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

TJHSST

May 9, 2008

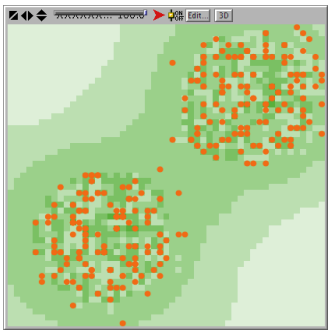
Projects
3rd Period

Exploring Artificial Societies Through Sugarscape

Jordan Albright
TJHSST Computer Systems Lab 2007-2008

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, or more complicated systems, such as self-segregating neighborhoods. The systems 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.



Background and Introduction

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 which are guided by rules directed at individual interactions rather than the group. One common simulation using agent based modeling is sugarscape, which is comprised of a set of agents who make calculated moves through a sugarscape a landscape that varies in the amount of sugar, a renewable source of energy for the agents, available at each square in the grid.

Methods and Procedures

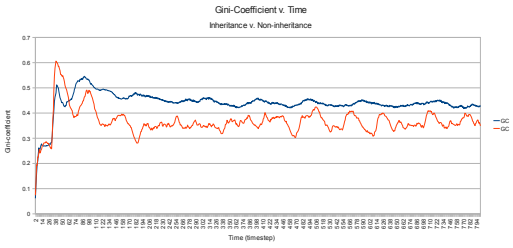
The Sugarscape agents behaviors are specified by a set of guidelines. One of these guidelines involves searching for food: in each timestep, each agent determines which patch or patches of the Sugarscape would be the best place to move. This is done within each agents scope of vision, a number specified by the user (usually between 1 and 10 patches). 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.

At each timestep, the agent may also reproduce. The user may choose what attributes of the parent agent will be inherited by its offspring. There is a switch that allows for the inheritance of vision and metabolism.

At each timestep, the agents may also die.

Each timestep, the amount of sugar in the patches adjusts to reflect the consumption by the turtles.

The wealth of the agents is analyzed using the Gini coefficient at each timestep.



Results and Conclusions

If metabolism and vision are inherited, the Gini coefficient varies by an average of 0.8, with the average Gini coeffienct over 800 timesteps of the non-inheriance simulation at 0.37 and the average Gini coefficient over 800 timesteps of the inheritance simulation at 0.44. This reflects a much greater inequality when the agents are able to inherit the ”genes”–good or bad–of their parent agents. It is important to note, however, that the wealth distribution during inheritance simulations is much more stable than the wealth distribution of the non-inheritance simulations.

Figure 1: The simulation system used to animate physically simulated characters. The locomotion controller obtains a desired velocity from the navigation controller and computes the joint positions that will achieve it. The desired joint positions are passed to the joint controller, where joint torques are computed to eliminate errors in joint position. The joint torques are used by the numerical integrator to compute new positions for all the character's body parts.

CAPTCHA Solving With Neural Networks

Tianhui Cai
TJHSST Computer Systems Lab 2007-2008

Abstract

CAPTCHAs, Completely Automated Public Turing tests to tell Computers and Humans Apart, are tests to determine if a user is a human or a machine. Variations include audio and visual CAPTCHAs, which are often found on registration webpages to prevent automated (spam) registration. The focus of this project is on visual CAPTCHAs, which consist of an image with letters or numbers that are to be typed into a form by the user. The goal of this project is to devise a system to break a particular CAPTCHA.

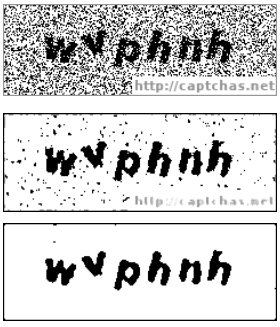
Background

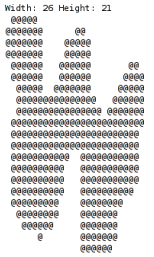
A CAPTCHA's purpose is to distinguish between a computer and a human by presenting a challenge that is easy for most humans, but difficult for computers. A common example is the visual CAPTCHA, which is an image with a series of letters and/or numbers, and a user is prompted to enter its value into a box. The image often contains distortions to make it difficult for a computer to read. These distortions include rotation, translation, scaling, background noise, and color.

For a computer to beat a CAPTCHA, it must identify which pixels comprise the letters. This is usually done after removing the background clutter. After the letters are separated from the image, they must be identified, which is often done with a neural network.

Neural networks model biological neural systems. Although each component is simple, because the entire network is highly connected, neural networks can model highly complex, nonlinear systems and can be proficient in classification and pattern recognition.

Research on neural networks has been in existence for several decades. In particular, the use of neural networks for classification has been used. Le Cun et al at AT and T laboratories has demonstrated that with a particular set of connections with a multi-layered perceptron, handwritten digit recognition can be done extremely efficiently.





Procedure and Methods

The general procedure for this endeavor consists of several steps. The first step is the acquisition of the image, which is done by downloading them from captchas.net. This particular website provides a free CAPTCHA service, with a formula to tell you what an image says. The images were downloaded and named with ruby, with filenames being the sequence of letters depicted in the image.

The second step is to remove background clutter. In this particular case, the CAPTCHAs provided contain a lot of black and white noise, which can be removed with a median filter. In contrast to a Gaussian blur, it does not blur the image, thus saving fine details of the image while removing noise. This is done in Java.

The next step is segmentation – separating out the letters from the background. This must happen after the removal of the background clutter. It is performed in this project using flood-fill. Flood-fill has the advantage that it is simple to code. However, it will not be useful if two letters are conjoined or if a single letter is broken up into multiple parts. In this scenario, these are not significant problems and cases in which this happens are thrown out. This step is also done in Java.

The last step is the identification of the characters that have been segmented. This will be done with a neural network – a three-layer backpropagation neural network. It will first be trained on a training set, so that it learns how to identify characters. Afterwards, it will be run to identify images using what it has learned. A key feature of this neural network must be to save the network into a file, so that it can be loaded and trained multiple times.

Testing

The image processing – noise removal – is tested on sample CAPTCHA images. It works. Saving the neural network works, too.

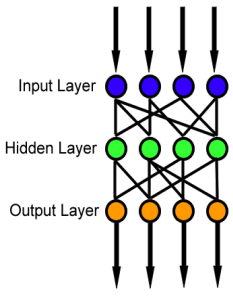
A set of 50 training images have been downloaded with filename as the letters depicted in the images. This works.

The neural network is tested by testing it on simple inputs and outputs, such as AND, OR, and XOR. This works, too.

The next step is to use the outputs of the image processing and segmentation as the inputs to the neural network for training, and then for testing.

Results and Conclusion

This section cannot be completed at the moment.



Excursions into Parallel Programming

By Michael Chen

November 1, 2007

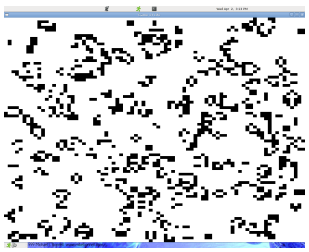
TJHSST Computer Systems Lab 2007-2008

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. MPI (Message Passing Interface) is one of the most crucial and effective ways of programming in parallel, and despite its difficulty to use at times, it has remained the *de facto* standard. 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.

Introduction:

-MPI has high computational power, used in every field for intense calculations (AI, molecular biology, ecosystems)
-Expansions in library to adjust for use with supercomputers
-Efficiency becoming an issue with latency v. processing Power: where is the middle point?



Running simulation of game of life with 9 processors

Sample Code:

```
void move()
{
if (xs>0)
   MPI_Send(arr,arrsize*arrsize,MPI_INT,rank-1,tag,MPI_COMM_WORLD);
if (xs<sqrt(size)-1)
   MPI_Send(arr,arrsize*arrsize,MPI_INT,rank+1,tag,MPI_COMM_WORLD);
if (ys>0)
   MPI_Send(arr,arrsize*arrsize,MPI_INT,rank-
sqrt(size),tag,MPI_COMM_WORLD);
if (ys<sqrt(size)-1)
MPI_Send(arr,arrsize*arrsize,MPI_INT,rank
+sqrt(size),tag,MPI_COMM_WORLD);

//after sending, all computers waiting for other processes
if (xs>0)
{

MPI_Recv(rec,arrsize*arrsize,MPI_INT,rank-1,tag,MPI_COMM_WORLD,&st
atus);
for (x=amount;x>=1;x--)
  for (y=ya;y<ya+xymax;y++)
    arr[xa-amount][y]=rec[xa-amount][y];
}
if (xs<sqrt(size)-1)
{
MPI_Recv(rec,arrsize*arrsize,MPI_INT,rank
+1,tag,MPI_COMM_WORLD,&status);
for(x=amount;x>=1;x--)
  for(y=ya;y<ya+xymax;y++)
arr[xa+xymax+amount-1][y]=rec[xa+xymax+amount-1][y];
}
if (ys>0)
{
   MPI_Recv(rec,arrsize*arrsize,MPI_INT,rank-
sqrt(size),tag,MPI_COMM_WORLD,&status);
for (y=amount;y>=1;y--)
  for (x=xa;x<xa+xymax;x++)
    arr[x][ya-amount]=rec[x][ya-amount];
}
if (ys<sqrt(size)-1)
{
MPI_Recv(rec,arrsize*arrsize,MPI_INT,rank
+sqrt(size),tag,MPI_COMM_WORLD,&status);
for (y=amount;y>=1;y--)
for (x=xa;x<xa+xymax;x++)
  arr[x][ya+xymax+amount-1]=rec[x][ya+xymax+amount-1];
}
}
```

Portion of move() method from the Game of Life

Procedures and Methodology:

-Uses C, C++, and Fortran
-Flexible, not restricted to certain capabilities
-Start with non-mpi code, then convert
-Works very case by case, no general testing program
-Optimization of code depends on a computer's specific latency and its processing power. Depending on which works better, the number of computers best used as well as amount of passing in code used changes.

Results and Conclusions:

-Many real-life applications
  -blocking vs. non-blocking checkpointing
  -supercomputing
-With the game of life, more factors will have to be considered than just number of computers: how often to pass information, and how much information to pass.
-Moving the program from theoretical to the practical
-Latency algorithms for testing possible

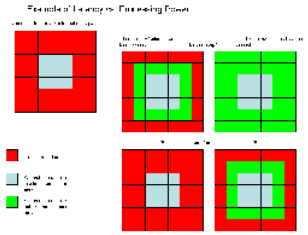


Diagram of latency vs. processing

Exploring Genetic Algorithms Through the Iterative Prisoner's Dilemma

Iterative Prisoner's Dilemma

The general Prisoner's Dilemma is a scenario in which there are two players that can each choose to either cooperate with the other or to defect. They must each make their decisions without knowledge of the other's decision. Each player then gets points based on what their decisions were. The points are given as follows:

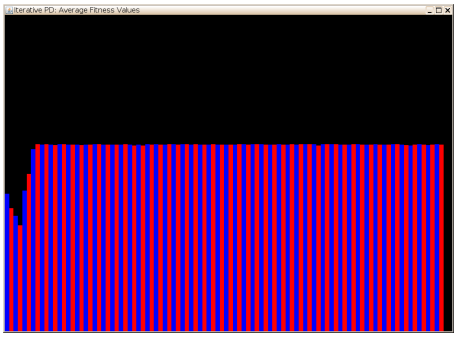
	Cooperate	Defect
Cooperate	R,R	S,T
Defect	T,S	P,P

The point values must satisfy certain inequalities. In order to be a Prisoner's Dilemma problem, the inequality R > T > P > S must be satisfied.

In the Iterative Prisoner's Dilemma, the two players go through this scenario many times, and remember the past. In order to be used for the Iterative Prisoner's Dilemma, the values must satisfy the inequality 2R > S + T. The points from each round are added to form a score for each player. I used the following table, which satisfies both inequalities and is the most commonly used set of values:

	Cooperate	Defect
Cooperate	3,3	0,5
Defect	5,0	1,1

The output of my program is a graph of the average fitness value for each generation (shown below) as well as the numbers represented by this graph in a text file, so that the graph can be recreated after the program is closed. I will use these graphs to determine how many generations the algorithm took to reach the best solution, and compare these among different algorithms.

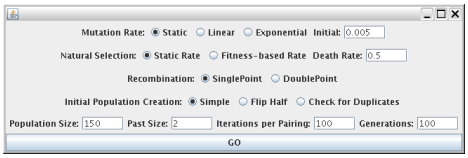


Genetic Algorithm

Genetic algorithms are used to find approximate solutions to optimization problems when the solution would be very time-consuming to compute. The general layout of a genetic algorithm is:

Initialization of gene pool
Loop over generations
    Natural Selection
    Selection
    Loop over empty slots in population
        Recombination
        Mutation

There are many ways in which to perform each of these steps. My program allows the user to select which method to use for each step. In this way, the different methods can be compared.



TJHSST Website Backend Redesign

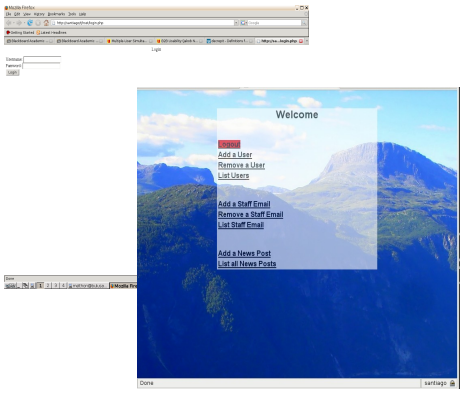
By Martin Elthon

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.

Background

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 LigHTTPD and MySQL 5 for the web and database server respectively, this new site is using the latest web-development technology. Hopefully, through careful documentation and good coding practices, the nightmare of the current site can hopefully be avoided in the future.



Progress:

Shown above is the visual transformation of the web site into its current form. The current features are spell checking, news management, staff email management, and account management. There is also a documentation system called PHPdoc, which creates javadoc-style documentation.

Analysis and Testing

As of right now, there is only one user of this back end 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 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.

Analysis of Runner Biomechanics Through Image Processing

Asa Kusuma
TJHSST Computer Systems 2007-2008

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.

Background
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.

Methods
Using a camera setup behind a treadmill, images of the runner are taken (Fig 2). The second 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 algorithims, 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, called the angle method, 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, the shift 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 (Fig 1). 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.

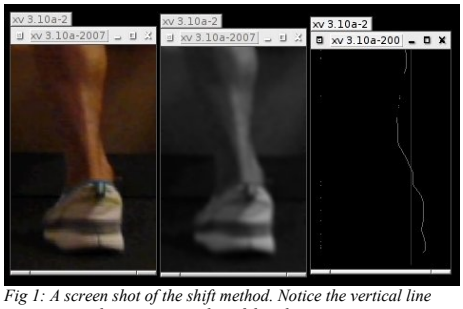


Fig 1: A screen shot of the shift method. Notice the vertical line representing the average x-value of the edge

Fig 2: The camera setup

Particle Swarm Optimization and Social Interactions Between Agents

TJHSST Systems Lab 2007-2008
Kenneth Lee

Abstract

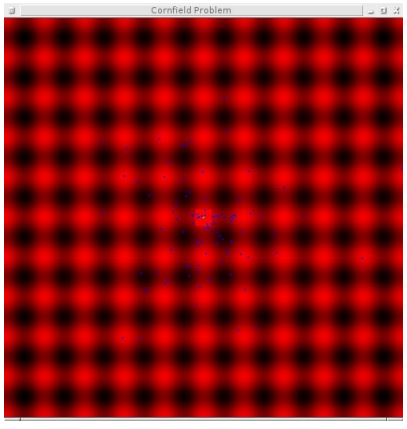
Particle Swarm Optimization is a method of optimization used in n-dimensional infinite search space problems. This project aims to test different social influences and topologies, the way in which the particles communicate with each other in order to find a global minimum, of the particles and determine their ability to converge on a correct solution as opposed to the more common social interaction seen in PSO. The different versions of the social interactions are tested against each other using various benchmark functions based upon iterative cost to run the swarm.

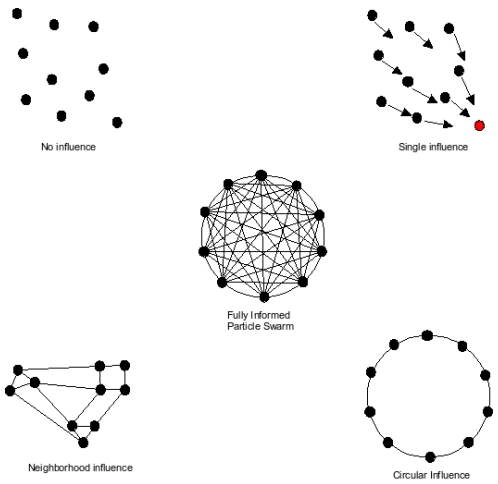
Background

PSO is a relatively new swarm intelligence technique. It was first created in 1995, inspired from flocks of birds and schools of fish. It is considered a good technique because it is both inexpensive in time and in memory.
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 highest fitness value, and wanting to return there.

The social influences are where the particle in influenced based on the other particles, either by their current position and fitness value or their personal best (pbest) fitness value.





Results and Conclusions

The results of the experiment so far are that the effectiveness of a swarm is very dependent on the topology of the swarm as they relate to the Rastrigin function. For instance, though the iterative cost of FIPS is relatively low compared to the iterative cost of RIPS. On the other hand, RIPS is substantially more accurate in determining the correct solution. The answer, as it stands now, seems to be an attempt to bridge the gap between RIPS and FIPS, the low k and high k values. This is seen in DIPS, which while being extremely accurate has one-forth the iterative cost of RIPS.

Though research is not yet complete in this field, it does seem to show promise in continuing to optimize PSO for all real optimization problems. With the addition of more benchmark functions in the proceeding paper, hopefully the results will be more conclusive then at the current junction.

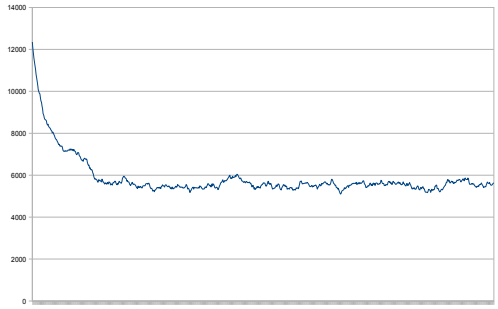
Genetic Algorithms to find Near Optimal Solutions to the Traveling Salesman Problem(TSP)

The Traveling Salesman Problem (TSP) is the classic nondeterministic polynomial-time hard(NP-hard) problem. The problem goes as such Given a number of cities and the costs of traveling from any city to any other city, what is the cheapest round-trip route that visits each city exactly once and then returns to the starting city? Although the problem is stated so simply and discreetly it is in fact a very difficult problem to solve. To simply use brute force to check all possible solutions it would require a O(N!). This quickly becomes very difficult to do even for relatively small n. Therefor it becomes pertinent to find near optimal solutions through other methods using more realistic times.

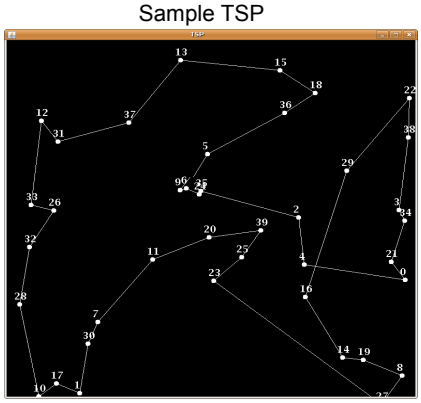
The methods which I use to find near optimal solutions to the TSP are genetic algorithms(GA). GAs are a search technique in computing used for optimization and search problems such as the TSP. GAs are inspired by evolutionary biology and incorporate such concepts as inheritance, mutation, selection, crossover, and reproduction.

Pseudo-code for generic GA
1)Choose initial population
2)Evaluate fitness for each individual in the population
3)Repeat
    1)Select the best individuals to reproduce
    2)Breed new generations using crossover and mutation
    3)Evaluate fitness of the offspring
    4)Replace worst ranked part of population with offspring

Fitness Level Verses Time



Sample TSP



Another method in which I will be employing in order to solve the TSP will be through Ant Colony Optimization. In ant colony optimization I use simulated ants which tour through the search space and leave a pheromone trail. Based on how efficient the trail is more or less of the pheromone is evaporated. Ants then stoicastically chose the various trails in order to find a near optimal solution.

An ant will move from node I to j with probabiliity -

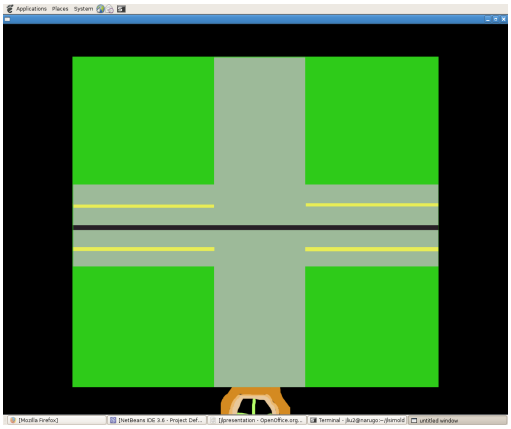
$$p_{i,j} = \frac{[\tau_{ij}]^{\alpha}[\eta_{ij}]^{\beta}}{\Sigma[\tau_{ij}]^{\alpha}[\eta_{ij}]^{\beta}}$$

Psuedo-Code for ACO
```
procedure ACO_MetaHeuristic
  while(not_termination)
    generateSolutions()
    pheromoneUpdate()
    daemonActions()
  end while
end procedure
```

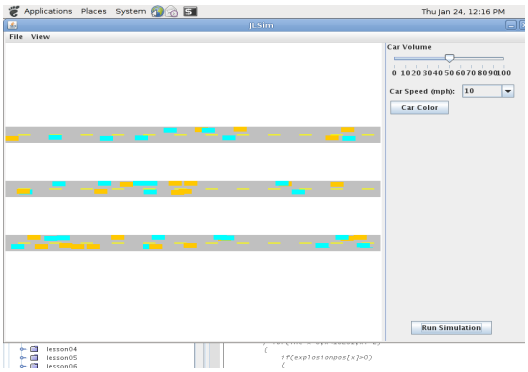
Currently I am using a cycle representation of a solution. This means that I represent each solution as the order in which it attends the cities. This is not very efficient, but it was fairly easy to code.
I am currently using single point mutation and double point crossover. This is how my solutions get better. Eventually I should work for a double point mutation. If I use a matrix encoding the double point crossover will be replaced by a more efficient matrix crossover.
To assess fitness I am currently using the difference between an individual solution and the worst solution in the current gene-pool. This ensures that the worst solution in any given gene-pool does not reproduce. Eventually I want the fitness be placed on an exponential curve in order for the better solutions to reproduce more often.

JLSim: Visual Traffic Simulation Application with Extensive User Interface

Jinyu Liu Pd. 3



Initial Prototype (C/OpenGL [1st Quarter]

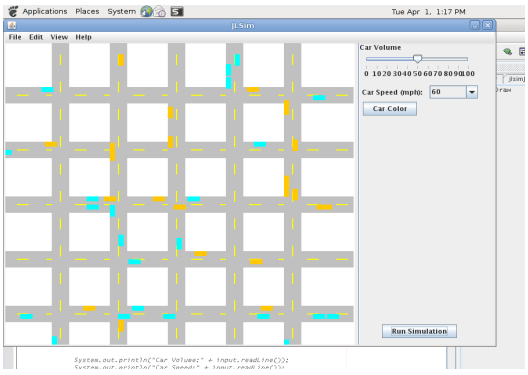


Second Prototype (Java) [2nd Quarter]

Abstract:
The primary goal of JLSim is to provide high customizability on the user-end. Many web applets have decent traffic simulations, but they offer minimal user interaction. The other primary goal is to provide an accurate simulation that reflects similarly to what would actually happen in the real world.

Background/Specification:
Written in Java, this application will use the Java swing class to implement to user interface. The program will be divided into two halves, the left halve being the visual part of the simulation and the right part being the extensive user interface where users can change program variables such as number of cars and traffic light length. (basic layout seen in screenshots)

Expected Results:
-Realistic simulation of real world traffic behavior.
-Extensive user interface to change program variables and design road networks.
-Crash analysis with independent probability calculations.(tentative)
-Design mode for developing road networks.(tentative)



3rd Quarter Prototype

Calculating Fractal Dimension from Vector Images

Kelly Ran

INTRODUCTION

Fractal dimension is a quantity that can be used as an index of complexity for fractals. In most research applications, fractal dimension is calculated using the raster graphics representation of images. This project investigates an alternative method by calculating fractal dimension from vector graphics.

The goal of this project is to demonstrate the viability of vector graphics calculations, to compare vector graphics calculations with raster graphics calculations, and to display a user interface that shows the calculations, step-by-step.

BACKGROUND

Fractals and fractal dimension. Fractals, geometric figures that exhibit self-similarity, are used in myriad applications. They accurately model many natural objects and phenomena, like tree branches, jagged coastlines, and particle motion.

Every fractal has a numeric fractal dimension that can be calculated using several methods. Researchers use fractal dimension as an index of complexity: it is used for texture classification in computer vision, protein molecule analysis in medicine, and plant growth analysis in botany.

Box dimension. To calculate fractal dimension using the box-counting method, a square grid of size s is superimposed over a black-and-white image of a fractal object. $N(s)$, the number of grid boxes that cover the object, is counted. Box dimension D is calculated using the formula .

$$D = \log(N(s)) / \log(1/s) \quad (1)$$

It can be seen that fractal dimension is an index of complexity by examining formula 1: as the scale of measurement s decreases, one can assess how the measurement $N(s)$ changes.

Graphics representation. Raster graphics is way of representing images. In raster graphics, images are collections of pixels, and numeric color values are stored for each pixel. Bitmap (BMP) is a raster graphics format. Vector graphics is a way of representing images by using primitives like paths (lines and curves) and points. Scalable Vector Graphics (SVG) is a format for creating and displaying vector graphics.

Raster graphics have poor resolution when they are examined on small scales. Vector graphics, on the other hand, retain clarity (Figure 2).

In this project, only path primitives were used. Paths consist of lines and curves. Path data is usually stored in a single attribute, and consists of commands and coordinates:

`d="M 200,66 L 266,66 L 266,133 L 200,133 L 200,66 z "`

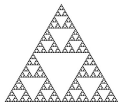
Commands such as `M`, `L`, `C`, and `z` indicate what kind of line or curve should be drawn.

Research applications of fractal dimension use raster graphics formats. Researchers take digital images (in raster format) and apply the box-counting method, using many grids of variable size on the same image. Their data are plotted on an x-y plane, with log($N(s)$) on the y-axis and log(1/s) on the x-axis. Using a linear regression, they find the line of best fit for the data, and D is the slope of that line.

METHODS

This presentation uses an example of a Sierpinski Carpet to demonstrate the methods of the fractal dimension calculator.

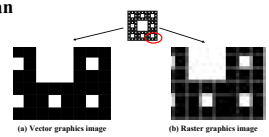
Creation of Scalable Vector Graphics (SVG). SVG images of fractals and non-fractals were created using Inkscape, an open source vector graphics editor. To make the Sierpinski Carpet fractal, a black square was cloned 9 times using the "Tiled Clones" option. The original square and the fifth cloned square were deleted. The remaining eight squares were grouped together and treated like the original square. Finally, all of the squares were transformed from object types to path types. SVG images were black-and-white, with black representing the object whose fractal dimension was to be calculated.



(a) Sierpinski Casket
D = 1.59

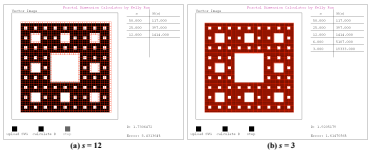
(b) Sierpinski Carpet
D = 1.89

FIGURE 1. Examples of fractals



(a) Vector graphics image (b) Raster graphics image

FIGURE 2. Zooming in: a comparison of vector and raster graphics



(a) s = 12 (b) s = 3

FIGURE 3. Screenshots of calculating the fractal dimension of a Sierpinski Carpet

s	$N(s)$	log(1/s)	log($N(s)$)
50	117	-1.699	2.068
25	397	-1.398	2.599
12	1414	-1.079	3.150
6	5107	-0.778	3.708
3	19333	-0.477	4.286

TABLE 1. Grid size s and number of grid boxes $N(s)$, calculated from the Sierpinski Carpet in Figure 3

s	$N(s)$	log(1/s)	log($N(s)$)
50	24	-1.699	1.380
25	96	-1.398	1.982
12	322	-1.079	2.508
6	1242	-0.778	3.094
3	4968	-0.477	3.696

TABLE 2. Data calculated from a rectangle

s	$N(s)$	log(1/s)	log($N(s)$)
50	7	-1.699	0.845
25	13	-1.398	1.114
12	25	-1.079	1.398
6	50	-0.778	1.699
3	98	-0.477	1.991

TABLE 3. Data calculated from a line

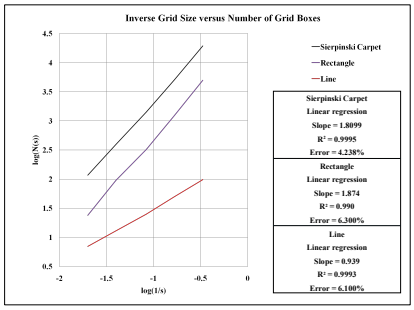


FIGURE 4. Graph of data from Table 1. The slope of the linear regression line is equal to calculated fractal dimension. Percent error is 4.238%.

All figures and tables by Kelly Ran

METHODS

Calculation of fractal dimension. The Processing language and environment were used to calculate fractal dimension. The SVG data file was loaded and parsed into an array of strings (each line of the file was an element of the array). Using methods from the Candy library, the SVG image was displayed onscreen. The functions `split(String s)`, `trim()`, and `indexOf(String s)` were used to process the SVG data file. Information for each path in the file was stored as an array of Coordinates, `paths`.

A global integer was created to keep track of s, grid size. A nested for-loop created an array, `grid`, that stored the Coordinates of each grid box's upper-left hand corner.

Every element in `grid` was tested to see if it covered any part of the black object (in the SVG image). Two methods were used: `case1` and `case2`. The number of elements in `grid` that covered the object was $N(s)$. S and $N(s)$ were stored in an array called `data`.

Whenever the button `step` was pressed, the grid size was decreased and the counting process was repeated. D, fractal dimension, was calculated using the 2 most recent additions to `data`:

$$D = [\log(N(s_2)) - \log(N(s_1))] / [\log(1/s_2) - \log(1/s_1)] \quad (2)$$

In the example in Figure 3, error was able to be calculated, because there is an actual value for the Sierpinski Carpet's fractal dimension.

$$\text{Percent error} = |D - D_{actual}| * 100 / D_{actual} \quad (3)$$

Showing the box method. To show the process of box-counting, red rectangles were superimposed over the SVG image on the screen (Figure 3). For every unique s, data were shown, along with calculated D and percent error.

CONCLUSION

Results. This project demonstrated that it is feasible to calculate fractal dimension from vector images. In the Sierpinski Carpet example, the calculated fractal dimension had 4.238% error, using all data points. A rectangle had a calculated dimension of 1.874, and a line had a calculated dimension of 0.939.

The box-counting method of calculating fractal dimension was shown onscreen.

Applications. In the future, this vector graphics method of calculating fractal dimension may be shown to be more efficient than the raster graphics method. In that case, researchers may be able to save time by converting their raster images to vector images and using the vector graphics method.

The properties of vector graphics allow minute image details to be shown. Researchers may want to find the fractal dimension of objects that have such minute detail, and using the vector graphics method may yield the best results in terms of accuracy. Researchers who use computer models of fractals may choose to use the vector graphics representation of their models.

Improvements and extensions of the fractal dimension calculator. The fractal dimension calculator program will be improved. In this phase, the program can process straight-line paths from SVG files. In the next phase, it will be able to process SVG paths that contain Bézier curves as well as straight lines.

The program will also be able to calculate fractal dimension from raster graphics, for the purpose of comparing raster graphics calculations with vector graphics calculations. The accuracy and efficiency (in terms of time) of both methods will be analyzed. A verification procedure will be designed to test if the calculator works properly.

A useful extension of the calculator program will upload many SVG files at once and calculate fractal dimension without visually showing the box-counting method.

An Interactive, User-driven Physics Simulator

Tom Smilack

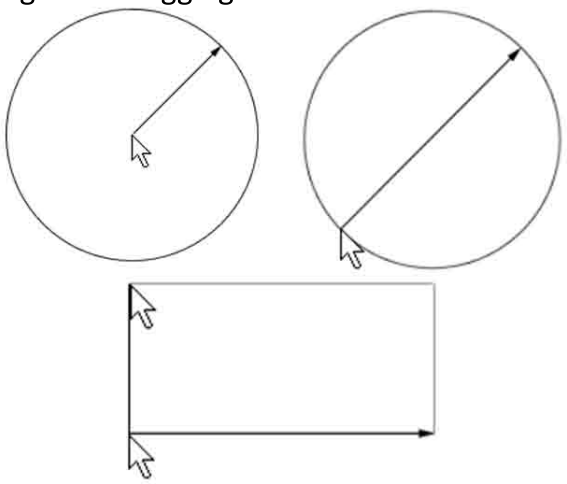
TJHSST Computer Systems Lab, 2007-2008

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 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.

Input

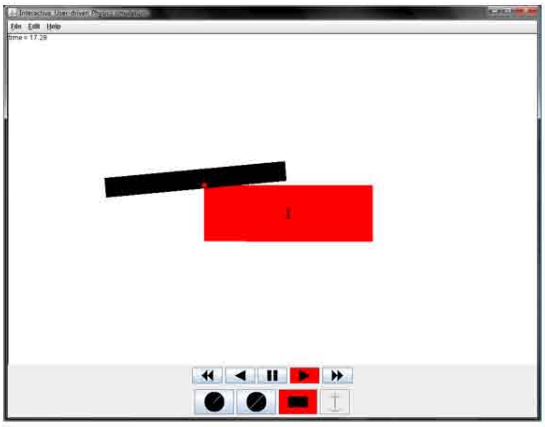
Although users cannot draw in my program, I wanted to employ an intuitive input method. I came up with two methods of creating circles and one of creating rectangles. They are explained in the diagrams below. An image of a cursor signifies a click, while a line with an arrow signifies dragging.



Background

ASSIST, the program that inspired my project, was made by a team at MIT. In ASSIST, the user uses a "sketchpad" system to draw a situation, which is then interpreted and fed into a commercial physics simulator. It was designed to help engineers in the beginning stages of planning a project, when precision matters less than ideas. My project is similar to ASSIST but focuses more on the physics of the user's situations than on the sketching system.

Screenshot



Preliminary Results

My program accurately represents projectile motion and collisions with walls without regard to friction, and with an elasticity of one. It also detects collisions between rectangles and can resolve them to some degree. More work is needed to finish collision modeling.

PRAM and the Ear Decomposition Algorithm

TJHSST Senior Research Project

2007-2008

Alex Valentin

The History of PRAM

PRAM refers to an abstract machine for designing algorithms in parallel. It allows for an infinite number of processors that can each access the same memory in uniform time. The first known implementation of PRAM is the University of Maryland A. James Clark School of Engineering's ParaLeap prototype 64-core supercomputer.

Steps of Ear Decomposition

Input: *A bridgeless, undirected graph G*

Output: *An ordered set of paths representing an ear decomposition of G*

begin

1. Find a spanning tree T of G

2. Root T at an arbitrary vertex r, and compute level(v) and p(v), for each vertex v ≠ r, where level(v) and p(v) are the level and the parent of v, respectively.

3. For each nontree edge, e= (u, v), compute lca(e)= lca(u,v) and level(e) = level(lca(e)). Set label(e): (level(e), s(e)), where s(e) is the serial number of e.

4. For each tree edge g, computer label(g).

5. For each nontree edge e, set P_e *= {e} U {g | label(g) = label(e) }. Sort the* P_e*'s by label(e).*

end

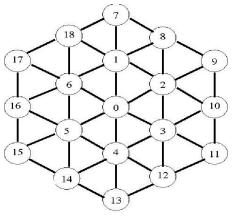
Abstract

This project takes the ear decomposition algorithm for partitioning maps and compares runtime efficiencies of different implementations. Four implementations are considered. Two implementations are run in a Parallel Random Access Machine (PRAM), while the other two are run serially. For each of these modes, one implementation uses only arrays and the other uses structures.

Procedures

The four implementations were written in XMT-C, even though the serial versions do not use the parallel capabilities. The Ear Decomposition was broken into three files, a main, span, and link. A convert file was also created for converting the input data from arrays to structures, if applicable. The span file finds a spanning tree of the data. The link file labels each edge and node with the correct ear. The main file runs the span and link files and then prints the ear of each edge and node.

Input data was given in the form of three arrays, vertices, degrees, and a two dimensional edges array. This data was quickly made with the help of the memMapCreate32 program in the XMT environment. Below is a visual of the hexagonal data set. Each data set was run on each of the four implementations of ear decomposition four times. The clock cycle counts were averaged and compared. The data is shown in Table 1 (which will arrive shortly.)

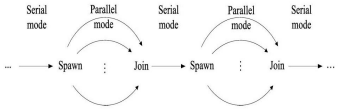


Results

The four ear decomposition implementations to be tested are the parallel using structures, parallel using only arrays, the serial using structures, and the serial using only arrays. The two parallel implementations should run faster than the serial implementations for obvious reasons. Of the two parallel implementations, the one using structures is expected to run slower because of the overhead caused by using structures. Therefore, from fastest to slowest, the implementations are parallel arrays, parallel structures, serial arrays, and serial structures.

How XMT-C Works

The language used on this supercomputer is called XMT-C, eXplicit Multi-Threaded C. Simply, the language is C with two extra methods, SPAWN and PS. The spawn method allows the programmer to use multiple processors. While any number of processors can be called for, the computer only has 64 processors to run at any given moment. The ps method, short for prefix sum, allows for the different threads to communicate with each other. The diagram below shows how XMT-C code alternates between serial mode and parallel mode.



Below is a simple XMT-C program that takes array A of length N and compacts its data into array B. N, Array A, and array B are declared in the header data.h.

```
#include "data.h"
#include <xmtc.h>
psBaseReg base;
int main()
{
base = 0;
spawn(0, N-1)
{
int step = 1;
    if( A[$] != 0)
    {
    ps(step, base);
    B[step] = A[$];
    }
}//end spawn
}//end main
```

The spawn method takes two integer arguments, which identify the range of the thread ids, inclusive. The id is referenced with the dollar sign, $. The ps method also takes two arguments, a local integer and a global psBaseReg. The local integer (*step* in the example above) is set to the global psBaseReg (*base* in the example above) and the psBaseReg is incremented by the size of step. Essentially, the ps method acts as a global counter without having to worry about concurrent writes.