9\ 10)(20\ 21)\]
\[(5\ 6)(13\ 14)(18\ 19)\ (6\ 7)(10\ 11)(16\ 17)\ (7\ 8)(14\ 15)(21\ 22)\]

We can extend this past 2-cycles (for example, permutations in the extended conjugacy class \(2, 4, 5\)) in the following manner:

\[(1\ 2)(11\ 23\ 24\ 12)(19\ 37\ 38\ 39\ 20)\ (2\ 3)(8\ 25\ 26\ 9)(17\ 40\ 41\ 42\ 18)\]
\[(3\ 4)(12\ 27\ 28\ 13)(15\ 43\ 44\ 45\ 16)\ (4\ 5)(9\ 29\ 30\ 10)(20\ 46\ 47\ 48\ 21)\]
\[(5\ 6)(13\ 31\ 32\ 14)(18\ 49\ 50\ 51\ 19)\ (6\ 7)(10\ 33\ 34\ 11)(16\ 52\ 53\ 54\ 17)\ (7\ 8)(14\ 35\ 36\ 15)(21\ 40\ 41\ 42\ 18)\]

Note that we simply add elements to cycles in the \(i\)th column until the cycles in that column have length \(a_i\). Note also that this is a balanced set by construction. It is easy to verify that this also defines an Eulerian cycle, and is thus connected and split. On the other hand, we have the following result:

**Lemma 3.13** If \(B\) is equivalent to \(k\) copies of \(A\), and if there exists \(T \subset C(B)\) that generates \(S_n\), then there exists \(T' \subset C(A)\) that generates \(S_n\), and furthermore such that \(|T'| = k|T|\).

**Proof** Split each \(\sigma \in T\) into \(k\) permutations such that each of these permutations lies in \(C(A)\). These obviously generate \(S_n\) since products of them generate \(S_n\).

This proves the base case of a final induction showing that \(f(A, n) = \lceil \frac{n-1}{\Phi} \rceil\) for all \(n \geq X_0\), where
$X_0 = pc(A) + 1$. This induction will finally prove Theorem 3.12. However, once again this new induction is identical to the completions of Propositions 3.1 and 3.8, and so we refer the readers there for the completion of the proof.

4 Automorphism Groups

We devote this section to the characterization of the automorphism groups of certain $C$-graphs.

**Definition** Given a split set of cycles $T \subset S_n$, the cycle graph $\text{Cyc}(T)$ is formed by associating each vertex with an element of $N$ and drawing edges $x_1x_2, x_2x_3, \ldots, x_{k-1}x_k$ if $(x_1, x_2, \ldots, x_k)$ is in $T$. Note that this involves arbitrarily choosing a “starting” and “ending” point for each cycle in $T$. When $T$ consists of transpositions, Feng [5] refers to $\text{Cyc}(T)$ as $\text{Tra}(T)$.

**Definition** Given a split set of cycles $T$, the degree of some $t \in T$ is defined as the number of distinct points in its support that overlap with other cycles. If $t$ has degree 1, we call it a leaf.

**Definition** A split set of more than two cycles generating $S_n$ is said to be normal if any element is adjacent to at most 1 leaf, and furthermore $\text{Cyc}(T)$ is a tree (note that this is stronger than assuming that $T$ be a minimal generating set of $S_n$, as it effectively adds the criterion that $n \equiv 1 \pmod{k}$, where $T$ consists of $k$-cycles).

We use this to offer a partial generalization to a theorem by Feng [5] that states that $\text{Aut}(\text{Cay}(S_n, T)) \cong R(S_n) \rtimes \text{Aut}(S_n, T)$, where $\text{Aut}(S_n, T) = \{ \phi \in \text{Aut}(S_n) \mid \phi(T) = T \}$, and furthermore that $\text{Aut}(S_n, T) \cong \text{Aut}(\text{Tra}(T))$. In the following, $T$ will always be normal, and if we talk about a graph it will be $\text{Cay}(S_n, T)$ unless otherwise specified:

**Theorem 4.1** Let $T$ be a normal set. Then $\text{Aut}(\text{Cay}(S_n, T)) \cong R(S_n) \rtimes \text{Aut}(S_n, T)$, where $R(S_n)$ is the representation of $S_n$ as an action on $\text{Cay}(S_n, T)$.

**Proof** We use Feng’s idea of finding cycles that force graph automorphisms to be group automorphisms. Certain lemmas requiring case analysis will be dealt with in the appendix.

**Lemma 4.2** Let $t_1, t_2 \in T$. Then there exists a unique 4-cycle containing the path $t_2 \rightarrow (e) \rightarrow t_1$ iff $t_1t_2 = t_2t_1$, and furthermore the cycle will be $(e) \rightarrow t_1 \rightarrow t_1t_2 \rightarrow t_2 \rightarrow (e)$.

**Proof** See appendix.

**Lemma 4.3** Let $t_1, t_2 \in T$ such that $t_1t_2 \neq t_2t_1$. Then the 6-cycle corresponding to $t_1t_2t_1t_2t_1$ is sent to another cycle of this form under graph automorphisms when $t_1$ and $t_2$ are transpositions. Otherwise, the same statement holds for the 12-cycle corresponding to $t_1t_2t_1^{−1}t_2^{−1}t_1t_2t_1^{−1}t_2^{−1}$.

**Proof** The case of transpositions was dealt with by Feng [5]. It is easily verified that the latter construction is a cycle when $t_1$ and $t_2$ are not transpositions (it is the union of two cycles when they are transpositions). Also note that no two consecutive edges correspond to commuting generators, and this property is preserved through graph automorphisms by Lemma 4.2. It is natural to try to prove that this is the only 12-cycle going through $t_1$ and $t_2$ where no two consecutive edges commute. However, this is false, as shown by the following counterexample: Let $a = (1 2 3 4)$, $b = (1 5 6 7)$, $c = (1 8 9 10)$, $d = (1 11 12 13)$. Then $aba^{-1}b^{-1}aba^{-1}b^{-1} = abcdcb^{-1}a^{-1}b^{-1}c^{-1}b^{-1} = (e)$. We say that edge types are preserved by an automorphism if, whenever $xy_1$ and $xyz_2$ are edges corresponding to the same element of the generating set, then so are $\phi(xy_1)$ and $\phi(xyz_2)$. If we only allow use of the symbols $a,b,a^{-1},b^{-1}$ and assume that edge types are preserved, then this is indeed the only noncommuting 12-cycle, as demonstrated in the appendix. This leads to a proof of our theorem in a special case, which we will make use of:

**Lemma 4.4** Theorem 4.1 holds when $|T| = 2$, assuming that edge types are preserved.

**Proof** The preceding comments show us that commutators of generators map to commutators of generators. Thus $\phi(a)\phi(b) = \phi(ab)$ for all generators $a,b$, so that $\phi(x)\phi(y) = \phi(xy)$ for all $xy$ by induction. The induction itself is sufficiently non-trivial that we feel obliged to include it, but sufficiently technical that we will relegate it to the appendix, even though it requires no case analysis. We have thus shown that all graph automorphisms fixing $(e)$ are in fact group automorphisms as well. That this implies Theorem 4.1 we wait to prove in full generality at the end of this section.

Now for any $a,b \in T$, look at $\Gamma_0 = \text{Cay}(S_n,\{a,b\}) \subset \Gamma$. The 12-cycle described above must lie inside $\Gamma_0$. We wish to show that, for any automorphism $\phi \in \text{Aut}(\Gamma)$ fixing $(e)$, $\phi(\Gamma_0) = \text{Cay}(S_n,\{\phi(a),\phi(b)\})$, from which it will follow that commutators map to commutators in general, and
we will have proved Lemma 4.3, whence Theorem 4.1 follows from the same arguments as in Lemma 4.4.

We will, in fact, prove a stronger contention, namely that if two edges represent the same group element, then their images also represent the same group element. We first offer an automorphism-invariant criterion for determining whether two adjacent edges represent the same group element of the Cayley graph when $T$ is normal.

**Lemma 4.5** Let $x \rightarrow y \rightarrow z$ be a path in $\Gamma$. Then $xy$ and $yz$ represent the same group element if and only if the number of 4-cycles going through $xy$ equals the number of 4-cycles going through $yz$.

**Proof** Note that if $xy$ and $yz$ correspond to the same group element, then the number of 4-cycles going through $xy$ definitely equals the number of 4-cycles going through $yz$ by Lemma 4.2. (Note that this is true even if $T$ consists of 4-cycles.) The opposite direction is an easy consequence of the normality condition.

Now note that, by looking at commutativity of edges, we obtain the incidence structure of $Cyc(T)$. Thus the group elements that each edge corresponds to is uniquely determined by which edge each leaf corresponds to (this is simply a consequence of the fact that a tree is determined by the paths between terminal nodes). Thus, given an edge from $v$ corresponding to a leaf $\lambda$, whose pre-image under $\phi$ is $\lambda_0$, it suffices to prove that any edge from an adjacent vertex $w$ corresponding to $\lambda$ also has pre-image $\lambda_0$. First note that unless $|T| = 2$, which has already been dispatched of, all leaves commute. We consider two cases: adjacency between $v$ and $w$ is induced by a leaf, or the adjacency is induced by a non-leaf.

**Case one:** We may assume that all leaves commute, whence we are done by Lemma 4.2.

**Case two:** By the normality condition, the group element associated with $vw$ must commute with all but one edge, from which we are again done by Lemma 4.2. Then, noting that leaves are mapped to leaves under any graph automorphism, the final leaf only has one place to go (actually, one could make the argument that there are two places to go – to itself or to its inverse, but both of these edges correspond to the same group element, which is all that we care about).

This completes our contention, so that we are finally done with Lemma 4.3.

By Lemma 4.2, commutativity of edges is preserved through graph automorphisms. Furthermore, cycles are preserved through graph automorphisms. Thus in particular, $\{\phi(t_1), \phi(t_1)\phi(t_2), \phi(t_2), (e)\}$ must be the image of $\{t_1, t_1t_2, t_2, (e)\}$ if $\phi$ is a graph automorphism fixing $(e)$ and $t_1, t_2 \in T$ commute. This implies that $\phi(t_1)\phi(t_2) = \phi(t_1t_2)$. By the same argument, and using Lemma 4.3, $\phi(t_1)\phi(t_2) = \phi(t_1t_2)$ if $t_1, t_2 \in T$ don’t commute. Thus $\phi(t_1)\phi(t_2) = \phi(t_1t_2)$ for all $t_1, t_2 \in T$. This implies that $\phi$ is not only a graph automorphism but a group automorphism, by the same argument as in Lemma 4.4.

It follows by abuse of notation that $Aut(Cay(S_n,T))_{(e)} \subset Aut(S_n,T)$, where $G_x$ denotes the stabilizer of $x$ under the action of $G$. But it is well-known that $Aut(S_n) \cong S_n$ (the isomorphism being with the inner automorphism group) for $n \neq 6$ [5], so that $Aut(S_n,T) \subset Aut(Cay(S_n,T))_{(e)}$ when $n \neq 6$ (it is easily verified that any inner automorphism of $S_n$ preserving $T$ must also preserve incidence in $\Gamma$ and is thus a graph automorphism). Note that $n = 6$ only when $k = 2$, which has already been dispatched, so the theorem holds for all $n$ that we care about. Since $Aut(Cay(S_n,T)) = R(S_n)Aut(Cay(S_n,T))_{(e)}$ and the two subgroups have trivial intersection, we will have a complete characterization of $Aut(Cay(S_n,T))$ if we can show that $R(S_n)$ is normal. This follows since $R(S_n)$ is closed under conjugation by elements of $Aut(S_n,T)$. Thus we have that $Aut(Cay(S_n,T)) \cong R(S_n) \times Aut(S_n,T)$, as stated.

**Comment** Though it is always regrettable when a result cannot be proven in full generality, we claim that the normality condition is relatively weak. Indeed, given any set $T$, we can define a normalization of $T$ to be a new $C$-graph obtained from $T$ by adding another cycle incident on each leaf of $T$. It is easily verified that this results in a normal set.

**Comment** Though non-normal generating sets for $S_n$ are too big to test, the characterization works for $A_7$ with the generating set $T = \{(1 2 3), (1 3 2), (1 4 5), (1 5 4), (1 6 7), (1 7 6)\}$, as the automorphism group has size 120960 (computed by nauty). In this case the group remains the semidirect product of $A_7$ and the automorphisms of $S_7$ fixing $T$.

## 5 Quasi-hamiltonicity

For the sake of future work on the conjecture of Rappaport-Strasser and on Hamiltonicity of directed graphs in general, we generalize the work of Gutin and Yeo to on quasi-hamiltonicity to undirected graphs (see [10] for the original paper). We will assume that $R \subset V(\Gamma)$. 
Theorem 5.1 Given \( \Gamma \), the last two vertices must also be connected.

Definition Let \( QH_{k}(\Gamma, R) := \{ e \in E(\Gamma) \mid e \cup R \text{ is in a cycle factor } \} \). For \( k > 1 \), let \( QH_{k}(\Gamma, R) := \{ e \in E(\Gamma) \mid QH_{k-1}(\Gamma, e \cup R) \text{ is connected } \} \). Then we say that \( \Gamma \) is \( k \)-quasi-hamiltonian if \( QH_{k}(\Gamma, \{ \}) \) is connected.

Obviously \( k \)-quasi-hamiltonicity in an undirected graph implies \( k \)-quasi-hamiltonicity in the associated digraph. Indeed, it is equivalent to \( k \)-quasi-hamiltonicity for digraphs if we disallow cycles of length 2 in the cycle factor. In particular, an undirected graph is Hamiltonian if and only if \( (n - 2) \)-quasi-hamiltonian, since this implies the existence of a cycle factor containing \( n - 2 \) connected vertices (so the last two vertices must also be connected).

Theorem 5.1 Given \( \Gamma \), define the bipartite graph \( B(\Gamma) \) to have vertex set \( T_{1} = \{ x_{1}, \ldots, x_{m} \} \cup T_{2} = \{ y_{1}, \ldots, y_{m} \} \), where \( m = |V(\Gamma)| \), and there exists a directed edge from \( x_{i} \) to \( y_{j} \) if vertices \( i \) and \( j \) are adjacent in \( \Gamma \). Create a flow network where each edge in \( B(\Gamma) \) has capacity 1 and there is a source \( s \) with an edge of capacity 2 into every vertex in \( T_{1} \), similarly an edge of capacity 2 from every vertex in \( T_{2} \) into a sink \( t \). Then there exists a cycle factor in \( \Gamma \) containing \( R \) if and only if there exists a flow of \( 2m \) from \( s \) to \( t \), such that all edges pertaining to elements of \( R \) have flow going through them.

Proof Suppose that there exists a cycle factor of \( \Gamma \) containing \( R \). Push flow through \( x_{i}y_{j} \) and \( x_{j}y_{i} \) iff the edge \( ij \) is in the cycle factor. This gives the desired flow. Now suppose that we have such a flow and wish to construct a cycle factor. It is well-known that we can “force” flow to go through an edge by finding an augmenting path containing that edge and then not adding the back-flow through that edge when we push flow through the augmenting path. Thus asking for the existence of such a flow is equivalent to forcing flow through all of the edges pertaining to \( R \) (for brevity, from now on we will call this “forcing flow through \( R \)”) and asking for the existence of a flow of \( 2(m - |R|) \) in the resulting graph. Since any choice of augmenting paths must give us the same maximum flow, we can choose any set of augmenting paths that forces flow through \( R \). In particular, given any augmenting path \( P \), we can define another path \( r(P) \) to be the path obtained by replacing all instances of \( x_{i} \) with \( y_{i} \) (and vice versa) and reversing the orientation of each edge in \( P \). Note that \( P \) and \( r(P) \) are edge-disjoint since \( \Gamma \) contains no self-loops. If whenever we augment by a path \( P \), we also augment by \( r(P) \), then it will be true that \( P \) is an augmenting path iff \( r(P) \) is an augmenting path. In particular, we do this while forcing flow through \( R \). We then continue to do this while performing the maxflow algorithm. By the symmetry of our algorithm, after we have completed it there will be flow through an edge \( x_{i}y_{j} \) iff there is flow through an edge \( x_{j}y_{i} \). Now take the subgraph \( \Gamma_{0} \) of \( \Gamma \) formed by all edges \( ij \) such that there is flow through \( x_{i}y_{j} \) in \( B(\Gamma) \). Since we have a flow of \( 2m \) by assumption, every vertex in \( \Gamma_{0} \) has degree 2, thus is a cycle factor, completing the theorem.

Due to the high degree of symmetry of Cayley graphs, if \( \Gamma \) does not contain a Hamiltonian cycle then it is (intuitively) likely to have a quasi-hamiltonicity number sufficiently high that it is infeasible to check. We would thus like a more efficient block to Hamiltonicity for Cayley graphs.

Definition A subset \( T \) of a group \( G \) is said to have a left coset partition if there exists a set \( S \) such that \( s_{1}T \) and \( s_{2}T \) are disjoint for distinct \( s_{1}, s_{2} \in S \), and such that \( ST = G \).

Definition A cycle factor is said to be symmetric if it is also a left coset partition.

Note that any Hamiltonian cycle is also a symmetric cycle factor. We can thus define the analogous form of quasi-hamiltonicity where all cycle factors are required to be symmetric. Given a sufficiently crisp characterization of sets with coset partitions, it seems likely that a more effective algorithm for Hamiltonicity blocks could be designed.

6 Conclusion and Open Problems

A minor but interesting detail of this paper is the dependence of our bound on \( X_{0}(A) \) on the existence of certain primes. There is no reason to believe that this bound should be strict, and so a more complete understanding of \( C \)-trees may be reached by a more precise study of the properties of \( X_{0}(A) \). If the bound is given by explicit constructions, then the result of such a study would also be smaller Cayley graphs that would be more feasible to analyze empirically.

Disregarding our poor understanding of \( X_{0}(A) \), \( C \)-trees have been effectively characterized. With this stepping stone, it would be useful to define some more concepts related to \( C \)-graphs (in a structurally interesting way). After a tree, the next simplest definition to make is that of a cycle. For a possible idea, we will
borrow ideas from matroid theory. We call a set independent
type if it is a subset of a tree, and dependent otherwise. A
simple cycle is then a subset of $T$ that is dependent, but whose proper subsets are all independent.
Of course, any other algebraic properties of graphs that could carry over to $C$-graphs would also
be interesting. The author believes that an alternate definition of $C$-trees leading to a nice matroid struc-
ture on the power set of $T$ would make all remaining generalizations transparent. The trees under this
structure would also most likely lead to even more structured Cayley graphs.

We have fully characterized the automorphism groups of certain $C$-trees. We would like a general-
ization, at the very least, of arbitrary split sets consisting of $k$-cycles. We conjecture that Theorem 4.1
holds for all such sets. Similarly, we seek a general-
ization of Feng’s theorem regarding the isomorphism
between $\text{Aut}(\text{Tra}(T))$ and $\text{Aut}(S_n, T)$ that gives a rel-
ation between the automorphism groups of $\text{Cyc}(T)$ and $\text{Aut}(S_n, T)$.

We turn to the spectral analysis of the Cayley
graphs in question. In particular, we look at the Cay-
ley graphs formed when $|T| = 2$. Clearly the largest
eigenvalue is 4 in this case, since, up to a scaling, a
positive semidefinite matrix. It is interesting to note
that the second-largest eigenvalue is $1 + \sqrt{3}$ when
$k = 2$ and $1 + \sqrt{7}$ when $k = 4$ (the latter was established empirically). It is therefore very tempt-
ing to conjecture that the second-largest eigenvalue is always $1 + \sqrt{2k-1}$, but this is unfortunately nonsense since it can never exceed 4.

Most importantly, this paper points to a deeper
connection between Cayley graphs formed by trans-
positions and by $k$-cycles. This is structurally ap-
parent in the similarities between the two in terms
of commutativity and conjugacy, and indicates that
more results should generalize to the case of $k$-cycles.
For example, in [1] the Coxeter representation of
the transpositions is used to gain information about the
spectrum of the Cayley graphs. A generalization of
the notion of Coxeter representation to account for
$k$-cycles would likely allow for the generalization of
these results.

Finally, we propose a more general mathematical
program to understand the nature of Cayley graphs
formed by conjugate generating sets in general, which
we believe to be a distinguished variety. So far, all
theorems regarding these graphs show that the auto-
morphism group is minimal in a certain sense. We
propose the task of finding cases when the group is
not minimal, but is close to minimal, and analyzing
what happens there, when everything should be more
transparent. This should point us towards more gen-
eral results regarding these graphs.

7 Appendix

7.1 A Bound on Dirichlet’s Theorem

Theorem 7.1 For each $n > 1$, there is a prime of
the form $kn + 1$ that divides $\Phi_n(n)$.

Proof It can be shown that if $p|\Phi_n(j)$, then either
$p|n$ or $p \equiv 1 \pmod{n}$. But $\Phi_n(n) \equiv 1 \pmod{n}$, so
we must have the second case. Additionally, $\Phi_n(n) =
\prod n - \xi$, for each primitive root of unity $\xi$. But

$$\prod n - \xi = \sqrt{\prod(n - \xi)(n - \xi)} = \sqrt{\prod n^2 + 1 - 2n \cos(\theta)} > \sqrt{\prod n^2 + 1}$$

so we must have some prime dividing $\Phi_n(n)$, and
we are done.

7.2 Case Analysis for 4-cycles

We are looking for $a’$ and $b’$ such that $abab = (e)$, or equivalently $bab’ = (e)$. Thus (for the
supports of $ab$ and $ab’$ to be the same) $a’b’ \in$
{$ab, ab^{-1}, a^{-1}b, a^{-1}b^{-1}, ba$}. $a’$ cannot be $b^{-1}$ and $b’$ cannot be $a^{-1}$ since this would correspond to a path
doubling back on itself. Since the product of two split
$k$-cycles is a $2k - 1$-cycle, $abab$ is a $2k - 1$-cycle, and
in particular not the identity. $aba^{-1} = ab(a^{-1})$ is
the product of two $k$-cycles with different supports,
and so is again not the identity. Similar logic holds
for $aba^{-1}b = (aba^{-1})b$. $aba^{-1}b^{-1} = (e)$ implies
that $ab = ba$, which is what we want. Finally, $aba = (e)$
implies $aabb = (e)$, which is impossible since $aa$ and
$bb$ have different supports.

7.3 Case Analysis for Commutators

We first look at all possible cycle structures of the
product of four of $a, b, a^{-1}, b^{-1}$ such that no
two adjacent terms in the product correspond to
the same generator. WLOG, $a = (1 \ 2 \ \ldots \ k)$ and
$b = (k \ k + 1 \ \ldots \ 2k - 1)$. Down to symmetry, there
are only four cases: $abab, abab^{-1}, ab^{-1}b, ab^{-1}b^{-1}$.

Case one: $abab$. $ab = (1 \ 2 \ \ldots \ k - 1 \ k +
1 \ k + 2 \ \ldots \ 2k - 1 \ k)$, so $(ab)^2 = (1 \ 3 \ \ldots \ k - 1 \ k +
2 \ k + 4 \ \ldots \ 2k - 2 \ k \ 2 \ \ldots \ k + 1 \ \ldots \ 2k - 1)$, i.e. a $(2k - 1)$-cycle.

Case two: $abab^{-1}$. $bab^{-1} = (1 \ 2 \ \ldots \ k - 1 \ 2k - 1)$. $a(bab^{-1}) = (1 \ 3 \ \ldots \ k - 1 \ k \ 2 \ 4 \ \ldots \ k - 2 \ 2k - 1)$, i.e. a $(k + 1)$-cycle.
Case three: \(abab^{-1}b\). \(abab^{-1} = (k - 1 + k + 1 + k + 2 \ldots 2k - 1)\). Analogously to case two, \((abab^{-1})b = (k+1 k+3 \ldots 2k-3 2k-1 k-1 k+2 k+4 \ldots 2k-2 k)\), again a \((k+1)\)-cycle.

Case four: \(abab^{-1}b^{-1}\). A simple calculation yields that \(abab^{-1}b^{-1} = (k - 1 \ 2k - 1)\).

We denote case 1 by \(w(a,b)\), and denote cases 2, 3, 4 analogously by \(x(a,b), y(a,b), z(a,b)\). We will consider \(f_1(e_1, e_2)f_2(e_3, e_4)\) for \(f_1, f_2 \in \{x, y, z\}\), \(e_1, e_2, e_3, e_4 \in \{a, b, a^{-1}, b^{-1}\}\) and show that except for \(z(e_1, e_2)z(e_1, e_2)\), either some two elements in the defined product commute, or the product is not equal to \(f_3(e_3, e_5)\) for any \(f_3, e_5, e_6\). WLOG \(e_1, e_2 = a, b\). Finally, unless we have the cycle \(z(a, b)z(a, b)\), there must be some three consecutive elements of the form \(e_1 e_2 e_1\), WLOG \(aba\). Furthermore, unless we have the cycle \(w(a, b)w(a, b)w(a, b)(a)\) (which is a \(2k - 1\)-cycle raised to the \(6\)th power, thus obviously not the identity unless \(k = 2\)), we must have four consecutive elements of the form \(abab^{-1}\). Thus we need only consider \(f_1 = x\). Finally, \(x(a, b)f_2(e_3, e_4)\) will either loop back on itself or contain two consecutive elements of the same type if \(e_3 \in \{b, b^{-1}\}\), so we omit these cases. This leaves only 16 cases to check: \(f_1 = x(a, b)\) and \(f_2 \in \{w(a, b), w(a, b^{-1}), w(a^{-1}, b), x, y, z, \ldots \}\). We begin with a list of \(f_1(e_1, e_2)\) for all \(f_1, e_2, e_3\). We also introduce a more concise notation for listing the permutations.

We let \([s, e, d]\) denote a permutation \((\ldots s s + d s + 2d \ldots e - d e \ldots)\). Thus, for example, \((1 \ 3 \ \ldots k - 1 k + 2 k + 4 \ldots 2k - 2 2 k \ldots k - 2 k + 1 \ldots 2 k - 1)\) is denoted \([1, k-1, 2][k+2, 2k-2, 2][k, k-2, 2][k+1, 1, 2k-1, 2]\). Multiplication is denoted by colons, and \([k, k, 1]\) is shortened to just \(k\). If we have, say, \([i, i, 1][j, j, 1]\), we denote it \(i, j\). Finally, instead of writing, say, \([e, s, -d]\) as the inverse of \([s, e, d]\) when \(s < e\), we simply write \([e, s, d]\). With this notation, we list the desired products:

1. \(w(a, b) = [1, k - 1, 2][k + 2, 2k - 2, 2][k + 2, 2k - 2, 2][k + 1, 2k - 1, 2]\)
2. \(w(a, b^{-1}) = [1, k - 1, 2][2k - 2, k, 2][2k - 2, 2][2k - 1, 1, 2]\)
3. \(w(a^{-1}, b) = [k, 2, 2][k + 1, 2k - 1, 2][k - 1, 1, 2][k + 2, 2k - 2, 2]\)
4. \(w(a^{-1}, b^{-1}) = [k, 2, 2][k - 1, k + 2, 2][k - 1, 1, 2][k - 1, 1, 2][2k - 2, k + 2, 2]\)
5. \(x(a, b) = [1, k - 1, 2][k, k - 2, 2][k - 1, 1, 2]\)
6. \(x(a, b^{-1}) = [1, k - 1, 2][k, k - 2, 2][k - 1, 1, 2]\)
7. \(x(a^{-1}, b) = [k, 2, 2][k - 1, k - 1, 1, 2]\)
8. \(x(a^{-1}, b^{-1}) = [k, 2, 2][k^2, k - 1, 1, 2]\)
9. \(y(a, b) = k - 1 + 2 k - 2, 2][k + 1, 2k - 1, 2]\)
10. \(y(a, b^{-1}) = [2k - 1, k + 2, 2k - 2, k - 1, 2]\)
11. \(y(a^{-1}, b) = [1 + 2 2, 2k - 2, 2][k + 1, 2k - 1, 2]\)
12. \(y(a^{-1}, b^{-1}) = [2k - 1, k + 1, 2][2k - 2, k, 2]\)
13. \(z(a, b) = k - 1 : k : 2k - 1\)
14. \(z(a, b^{-1}) = k - 1 k + k + 1\)
15. \(z(a^{-1}, b) = 1 k : 2k - 1\)
16. \(z(a^{-1}, b^{-1}) = 1 k : k + 1\)

Note that only \(2k - 1, k + 1, 3, 1\) can be sent to \(1\). Since \(1\) is sent to \(3\) under \(x(a, b)\), \(f_2\) must send \(3\) to \(2k - 1, k + 1, 3\), or \(1\). Only \(w(a^{-1}, b)\), \(w(a^{-1}, b^{-1})\), \(x(a^{-1}, b)\), \(x(a^{-1}, b^{-1})\), \(y\), and \(z\) do this, and they all send \(3\) to either \(3\) or \(1\), so we can consider only \(f_3\) that do this. Thus either \(f_2 \in \{y, z\}\) and \(f_3 \in \{w(a^{-1}, b), w(a^{-1}, b^{-1}), x(a^{-1}, b^{-1})\}\) or vice versa.

If \(f_3 = x\), then a cyclic rotation of \(f_1, f_2, f_3\) causes \(f_1 = x, f_2 = x\). Thus we can assume that either \(f_2, f_3 \neq x\), or \(f_2 = x\). When \(f_2 = x\), we have either \(x(a, b) x(a^{-1}, b)\) sends \(k \rightarrow 2k - 1 \rightarrow k\), or \(x(a, b) x(a^{-1}, b^{-1})\) sends \(k \rightarrow k + 1 \rightarrow k\). None of \(w, x, y, z\) reverse this, so we can assume that \(f_2, f_3 \neq x\). Thus either \(f_2\) or \(f_3\) is \(w\). But if \(x f_2 w = (e)\), then \(w f_2 w = (e)\), so we need only check that \(x w = w x\) and cannot be reversed by any \(f_2\).

Case one: \(x(a, b) w(a^{-1}, b)\). Contains the cycle \(k - 2[2k - 3, k + 1, 2]k - 1\) which is a \((\frac{k}{2} + 1)\)-cycle. This only has the same cycle length as some \(f\) when \(k = 4\), in which case it is a \(3\)-cycle, but then the cycle is \((2 5 3)\) which is not equal to \(e\) (as \(4\) is always in the \(3\)-cycle for \(z\)).

Case two: \((a, b) w(a^{-1}, b^{-1})\). Also contains the cycle \(k - 2[2k - 3, k + 1, 2]k - 1\), whence we are done by the same argument as case one.

Case three: \((a^{-1}, b)x(a, b)\). Since \(gh\) and \(hg\) are conjugate, this contains a \(3\)-cycle by the same
argument as case one. Thus we need only worry about \( k = 4 \), when we have \((4 \ 2 \ 5 \ 7 \ 3 \ 1 \ 6)(1 \ 3 \ 4 \ 2 \ 5)\), which contains the cycle \((2 \ 1 \ 6)\), which is again not equal to any \( z \).

**Case four:** \( w(a^{-1}, b^{-1})x(a, b) \). We need only consider \( k = 4 \) by the same argument as case three, in which case the permutation similarly contains \((2 \ 1 \ 6)\).

This completes our case analysis.

### 7.4 Induction Argument for Lemma 4.4

We have already shown that \( \phi(a) \phi(b) = \phi(ab) \) for any automorphism \( \phi \) of \( \Gamma \) fixing \((e)\). We have the following lemma:

**Lemma 7.2** If \( \phi \) is a (graph) automorphism of \( \Gamma \), then so is \( \phi_y = \phi(y^{-1})\phi(yx) \).

The proof is a routine verification. Now, we wish to show by induction that

\[
\phi(t_1 t_2 \ldots t_n) = \phi(t_1)\phi(t_2)\ldots\phi(t_n)
\]

for all \( \phi \in \Gamma \). Now note that

\[
\phi(t_1 t_2 \ldots t_n) = \phi(t_1)\phi(t_1^{-1} t_2 \ldots t_n) = \phi(t_1)\phi(t_1)\ldots\phi(t_1)
\]

where the equality between the second and third expressions follows by the inductive step. This completes our induction.

### 8 Acknowledgements

The author would like to thank John Dell, Dave Jensen, Alfonso Gracia-Saz, Jim Lawrence, and Brendan McKay for their help, as well as all of the staff of MathCamp 2007 and Thomas Jefferson High School for Science and Technology.

### References


Conformal Mapping Using the Schwarz-Christoffel Transform
2007-2008
Evan Warner

Abstract

The Schwarz-Christoffel transform is a conformal mapping from the upper half of the complex plane to a polygonal domain. It allows many physical problems posed on two-dimensional, polygonal regions, such as heat flow, fluid flow, and electrostatics, to be solved numerically. This type of problem cannot generally be solved in closed form; the Schwarz-Christoffel transform provides an exceptionally accurate method of solution. This paper describes the implementation of a working software unit that efficiently and accurately calculates Schwarz-Christoffel transforms and inverses. The program incorporates graphical, easy-to-use interfaces and will contain resources to aid in solving physical problems. Future research into mathematical extensions to the Schwarz-Christoffel transform, such as the inclusion of simple curves, will be conducted.

Keywords: Schwarz-Christoffel transform, conformal mapping, numerical analysis, Laplace’s equation, fluid flow, heat flow

1 Introduction

Many physical problems are expressed as differential or boundary value problems over a surface. Often, these surfaces are or can be approximated by two-dimensional polygons. When this occurs, one method of determining accurate solutions is by assuming the polygonal domain exists in the complex plane and determining a conformal map, which preserves the structure of Laplace’s equation, that restates the problem in a simpler domain. Here, the upper half-plane is used. A solution to the problem, now easy to solve analytically or numerically, is then mapped back to the original domain. For such polygonal domains, a method of determining the specific transform needed is provided by the following formula, known as the Schwarz-Christoffel transform:

\[ f(z) = A \int_0^z \prod_{j=1}^n (\zeta - x_j)^{-\theta_j/\pi} d\zeta + B. \]  

In this formula, \( \zeta \) is an independent complex variable in the upper half-plane, the \( \theta_j \) are the exterior angles of the polygon, the \( x_j \) are prevertices of the mapping (given along the real axis), \( n \) is the number of vertices of the polygon, and \( A \) and \( B \) are complex constants that specify the location, size, and orientation of the image polygon in the complex plane. The \( \theta_j \) must satisfy

\[ \sum_{j=1}^n \theta_j = 2\pi, \]  

which ensures the completeness of the image polygon [2]. Unfortunately, the Schwarz-Christoffel formula is not easy to evaluate, and requires both effective integration algorithms and an efficient, convergent method to solve a specific nonlinear equation. Implementation of such numerical routines is not a trivial problem, and is the subject of this paper.

The initial project may be divided into four separate problems. First, a method to effectively evaluate integrals of the form found in the Schwarz-Christoffel formula is required. Second, a numerical algorithm to solve the so-called Schwarz-Christoffel parameter problem, a system of nonlinear equations for the prevertices, must be developed. Third, methods to evaluate the forward and backward transforms based on given prevertices must be coded. Fourth, a user interface is needed, which should be robust and accessible to allow nonspecialists to systematically solve various physical problems. The four components may be coded simultaneously or in series, as they are by nature almost entirely separable problems. In this project, each part was coded in series.

Subsequent research will be done into improvements and optimizations to the numerical algorithms for various subproblems and extensions. These include the problem of mapping polygons with large aspect ratios, which are generally highly ill-conditioned,
and the extension of the Schwarz-Christoffel formula to simple curves. The goal of the project is thus twofold: to produce a piece of software that will be useful in the solution of real, physical problems, and to improve upon current algorithms for producing the Schwarz-Christoffel transform.

2 Background

The Schwarz-Christoffel transform was first discovered independently in the late 1860s by Elwin Christoffel and Hermann Schwarz. Schwarz used some of the ideas of the transform to provide a more rigorous proof of the Riemann Mapping Theorem, which he had previously shown to be incomplete, but the majority of this work was on a purely theoretical level [5]. The usefulness of the transform was mitigated by the formula’s unwieldiness, as the mappings for all but the simplest domains could not be calculated in closed form. Numerical estimates, especially for nonsymmetric polygons with four or more vertices, could not be effectively calculated by hand. Application to physical problems, therefore, was limited at best until the advent of the computer. A computer algorithm to compute the Schwarz-Christoffel transform was first written in the 1960s, and others have been written and modified since then [3].

The first problem in calculating the Schwarz-Christoffel mapping is the evaluation of the integral given by Eq. (1). The integrand contains singularities at each of the endpoints of the image polygon, which tend to render ordinary numerical integration routines either useless or hopelessly slow. In addition, the presence of negative powers in $f$ means that domains of applicability for each of the subfunctions $(\zeta - x_j)^{-\theta_j/\pi}$ must be chosen so that the entire domain in and immediately around the image polygon is meromorphic. Although several quadrature routines have been used for this problem, the method of choice today is Gauss-Jacobi quadrature, which uses a specially-tailored weighting function to choose points of evaluation and weights for the points that maximize efficiency. In practice, the Schwarz-Christoffel formula is altered so that the last prevertex, $x_n$, is chosen to be both $-\infty$ and $+\infty$ (the values are equivalent for a conformal map, which acts on the Riemann sphere). This can always be done due to the extra degrees of freedom contained in Eq. (1). The integrals that must be evaluated in practice in the course of the Schwarz-Christoffel transform are of the form

$$\int_{x_{i-1}}^{x_i} \prod_{j=1}^{n-1} (\zeta - x_j)^{-\theta_j/\pi} d\zeta.$$ (3)

These integrals can always be written as required for Gauss-Jacobi quadrature; that is, in the form

$$\int_{a}^{b} (z - a)\alpha (z - b)\beta \psi(z) dz, \quad (4)$$

where $\alpha$ and $\beta$ are real numbers greater than $-1$.

The points and weights of a Gauss-Jacobi quadrature are calculated here using a routine from Numerical Recipes [4] which efficiently estimates and solves for the roots of the Jacobi polynomials, which form the sample points just as the roots of the Chebyshev polynomials form the sample points for standard Gaussian quadrature. These points, however, are uniformly calculated in the range $[-1, 1]$, and the integrals must be adjusted slightly to conform to this range. During the calculation of the prevertices, discussed below, the $z$ in Eq. (4) are restricted to the real axis; however, in direct calculations once the prevertices have been found, the $z$ will generally be fully complex.

The second problem is the Schwarz-Christoffel parameter problem, where the $x_j$ in Eq. (1) are calculated. As described in [2], a series of nonlinear, constrained equations can be formed from the requirement that the image polygon and the desired polygon be similar (the constants $A$ and $B$ in Eq. (1) then ensure congruency). Written out, there are $n - 3$ linear equations in $n - 3$ unknowns, once the extra degrees of freedom have been taken care of by arbitrarily giving three of the $x_j$ precise values. Here, as in the literature, we take $x_1 = -1$ and $x_2 = 0$ in addition to the already-defined $x_n = \pm \infty$. The equations to be solved, ensuring that the target polygon and the image of the Schwarz-Christoffel map are similar, are then

$$\frac{|\int_{x_{i-1}}^{x_i} \prod_{j=1}^{n} (\zeta - x_j)^{-\theta_j/\pi} d\zeta|}{|\int_{x_1}^{x_2} \prod_{j=1}^{n} (\zeta - x_j)^{-\theta_j/\pi} d\zeta|} = \frac{|w_i - w_{i-1}|}{|w_2 - w_1|} = 0, \quad (5)$$

where the $w$ are the target vertices, $i = 3, 4, ..., n - 1$, and all other variables are as defined above. However, there is an additional complication, as preserving the order of the prevertices on the real axis is important. The extra constraint can be expressed as

$$1 < x_3 < x_4 < ... < x_{n-1} < \infty. \quad (6)$$

The unconstrained problem is relatively easy to solve; however, the constraint prevents a naive application of a Newton’s Method variant to this problem. To get around this, Trefethen in [3] suggests a simple change of variables that ensures the inequalities of Eq. (6).
Take a new series of variables, \( \chi_j \), and let
\[
\chi_j = \ln (x_j - x_{j-1}). \tag{7}
\]
The resulting \( \chi_j \) will automatically obey Eq. (6), and the original \( x_j \) are found by the simple inverse formula
\[
x_j = x_{j-1} + e^{\chi_j}. \tag{8}
\]
This new set of equations in the \( \chi_j \) is readily solved by a variant of Newton’s Method that does not require the calculation of the Jacobian matrix (which would be hopelessly complex), but rather uses progressive estimates.

### 3 Development

The software has been written entirely in Java, although certain routines may be later written in C to increase speed if there is a bottleneck at any point in the process. The entire development of the program is designed to be achieved in stages by attacking the subproblems individually. The following is a list of classes, with short descriptions, written up to this point:

- **class Complex** - this class stores and performs arithmetic on complex numbers, which are not directly supported by Java. Several of the methods, including the multiplication and division algorithms, are designed to run as quickly as possible while avoiding intermediate overflow and floating-point error propagation. The multiplication method, for instance, requires only three real multiplications rather than four. (see Appendix for examples)

- **class GaussJacobiWeights** - this class calculates and stores the sample points and weights for a given Gauss-Jacobi quadrature over the interval \([-1, 1]\). This routine uses Newton’s Method to find the roots of the Jacobi polynomials, which are the sample points for the integral, and was taken and translated from [4].

- **class SchwarzFunction** - this class evaluates the integrand of a given real-valued Schwarz-Christoffel integral, serving as a storage class for data of this kind.

- **class GaussQuad** - this class accepts as input \( \psi \), \( a, b, \alpha, \) and \( \beta \) from Eq. (4). For an arbitrary integral in that form, shifting and scaling the bounds produces the equivalent integral
\[
e^{a+\beta+1} \int_{-1}^{1} (\zeta - 1)^{\alpha}(\zeta + 1)^{\beta} \psi(c\zeta + m) \, d\zeta, \tag{9}
\]
where \( b = \frac{a+b}{2} \) and \( c = b - m = m - a \). This integral is then evaluated using the sample points and weights given by the GaussJacobiWeights class and returned. For any GaussQuad object, varying numbers of sample points (and thus varying accuracy) are accepted by its integrate() method. (see Appendix for example code)

- **class RealNewtonRaphson** - this class accepts an array of vertices and calculates the necessary prevertices as well as the constants \( A \) and \( B \) from Eq. (1). The method employs a standard Newton-Raphson method to solve the Eq. (5). At each step, an approximate Jacobian matrix for the function is calculated using a forward-difference method in each dimension; the step vector is then solved for using an LU factorization on the equation
\[
J \delta \vec{x} = \vec{f}, \tag{10}
\]
where \( J \) represents the Jacobian, \( \delta \vec{x} \) the step vector, and \( \vec{f} \) the current function vector. Note that by employing a forward-difference method to find the Jacobian, the number of function evaluations can be cut in half, as the current function vector can be reused in the Jacobian calculation. (see Appendix for example code)

- **class ForwardGaussQuad** - this class, using already-calculated values for the prevertices, evaluates the Schwarz-Christoffel integral at a given point. To minimize error caused by the presence of singularities near the path of the integral (the singularities at the endpoints are handled by the Gauss-Jacobi quadrature), the path of integration is divided recursively such that no segment is closer to a singularity than one-half its length, a technique employed in [3]. Such recursive subdivision is known as compound Gauss-Jacobi quadrature.

- **class SchwarzChristoffel** - this class runs the graphical user interface and calls RealNewtonRaphson and ForwardGaussQuad when necessary. The graph itself has the ability to show axes and manually adjust window parameters.

In future iterations of the project, a new set of routines will be implemented to calculate continuous Schwarz-Christoffel problems. Immediately following from Eq. (3) above, we have
\[
f'(z) = A \prod_{j=1}^{n-1} (\zeta - x_j)^{-\theta_j/\pi}. \tag{11}
\]
To change this into a continuous problem, we can rewrite this as

$$f'(z) = Ae^{\frac{1}{2} \sum_{j=1}^{n-1} -\theta_j \ln (z-x_j)}.$$  \hspace{1cm} (12)

Then, defining the natural logarithm function as single-valued in the upper half-plane, except where $x_i = z$, $f'$ becomes an analytic function in the required domain. To formulate the continuous-boundary problem, we simply replace the sum in Eq. (12) with an integral, and integrate the entire function to find $f(z)$:

$$f(z) = A \int_0^z e^{\frac{1}{2} \int_{-\infty}^{\infty} -\theta(x) \ln (\zeta-x) d\zeta} d\zeta + B, \hspace{1cm} (13)$$

where $\theta(x)$ represents the amount of turning per unit length on the real axis, such that

$$\int_{-\infty}^{\infty} \theta(x) dx = 2\pi. \hspace{1cm} (14)$$

The continuous problem therefore has an extra subproblem to solve, namely, the solution of the integral equation, Eq. (13), to find $\theta(x)$ at every $x$.

The majority of testing of the program is specific to a single numerical routine; that is, each of the algorithmic components are tested individually. To calculate the GaussQuad routines, for instance, randomly generated sample problems are solved by MATLAB to provide an approximate check on the accuracy of solutions, then precision is achieved by manipulating the number of sample points used for the quadrature.

Shortly, a program specifically designed to track approximate error propagation and runtimes of each of the components will be developed. For the majority of the routines used in the program, strict error bounds can be calculated, and for the remaining algorithms error can be accurately estimated. For all subproblems, as well as the entire routine, plots of runtime versus precision will be generated to examine the efficacy of each routine. For the final program, random polygonal generation will be used to dynamically test the program for a range of inputs.

### 4 Results and Discussion

The purpose of this project was to calculate and display Schwarz-Christoffel transforms, which conformally map the upper half-plane to an arbitrary polygon, efficiently and accurately. In addition, additional research into the Schwarz-Christoffel transform itself, including its extension to curved target domains, was investigated. The evaluation of the Schwarz-Christoffel formula involves several parts, including the efficient calculation of a certain class of integrals as well as a solver of nonlinear systems of equations. Solving the continuous-parameter problem will require numerical solutions to a certain class of integral equations.

Results on accuracy versus time data are forthcoming; however, preliminary results have confirmed that answers are correct to within the modest relative error tolerance of about $10^{-4}$. Nevertheless, far greater accuracy is expected once dynamic testing begins.

The first problem, that of numerical integration, has been solved and refined, and a basic user interface has been designed. The second problem, that of a nonlinear equation solver to calculate the prevertices, has also been completed to satisfaction; current research focuses on correct implementation of the forward transform using given prevertices. Preliminary results indicate the general correctness but inexactitude of the forward transform in the absence of compound quadrature, especially near the boundaries of the given polygon. It is hoped that a full implementation of the compound quadrature will ameliorate these concerns.

The completed program will be useful on several levels: as a teaching aid, and as a tool for researchers solving certain equations on polygonal regions. Once the basic Schwarz-Christoffel problem is numerically solved, the program can form an easy basis for testing research in numerical analysis and mathematics that deals with improving or expanding the Schwarz-Christoffel transform.

### Appendix

**Code for class Complex**

```java
public class Complex {
    // ... public Complex multiply(Complex z) {
    double temp1=x*z.real();
    double temp2=y*z.imag();
    return new Complex(temp1-temp2,(x+y)*(z.real()+z.imag())-temp1-temp2);
    }
    public Complex divide(Complex z) {
    double temp2=z.imag()/z.real();
    double denominator=z.real()*temp2+z.imag();
    if(Math.abs(z.real())>=Math.abs(z.imag()))
    {
        double denominator2=z.real()+z.imag()*temp2;
        return new Complex((x+y*temp2)/denominator,(y-x*temp2)/denominator);
    }
    else
    {
        double denominator2=z.real()*(x+y*temp2)/denominator;
        return new Complex((x+y*temp2)/denominator,(y-x*temp2)/denominator);
    }
    public double modulus() {
    if(z==0&&y==0) { 
    // ... }
```
return 0.0;
else if(y==0)
    return Math.abs(x);
else if(x==0)
    return Math.abs(y);
if(Math.abs(y)>=Math.abs(x))
    return Math.abs(x)*Math.sqrt(1.0+(y*y)/(x*x));
else
    return Math.abs(y)*Math.sqrt(1.0+(x*x)/(y*y));
}

public Complex sqrt(){
    double w=0;
    if(x==y&&y==0)
        return new Complex(0.0,0.0);
    else if(Math.abs(x)>=Math.abs(y))
        w=Math.sqrt(Math.abs(x))*Math.sqrt((1.0+Math.sqrt(1.0+(y*y)/(x*x)))/2.0);
    else
        w=Math.sqrt(Math.abs(y))*Math.sqrt((Math.abs(x/y)+Math.sqrt(1.0+(x*x)/(y*y)))/2.0);
    if(x>=0)
        return new Complex(w,y/(2*w));
    else if(y>=0)
        return new Complex(Math.abs(y)/(2*w),w);
    else
        return new Complex(Math.abs(y)/(2*w),-w);
}

public Complex power(double a){
    double theta=this.argument();
    double r=this.modulus();
    Complex temp = new Complex(Math.cos(theta*a),Math.sin(theta*a)).multiply(Math.pow(r,a));
    return temp;
}

public Complex ln(){
    double theta=this.argument();
    double r=this.modulus();
    return new Complex(Math.log(r),theta);
}

public Complex exp(){
    double etothex=Math.exp(x);
    return new Complex(etothex*Math.cos(y),etothex*Math.sin(y));
}

...
if(h==0)
    h=TOL;
    x[j+2]=temp+h;    //Reduces floating-point error
h=x[j+2]-temp;
double[] funcvalues2=function(x);
x[j+2]=temp;
for(int i=0;i<(n-3);i++)
    jac[i][j]=(funcvalues2[i]-funcvalues[i])/h;
}
return jac;
}
private double[] function(double[] x)
{
    double[] f = new double[n-3];
    SchwarzFunction denorm = new SchwarzFunction(x,angle,0);
    GaussQuad gq = new GaussQuad(denorm, denorm.alpha(), denorm.beta(), denorm.a(), denorm.b());
    double den = Math.abs(gq.integrate(GQN));
    double den1 = vertex[1].subtract(vertex[0]).modulus();
    for(int i=0;i<(n-3);i++)
    {
        SchwarzFunction numer = new SchwarzFunction(x,angle,i+1);
        gq = new GaussQuad(numer, numer.alpha(), numer.beta(), numer.a(), numer.b());
        double num = Math.abs(gq.integrate(GQN));
        double num1 = vertex[i+2].subtract(vertex[i+1]).modulus();
        f[i]=num/den-num1/den1;
    }
    return f;
}

References


Projects

7th Period
Implementation of Image Deblurring Techniques in Java

Peter Chapman
Computer Systems Lab 2007-2008
Thomas Jefferson High School for Science and Technology
Alexandria, Virginia

Abstract

Families, friends, professionals, and enthusiasts take countless numbers of photographs every day, and inevitably, many images suffer from some sort of "blurring." A program with the power to take a blurred image and create a much crisper and clearer "deblurred" form would be immensely valuable. Law enforcement attempting to read the license plate off a blurred photo, or a family attempting to improve the clarity of their grandfather’s smile would find such a piece of software useful. In my implementation, I attempt to deblur images suffering from simple types of motion blur using the alternate domains granted by the use of Fourier transformations and a basic understanding of image deconvolution.

Keywords: Fourier Transformations, Spatial Domain, Frequency Domain, Phase Domain, Blind Image Deconvolution, Fast Fourier Transformation, Cooley-Turkey Fast Fourier Transformation Algorithm, Discrete Fourier Transformation, Image Deblurring

1 Introduction

Photographs are utilized in many different fields for a wide variety of purposes; Regardless of the subject area, a blurred image is often a useless one. A program with the ability reverse the such damages would be extremely useful. Such functionality could be bundled into the software of cameras with adjustments performed automatically after each shot, or an available feature on standard photo manipulation software. Due to the complexities involved in the image deblurring process my research is focused on blind image deconvolution, where the application is given a general overview of how the image was blurred presumably by the user. In order to further simplify the project further, my application will only be built to handle images suffering from motion blur.

2 Background

Due to the value of a program that can deblur images, many have tried to create an all-purpose deblurring program, but few have found much success with a general approach. The tendency in the field is to focus on motion blur and narrow the application of the program in order to get a more effective method that often applies to a smaller range of tasks. In one such project, the researchers used a modified camera with motion-sensing technology. Upon taking a picture, the researchers were able to read the data collected from the motion sensors and calculate how the image was blurred (Raskar, Agrawal and Tumblin). The results were phenomenal (Figures 1 and 2).

However, it is possible to have some success with more general applications such as that found in the work of M. D. Cahill. His program, called Unshake, attempts to reverse any type of motion blur. Although the application works effectively on relatively minor motion blurring, such as those less than ten pixels, the general solution, however, simply cannot handle blurs as severe as more specialized programs can.

Another paper, Image Deblurring with Blurred/Noisy Image Pairs, conquers blurred images by taking two photographs. The first has a very low exposure, resulting in a dark, noisy photograph with close to zero blurring. The second is a long exposure photo that gets the color and...
brightness in the image with lot of motion blurring. The software developed by the team combines the two by performing image deconvolution techniques on the blurred image using the short exposure photo as a reference that reveals how the high exposure image was blurred. With excellent results, Yuan, Sun, and their colleges plan to implement their findings in video cameras.

Figure 2: Deblurred photograph (Raskar, Agrawal and Tumblin).

In order to reverse the blur on an image, it is necessary to approach the task mathematically. If the process that blurs the image is considered a mathematical function, it must be reversed in order to restore the image; however, to do so, it is necessary to understand how the image was blurred, characteristics such as direction, type (motion, out of focus image, etc.), and magnitude. The best way to approach such a complex task is to convert the image into a different domain. The way in which we normally view images is known as the spatial domain, but if the image is converted into a series of sin functions through a mathematical technique known as a Fourier transformation (Figure 3) it is possible to view the image in the frequency domain (Gonzalez and Wintuz). Once in the frequency domain, it is now possible to perform advanced analysis and mathematic operations on the image in a generalized fashion. It is understood that using the Fourier transformation of an normal image and the Fourier transformation of the blur (a five pixel horizontal line corresponds to a five pixel blur) with a process known broadly as image convolution the Fourier transformation of the blurred image is produced (Figure 4). Thus, by performing the inverse, a deconvolution on the image, the original image can be restored. The most difficult part of this process is determining what the blur factor was when the picture was taken. In theory, if one can determine how the image was blurred, it is possible to deblur the image (Cahill).

3 Rendering the Fourier Transformation

The first step is to render the blurred image in the frequency domain. This is accomplished using a Fourier transformation. The general formula for a Fourier transformation involves integration of a continuous function. Since an image can seldom be represented as a continuous function, it is necessary to treat the image as a set of values in a limited domain. Using the formula for the discrete Fourier transformation (Figure 3) it is possible to view the image in the frequency domain (Gonzalez and Wintuz). Once in the frequency domain, it is now possible to perform advanced analysis and mathematic operations on the image in a generalized fashion. It is understood that using the Fourier transformation of an normal image and the Fourier transformation of the blur (a five pixel horizontal line corresponds to a five pixel blur) with a process known broadly as image convolution the Fourier transformation of the blurred image is produced (Figure 4). Thus, by performing the inverse, a deconvolution on the image, the original image can be restored. The most difficult part of this process is determining what the blur factor was when the picture was taken. In theory, if one can determine how the image was blurred, it is possible to deblur the image (Cahill).

$$F(u,v) = \frac{1}{NM} \sum_{x=0}^{N-1} \sum_{y=0}^{M-1} f(x,y)e^{-2\pi i \left(\frac{xu}{N} + \frac{yu}{M}\right)}$$

Figure 3: Equation for a two-dimensional Discrete Fourier transformation (Bracewell).

Figure 4: The blurring process with images taken from Cahill.
6). (Jones)

\[ X(k) = \sum_{n=0}^{N-1} x(n)e^{-\frac{2\pi ink}{N}} \]

\[ = \sum_{n=0}^{N-1} x(2n)e^{-\frac{2\pi i(2n+1)k}{N}} + \sum_{n=0}^{N-1} x(2n+1)e^{-\frac{2\pi i(2n+1)k}{N}} \]

\[ = \sum_{n=0}^{N-1} x(2n)e^{-\frac{2\pi i nk}{N}} + e^{\frac{2\pi i k}{N}} \sum_{n=0}^{N-1} x(2n+1)e^{-\frac{2\pi i nk}{N}} \]

\[ = \text{DFT}_2[x(0), x(2), \ldots, x(N-2)] + W_N^{-1} \text{DFT}_2[x(1), x(3), \ldots, x(N-1)] \]

Figure 5: The derivation of the fast Fourier transformation, taken from Jones.

Figure 6: Chart illustrating the increased efficiency provided by the fast Fourier transformation, taken from Jones.

The inverse of the FFT, a step necessary for returning the deblurred image back to the spatial domain, is easily performed by essentially taking the conjugate of the image in the frequency domain, realizing that the data resulting from the FFT is a series of complex numbers; then performing a FFT; and finally calculating the conjugate once again. The product of the entire process results in a significant level of noise for which must be compensated.

References


Abstract

This project is a decompiler capable of processing outputted Java bytecode into fully-recompilable and functionally-equivalent source code. Several people could benefit from being able to decompile source code, including potentially large companies.

Keywords: decompiler, static analysis, reverse engineering

1 Introduction

It is very commonly asked if it is possible to get working source code from the executable binaries, and the most common replies are along the lines of “It’s impossible,” often including references to turning hamburgers to cows or similar. These replies are, simply put, false in every way. Compiling is not anything like turning cows to hamburgers. The latter process involves ripping out large portions of the cow and recombining much of the rest, in essence, fundamentally changing every aspect of the cow. Compiling, instead, only discards structural information and rewrites the rest in a way that a processor can understand better. Executable code is essentially source code without the structure.

The primary problem with decompiling is that the problem tends to be ill-defined. Executable binaries represent the main input, the other inputs being auxiliary information that can help the decompiler or mere stylistic guidelines. But the output is hard to define. Three definitions quickly come to mind: the original source code used to compile the binary; a source code that will, fed into the original compiler with the same options, produce the same binary code; and source code, when fed into a compiler, will produce not necessarily identical but functionally equivalent source code. The first one is obviously impossible, the second one (surprisingly) is often unfeasible or undesirable, so the latter is what most people focus on.

This project’s primary goal is to be able to produce fully recompilable Java code given the class files input into a Java virtual machine. The original versions will focus on being able to handle only the output of the most recent Java compiler, Sun Java 6 SE, in unoptimized format. Time permitting, support for handling older versions (mostly limited to the more complex finally handling) will be added. Further improvements to handling optimized code and code not produced from any standard Sun Java compiler will be next, followed by improvements for handling entire JAR or ZIP files at once.

2 Background

In the realm of computer code, there are several layers of code. Amongst the so-called high-level programming languages (like Java, C#, Python, or LISP), there are variations in expressiveness and readability. These variations make ranking the languages in any hierarchal order next to impossible. Java, for example, as string capabilities that far outreach those of LISP, but LISP is more effective at dynamic interpretation than Java. However, it is possible to classify these languages by levels of expressiveness—loosely defined, how much the language impedes certain tasks. This same metric can be applied to lower-level languages and even executable code representations.

In this hierarchy of languages by expressiveness, the most expressive languages would be machine code as executed by a native processor, such as Intel’s i686 instruction set or the instructions for a MIPS processor. Corresponding almost exclusively one-to-one with these ‘languages’ are the respective assembly languages, mostly a series of mnemonics for the actual instructions (although certain operations are prohibited in the assembly language that are permitted in the machine code). Slightly less expressive are various types of portable assembly, like GCC’s RTL. Older languages like C or FORTRAN are the next level, representing easier representations of the same information with a thin veil of type-checking. In the next tier are bytecode languages, like that of Java or Python, which retain significantly more structure and
have much more intensive sanity checks. In the top tier lies many of the modern languages, with complex features like static type-checking or stack-unrolling exception handling.

This hierarchy is generally collapsed into four segments: machine code, bytecode, \(^{2}\) assembly, and source code. Transforming code between these various classes has different names, specified in Table 1. Conversions between source code and assembly are not typically used, so their common names will vary considerably, mostly depending on whether or not the context dictates where assembly falls on the line between machine code and source code. In terms of decompiling, determining the assembly from machine code is much more difficult than the source code from assembly, so these lines are more viewed as source level analyses, albeit more difficult than the more common ones. [2, 3]

Several examples of decompilers exist at the present time. In the early days of Java, several decoders were written that took advantage of the ease of decompiling bytecode, prompting several articles to be written detailing the scope of issue, including fooling decompilers. Most of these early decompilers are helpless at modern code, and several no longer exist. Furthermore, very few decompilers exist for non-Java programs. A search of SourceForge revealed one Flash decompiler, a Python decompiler, one C decompiler incapable of decompiling even simple code (although it is innovative in its usage), Boomerang (another decompiler), and several defunct Java decompilers. Aggregating all together, there are currently only four decompilers of note:

- Jad, the best Java decompiler currently out there (although closed source and written in C).
- Boomerang,[1] the best open-source C decompiler and the only one easily obtainable.
- JODE,[4] the best open-source Java decompiler, but seems to be more- or- less abandonware.
- Hex-Rays, a decompiler that plugs into the popular IDA program. Closed source, expensive,

<table>
<thead>
<tr>
<th>Machine code</th>
<th>Assembly</th>
</tr>
</thead>
<tbody>
<tr>
<td>Porting</td>
<td>Disassembling</td>
</tr>
<tr>
<td>Assembling</td>
<td>Porting</td>
</tr>
<tr>
<td>Compiling</td>
<td>Depends on author</td>
</tr>
</tbody>
</table>

Table 1: Nomenclature of various transformations

The first stage of the decompiler is to parse the incoming files. Most of the internal representations are handled in the info package. Internally, the class files are handled through a service architecture: a central class, ClassPool, contains a pool that manages the various known classes. Class files are given to this pool by registering various ClassSource interfaces that can produce an input stream for a requested class. The main shell registers a source based on the files passed into the command line arguments and then proceeds to find outputs for all of these classes, by requesting fully-decompiled versions from ClassPool.

Whenever a class (internally represented using ClassInfo references) is requested, a level of decompilation is requested. If the class has not been handled yet, the class pool grabs the input stream and starts parsing it to the required level. If it has been handled, the internal decompilation level is compared to the requested level and proceeds until the requested level is reached. For all but the last two levels (PROCESSED and FILTERED), the stream is parsed to the given point. The possible levels to parse to are the header information, constant pool, class metadata, fields, methods, and annotations; all parsing is done by the ClassParser class, which has intimate access to the internals of ClassInfo.

Should the parsing run into any problem that violates the Java VM specification, it will try to continue whenever possible, logging a verification error. Examples of these errors are mismatched magic numbers, illegal flags, and improper versioning. Should continuation prove impossible, the decompiler stops attempting to parse the class, printing out an error. An example would be illegal constant pool tags; other examples include I/O errors.

When the processing stage is reached, the input stream is discarded to conserve memory and the class then focuses on trying to make sense out of attributes. Signatures are attached to methods and fields at this stage, and code is actually physically decompiled here. Signatures are not parsed here, but are lazily evaluated when the class is being printed.

The first prototypes of the decompiler ignored the Code attribute (where all the instructions are actually stored) and focused on printing out the full signatures, including generics. These signatures are decoded through the hand-written SignatureParser

\(^{2}\) In most circumstances, bytecode and machine code are considered identical classes. The distinction is only important when classifying difficulties between various transformations.
class. This class is currently not optimized for speed; it switches from using a `StringBuilder` to using a `String` several times, a process which can incur very large overhead costs. This class has five entry points; two for the internal field and method types and three for the stored generic signatures. It is also capable of returning full generic signatures, as example 1 shows. The string on the bottom is the actual signature stored in the class file while the above code is the returned output of the decompiler.

```java
abstract class GenericsTest {
    public abstract <T extends java.lang.Object, E extends java.lang.Throwable> T foobar(T var_0) throws E;
}
```

Example 1: Generic method example

Later prototypes developed the full capability to correctly reproduce annotations that were retained in the class file. Annotations have some confusing rules: in particular, an annotation element cannot have a type of `Object` or multidimensional arrays. The decompiler can decompile both the declaration of an annotation type and the use of an annotation on classes, methods, method parameters, and fields. Annotations in other places are not passed down to the source code, so their proper decompilation is impossible.

### 4 Static Stack Analysis

The Java VM specification specifies a total of 202 opcodes, as well as reserving three more (a breakpoint and two implementation-defined opcodes). Of this list of 202, 6 are not used in the Java 5 or Java 6 compiler. The opcodes `jsr` and `ret` were deprecated because of their incompatibility with the new stack-frame verification; `goto.w` and `jsr.w` are 4-byte instructions that are unused since code is limited to 2-bytes; `nop` is unused for obvious reasons; and, finally, `invokedynamic` was created for the ease of dynamic programming languages, and its semantics are not concisely represented in Java syntax.

The Java VM is ultimately stack-based. An array of local variables is maintained, and operands are pushed onto or off of the stack by the various operators. Other bytecode-interpreted languages have similar semantics, but most modern processors are register-based, where the operator declares which registers it operates on. This stack makes decompilation easier, since the stack starts empty and is required to end empty, unlike real processors where the registers may be used as arguments or return parameters.

Of the previously mentioned 202 opcodes, many are shortcuts for what could be larger opcodes. `iload_1`, for example, is semantically identical to `iload 1`, both of which are very close to `aload 1` or even `istore 1`. Taking advantage of these similarities, the opcodes are simplified into the following 22 classes:

- `LoadConstantInstruction`: Push a constant value onto the stack
- `LoadStoreInstruction`: Push a local variable to the stack or pop the stack to a local variable
- `ArrayLoadStoreInstruction`: Store or load a variable to an array
- `PopInstruction`: Pops one or two values from the stack
- `DupInstruction`: Duplicates potentially several values on the stack
- `SwapInstruction`: Swaps the top two values on the stack
- `ArithmeticInstruction`: Performs an arithmetic operation on the top one or two variables on the stack
- `CompareInstruction`: Performs a comparison on the top two variables on the stack
- `IncrementInstruction`: Increments an integer by a specified count
- `PrimitiveConversionInstruction`: Converts between the primitive types
- `IfInstruction`: Conditionally jumps based on a variety of conditions
- `GotoInstruction`: Unconditionally jumps
- `JSRInstruction`: Unconditionally jumps, but pushes the current address on the stack
- `RetInstruction`: Jumps to the address popped from the stack
- `SwitchInstruction`: Executes a switch instruction
- `ReturnInstruction`: Returns control from the function
FieldAccessInstruction  Accesses a field of a class
InvokeInstruction  Invokes a method
NewInstruction  Instantiates a new class
ArrayNewInstruction  Instantiates a new array
ArrayLength  Pushes the length of the array onto the stack
ThrowInstruction  Throws an exception
CastInstruction  Checks the type of a class
MonitorInstruction  Enters or exits a synchronization monitor

Every opcode, except nop, goto, and goto.w, involves operating on the stack. The first stage of decompilation is to therefore analyze the stack. Analysis of the stack involves determining a few components: ensuring that the VM types (integer, float, long, double, and object) are proper at all times, construction of a few invariants for later analysis, and the determination of variable typing and scope. All three operations are carried out simultaneously in the first analytic pass through the bytecode (the initial construction technically counts as a pass, although the only analysis it performs is the insertion of pseudo-bytecodes and the removal of GotoInstruction. The first operation—VM typing—is trivial and carried out normally as part of the VM's bytecode analysis; it is the last two operations that present the problems.

The main invariant that is needed for further decompilation is the invariant that any instruction with two parent nodes should have an empty stack. This requirement makes the decompilation of many of the opcodes simpler by ensuring that the operand can only be one thing and not several. For the most part, Java code already satisfies this assumption, with one exception: the ternary operator. Since the bytecode is more liberal in this aspect than Java permits, some sort of heuristic needs to be used to determine whether or not this branch pair is a conditional expression or an optimized branch.

4.1 Variable Analysis

Variable typing and scoping is the most difficult aspect of decompiling, and it is a problem shared by all types of decompilers, be they native-code decompilers or bytecode decompilers. An interesting aspect is that if either type or scope is known, the other becomes simple to find, whereas finding both together is much harder.

The harder problem by far is scoping. To see its difficulty, one needs to learn a little about the structure of the Sun Java compiler. Each variable is mapped to one spot in the local array buffer, or two if it is a long or double. This spot is reserved for it from the point of declaration to the end of its scope. The naïve implementation would be to rely on the scope of control structures to dictate variable scope, but this can lead to some over-scoping of variables. Even worse is the fact that Java allows the introduction of arbitrary variable scope. Therefore, a decompiler cannot rely solely on control structures even if code only compiled from the Sun Java compiler was input (there is one simplifying caveat: the argument variables’ scope is that of the entire program, so the Sun Java compiler will never reuse these slots).

The solution to variable scoping is by using a form known as SSA, or static single assignment. More commonly known for its use in compilers, SSA form is a modified form of code where each variable is limited to only one assignment.

Currently, only a mild portion of SSA is formed. The stack analyzer is incapable of handling branches, and control structures with it.

5 Post-Decompilation Transformations

Surprisingly, much of Java since Java 1.0 is essentially a hack in terms of the bytecode. The only real changes to the bytecode are the now-allowed use of class in the ldc instruction, a slight change in the lookup of the nonvirtual method lookup, and the deprecation of the jsr, jsr.w, and ret instructions due to the difficulty of verifying them under the new Java 6 stack verification model. Everything since then—including, but not limited to, generics and inner classes—is merely a compile-time hack, see example 2 for a partial comparison of a case involving enums and inner classes.

These advanced constructs can be detected by simple analyses. Each synthetic method is not reproduced to output if its existence can be explained and rectified. So far, no explanations or rectifications are performed.

References


[2] Emmerik, Mike Van. “PhD Confirmation Report: Type Inference Based De-
public enum EnumTest {
    A, B;
    private static int foo = 5;
    static class Bar {
        public String toString() {
            return Integer.toString(5);
        }
    }
}

public class EnumTest {
    private EnumTest A, B;
    private static int foo = 5;
    public static int access$100() {
        return foo;
    }
}
public class EnumTest$Bar {
    static class Bar {
        public String toString() {
            return Integer.toString(EnumTest.access$100());
        }
    }
}

Example 2: Simplified example of how some constructs are handled internally


Simulation of the Spread of a Virus Using Agent Based Modeling

Matt Wade

Abstract

My goal is to make an agent based modeling simulation that shows the spread of a cold through a school. It will start with an amount of infected students and healthy students received as inputs and will show how much the virus spreads or possibly recedes over time.

1 Introduction

My program will answer the question as to how quickly and fully different types of sicknesses will be able to spread through the population of a school once introduced.

This will show how likely it is for a disease to be spread by a set amount of sick people coming to school with the sickness. This will show if the danger of infecting others is actually a valid excuse not to come to school or if you should come to school unless you actually don’t feel like you will be able to do work.

I expect my program to show how many people are infected with a virus after any amount of time given an input of how many people there are and how many are infected at the start. It will display the number of infected people and the period that the agents are in at the moment in the display window plus possibly some other information such as the ratio of infected to healthy people, the total number of people that have been infected (including those that have gotten healthy after being infected), and other similar results.

When finished the program will display values for the number of agents are sick, healthy, total agents, total infections, and total recoveries. It will also have a display showing the locations of all of the agents and their status (healthy or sick). Eventually it will be able to have different diseases which have different recovery times and infection rates. I will use research data for all of the stats relating to the different diseases such as the recovery time and the infection rates based on real values for different diseases.

If I finish all of this early I could move on to possibly exploring different methods of contracting the diseases. At the moment viruses are all passed through close contact with a sick agent, but I could add diseases passed through contact with where a sick person had recently touched such as a doorknob or a keyboard.

2 Background

I have been researching different examples of agent based modeling using MASON through the MASON website and the book Growing Artificial Societies which specifically discusses the Sugarscape model. A model similar to mine is part of the demo models for MASON which describes the spread of a virus through a workplace. In this example the agents become immune to the virus once they become healthy, they stay sick for a set amount of time, and move to work and come back home without moving to other places. They are infected through close contact with sick agents just like in my program and this program only has one type of disease unlike mine will when finished. I will attempt to make my model more realistic in most of these areas by using actual data. There are also many papers documenting the usefulness of computer models and simulations in science such as Tutorial on Agent-Based Modeling and Simulation by Cayzer and Sullivan, and Modelling Danger and Anergy in Artificial Immune Systems by Macal and North, which is another paper documenting the use of a computer model in analyzing the human immune system. References: Macal, C., and North, M. (2005). Tutorial on Agent-Based Modeling and Simulation; Cayzer, S., and Sullivan, J. Modelling Danger and Anergy in Artificial Immune Systems; Macal, C., and North, M. (2005). Tutorial on Agent-Based Modeling and Simulation Part 2: How to Model with Agents; Epstein, J. (1996). Growing Artificial Societies: Social Science from the Bottom Up. Brookings Institution Press.

3 Development

My program has three main classes. An Agent class which defines what values an Agent will store and how to construct it. A class that creates the GUI.
And the most complicated of the three, the Model class which is where all of the calculations occur. In the Model class there is an arraylist containing all of the Agents in the simulation. With these Agents the class has to define a step() function which moves the simulation forward. This function has to update the locations of all of the agents, check to see if any of them get infected, and check to see if any recover from being sick. First it goes through the list of Agents and moves them all to the next location in their schedule. In order to check for any new infections it goes through the entire list of Agents finding each sick Agent. Whenever it finds a sick Agent it finds any healthy Agents in the same location and checks a randomly generated number against the sick Agent’s infectiousness value. If the random number is lower then the healthy Agent is switched to sick and the method continues on through the rest of the list. To check if any Agents recover from sickness it goes through the list checking the recoverytime value and if it equals zero the Agent is switched to healthy.

4 Results

At the moment my program is pretty much in its final state in terms of the actual simulation. All of the methods relating to the simulation, such as step(), checkinfection(), or checkrecovery(), are all completed and working as they should. This means that the results I am getting right now are pretty much the same results I will have when the whole project is completed except for any analysis tools that I plan to add to the program, such as a picture showing the locations of all of the Agents, and a graph that shows the number of sick and healthy Agents over time. As of now though, it will give you data on the number of sick and healthy agents, the total number of infections and recoveries, the number of steps taken, and the locations of all the agents.
Abstract

The main purpose of this program is to accurately simulate the genetic evolution of a species. It will attempt to do so using methods such as genetic mutation, genetic drift, and natural selection by means of both microevolution and macroevolution.

Keywords: genetic algorithms (methods of representation, methods of selection, methods of change), evolutionary computation, genetic mutation, genetic drift, optimal state, microevolution, macroevolution, Darwinism, natural selection, genetic variation, recombination, gene flow, speciation, (if time)

1 Introduction

1.1 Rationale

The computer will simulate an environment and the user can modify that environment. Modification of an organism environment can force it to adapt and in essence, better survive. Because of this, those that are not cut out for the change will die off and those that are will live on and reproduce, thus creating a genetic drift in the species. This is the theory. According to this theory, one should be able to predict the changes in a species genetic make-up due to a change in its environment. If the environment becomes hotter, those creatures with higher temperature tolerance should be less affected than those without a high temperature tolerance and so one would expect to see the species evolve to have a greater tolerance to higher temperatures.

1.2 Purpose

Evolution often thought of as the changes that occur in an organism to better suit them to their environment. However, this is not completely true. Evolution occurs in both positive and negative directions. It is completely random and the result could be in favor of the organism or it might not. I am trying to simulate evolution and track the change in a species’ traits to better understand how evolution really works.

2 Background

"Genetic changes do not anticipate a species' needs and those changes may be unrelated to the selection pressures on the species. Nevertheless, evolution is not a fundamentally random process." [3]

Agent-Based Modeling

The actual evolution simulator is an ABMS with the Organism and Predator classes being the 'agents.' An agent is "autonomous and self-directed." It can “function independently in its environment and initiates dealings with other agents.” Mostly, an ABMS focuses on the interactions between the agents. In this project, I will be observing both the interactions between the agents and the interaction between the agents and their environment. [8]

Basic Concepts

A population of any given species is greatly affected by its environment. This is where an animal will get its food and raise its young. In order to do this, it has to be well adapted to the environment it lives, yet also able to change under stress (such as a change from the norm). This is when evolution will occur. The members of a species that are best able to handle stress are the ones that will live on to populate the species; therefore, their young will acquire the “better” traits and be able to live in the newly changed environment. The environment in which a population lives provides resources for the population such as food and shelter. If there is limited food, then the environment will only be able to support a given number of species, meaning that the population will have a max value. The function of the population over time should be logarithmic, approaching that max value. However, this
is just a basic model of an environment, void of predators and many other factors that affect the size of the population. If there are predators, then the population size should oscillate in accordance with the predators (though there is a slight lag in the predator’s population graph). [7]

1. Mechanisms that Decrease Genetic Variation[1]

(a) Natural Selection
Natural selection was introduced by Charles Darwin. It is when the frequency of the more prolific members of a species increases because they are better adapted to the environment.

(b) Genetic Drift
This occurs when the allele frequency changes (can allow mutant alleles drift into fixation).

2. Mechanisms that Increase Genetic Variation[1]

(a) Genetic Mutation
This occurs when the gene sequence altered because the copy of "DNA" is corrupt.

(b) Recombination
This includes crossover of genes from the mother and the father to produce genes of the child (gene shuffling).

(c) Gene Flow
Gene flow occurs when genes drift into a population from a different population through mating.

3. Types of Evolution[1]

(a) Macroevolution
Macroevolution includes speciation, or the separation of one species into two. It is an evolutionary change at or above the species.

(b) Microevolution
Microevolution is evolutionary change below the species level.

4. Types of Genetic Variation[4]

(a) Variation Under Domestication
(b) Variation Under Nature

3 Development Sections

3.1 Requirements, Overview, Limitations, Development Plan

The minimum requirement for this project would be an environment that includes the species under evaluation. There obviously must be a species to study and that species must have traits that can be altered, affecting its ability to survive. The environment must also provide a regenerating food source for the organism being studied else it will not live long enough to produce various generations leading to evolutionary results.

The project’s main limitation is time. I am not sure how far involved my program will be because it really depends on how many problems I am faced with while creating the simulation and how easy they can be resolved. If I do manage to get pretty far to the point I want to include gene flow and speciation, I may find myself limited by the processing speed of the computer. This may also be a problem in the earlier stages if the population of a Species becomes too large and the time taken to calculate each Species' actions becomes too much to handle.

Software

1. Java
2. Java3D

3.2 Research Theory and Design Criteria

Methods

1. Food Methods

- breed()
The Food class has a simple breed method because it is asexual and does not have a limitation on the number of times it is able to breed in one lifetime. Food breeds at random, but only if there is room for offspring around it.

- act()
The act method merely consists of randomly determining whether a Food will breed and/or whether it will die.

- die()
The die method acts randomly, generating a number between one and zero. If that number is below the given deathRate of the
1. Food, it will die and the World will remove it from the Species HashMap.

2. Organism/Predator Methods

- **act()**
  This method determines the next action of the Organism or Predator depending on its age. For example, younger and older Species will not be able to breed and will focus more on eating.

- **move(Location loc)**
  This method moves an Organism or Predator one step closer to their target (which may be food or prey).

- **breed()**
  This method calls the crossOver method to determine the genetic code of the child with respect to that of its parents. If there is no space to breed or the Species is too young or too old, it will not have access to this method.

- **crossOver(int trait1, int trait2)**
  This method takes a trait from the father and a trait from the mother and produces a trait that is equal to one or between the two. If it is mutated, it might not reflect the traits of either the mother or the father.

- **die()**
  Same as Food’s die().

3. World Methods

- **step()**
  This method calls act() on each of the Species located in the World (stored in a giant HashMap).

- **getClosest(String type, Location loc)**
  This method return the closest specified Species to the Location given. This is useful when an Organism is searching out food, or a Predator is hunting an Organism. It goes through the HashMap of all the Species in the world, comparing the distances from the Location given and returning the specified Species with the shortest distance from the given Location.

- **add(Species s)**
  This adds a Species to the World and is called by the breed() method in each Species when it gives birth to a child. add(Species s) also tallies the amount of each Species that is present in the World, so when a Species is added, the number count for the added Species (Food, Organism, Predator) is increased by one.

- **remove(Species s)**
  Similar to add(Species s), this method removes the given Species from the World and is called by the die() method of each Species. It also subtracts the count of the given Species by one.

- **aggAve()**
  This method calculates the average aggression values of the Predators living in the World.

- **defAve()**
  This method calculates the average defense values of the Organisms living in the World.

4. Sim Methods

- **actionPerformed(ActionEvent e)**
  This method is called repetitively by a Timer that is fired faster than the eye can see. This method calls step() on the World and then draws every Species, specific to their type, giving the simulation animation.

Algorithms

1. Process for Recombination

   The process for creating a new organisms with a new combination of genes mixed from its parents (and sometimes randomly mutated) takes the traits from both parents and gives the child a trait that is either equal to one of the parents, or is a mix of the two (something in between). The assignment of the trait is semi-random.

2. Randomization for Mutation

   The process by which genes are mutated is completely random. In fact, it is double random because the swapping of genes is random and the chance that it is mutated is also random.

3.3 Testing and Analysis

   There have been many tests done to the program because it is so fragile and susceptible to errors. Currently, I have been testing the functionality of the program through system outputs. This includes running the program with only Organism and few Food and individually tracking each one to make sure they are working correctly. Then, I run the program again, but with many Organisms and Foods keeping track of
the average values of traits to see if they are changing or remaining the same. If they are the same, the program is not working properly and if they are changing by too great of a degree or too rapidly, there is also a problem.

3.4 Visual Representation of Data and Results
Currently, I have not been able to produce graphs and charts to display the information from the simulation because I am not far enough in my program to do so. I just finished getting a semi-working model up and running and when the model is stable, I should be able to make graphs and charts that are updated by the program itself. This is most efficient because the program has to the information in the simulation.

3.5 Development Procedures
Steps to Simulating Evolution
1. Create a changing environment with which a species may interact
2. Create a food source for the species
3. Create a species with designated traits to be tracked
4. Possibly create an herbivorous species and a predator
5. Define how the species may evolve (genetic algorithms)
6. Track the changes in traits and make observations
7. Adjust the model until a balance is achieved

4 Results, Discussion, Conclusion, Recommendations

With the addition of the JSlider bars, the simulation will run a lot smoother than it was previously due to the increased stability of user input to regulate the breedRate and deathRate variables of each of the species. Because of this stability, it is easier to recognize trends in the populations when certain variables are altered. For example, when the Predator aggression’s average is one point above the average defense levels of the Organisms, the Predator population seems to be more stable and is increasingly so the more it rises above the Organism’s defense. However, this negatively affects the Organism’s stability and thus the system tries to find a stable point. However, in my simulator the Predators always die out first, and then the system becomes more stable with just the Plants and Organisms. Because of this, I think I will focus more on the Organism class and its reactions with the envirorment than with the Predator class. Also, when the reproduction rate of Food is higher, the stability of the system improves.

5 Literature Cited

References