A Gibbs Sampler Method for Maze Generation

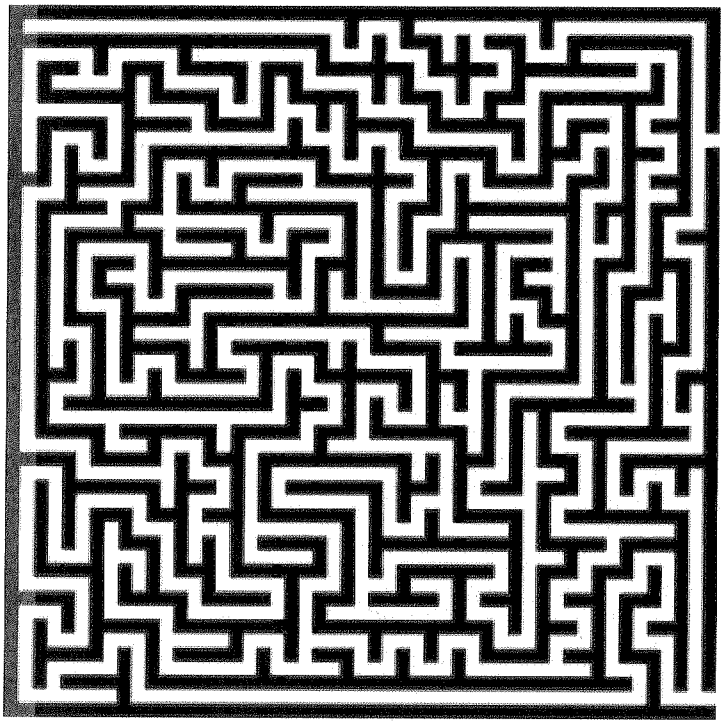
Brian Lynch

Math 350 Final Project

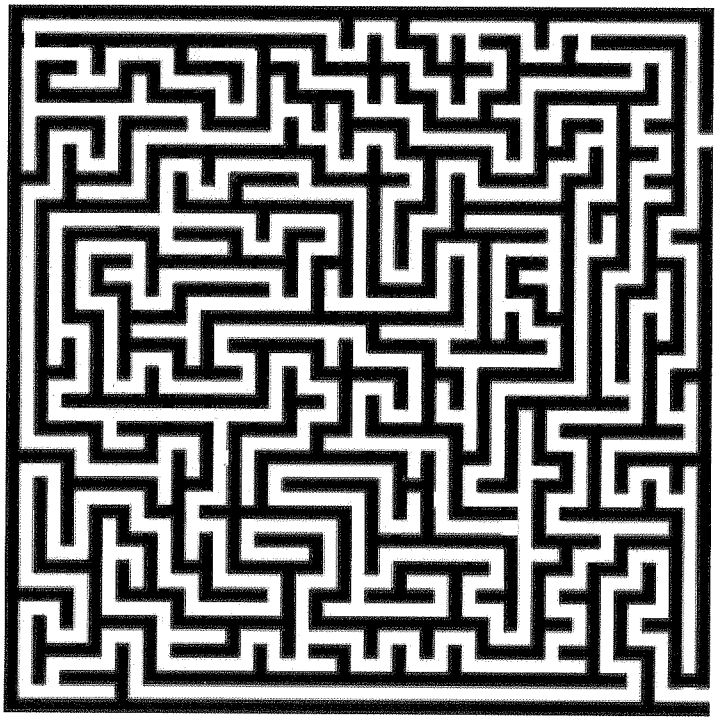
1 Introduction

A maze is a puzzle that requires the player to trace a path from one point, the start, to another, the end, on a 2-dimensional grid. This grid contains white space, referred to here as a "path", on which the player must travel, and black space, referred to here as a "wall", which the player cannot cross. Below are two examples of mazes. In each, the player must get from the top-right start to the bottom-right end. The mazes' solutions are on the next page in red:

Bad maze:

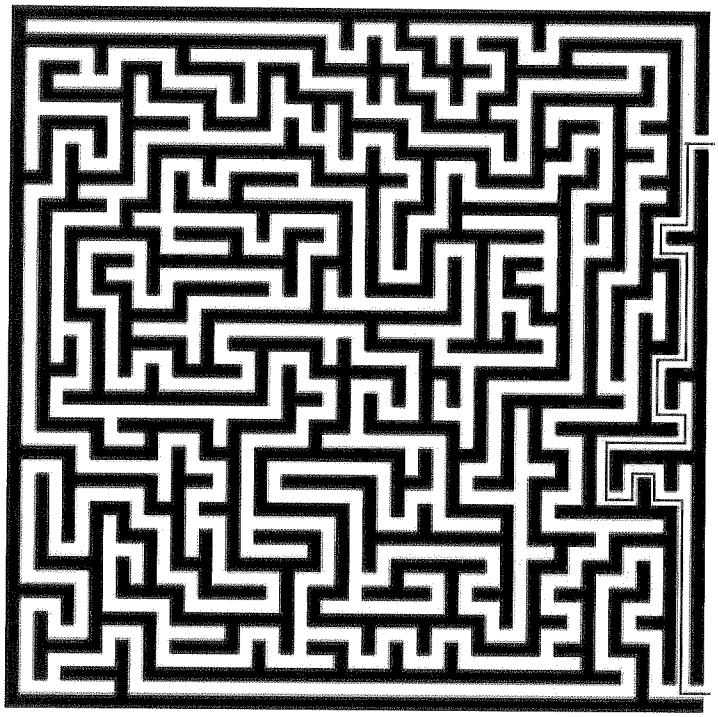


Good maze:

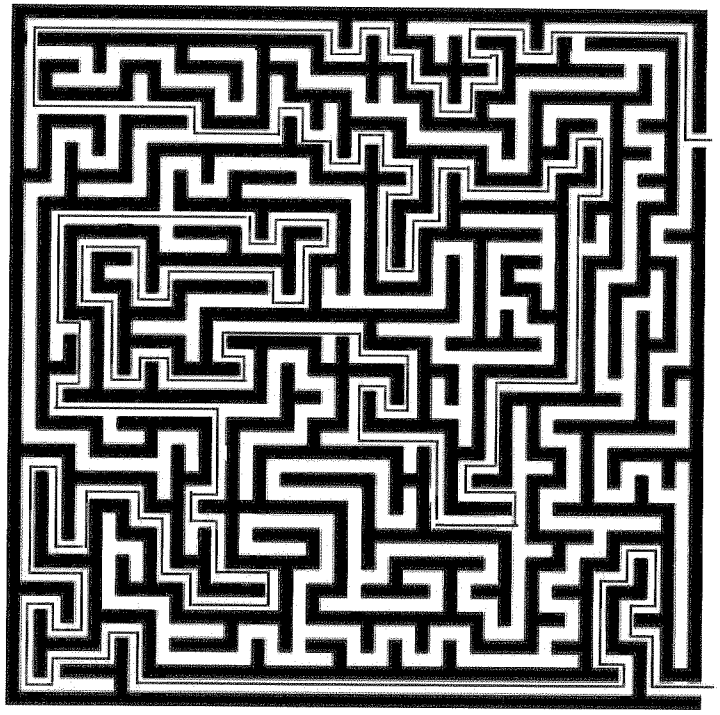


These mazes have been adapted from [1]

Bad maze:



Good Maze:



We can make a few generalizations about the above mazes. Both of them have all paths, or white space, accessible. Both have only one path that is the correct path, i.e. the path that successfully takes one from start to finish. The "bad" maze is undesirable because its correct path is very short and obvious, since the player's instinct is to take the shortest path downward to the end. The "good" maze is much more challenging. It has many long paths branching out from the correct path, all of which lead to dead ends. In order to reach the end, the player will most likely have to backtrack and traverse a large portion of the maze, making for a longer and more rewarding game.

In both of the above mazes, the set of white space can be viewed as a tree. A tree is a simple graph, i.e. a graph G=(V(G),E(G)), where V(G) is a non-empty finite set of elements, called vertices, and E(G) is a finite set of unordered pairs of distinct elements of V(G), called edges [2]. What distinguishes a tree from other simple graphs is that it has no circuits, or closed loops back onto itself [2]. The corners, intersections, and dead ends of the white paths comprise the vertices of the tree, and the white space connecting these vertices comprises the edges. One maze generation algorithm that treats a maze as a tree is Prim's algorithm, which constructs a maze from a minimum spanning tree [3]. However, it is possible to make challenging mazes that are not trees. Such mazes use paths that loop back on to each other in order to confuse the player.

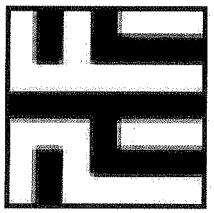
In this project, I will view mazes not necessarily as trees, but as configurations of a two-dimensional square-grid Ising model lattice. The Ising model describes a two-dimensional grid graph, a graph where each unit-length square can be thought of as a vertex v. Each vertex is connected to its four neighboring vertices, the squares which lie directly above, below, to the left, or to the right of it. V is defined to be the set of all vertices, and E is the set of all edges connecting neighboring vertices. We define a configuration of the graph as a function $\xi: V \rightarrow \{-1,1\}$, which assigns a value of either -1 or 1 to each vertex v. In the Ising model, each configuration has an associated energy $u(\xi)$. The probability of finding the graph in a given configuration is

$$\pi_\beta(\xi) = \frac{1}{Z(\beta)} e^{-\beta u(\xi)}$$

Here, β, the "temperature parameter," is a positive real number, and $Z(\beta)$ is a normalization factor. Thus, states with lower energy have higher probability [4]. In this project, I attempt to define an "energy function" $u(\xi)$ for a maze. With this, I use the Gibbs sampler method to generate mazes with desirable features.

2 Approach and Theory

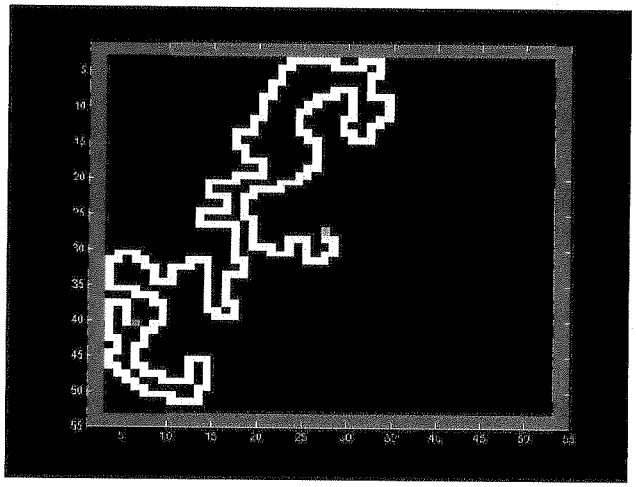
I define a maze as an Ising model grid graph, as described above. Here, mazes are graphs but not necessarily trees, since paths are allowed to cross and loop back onto each other. Each unit square of white space on the graph is defined to be a vertex with value -1, and each unit square of black space is defined to be a vertex with value 1. Black corresponds to a wall, while white corresponds to a path. Thus, a maze can be represented as a matrix of 1's and -1's, for example:



$\leftrightarrow$

-1	1	-1	1	-1	-1	-1
-1	1	-1	1	1	1	1
-1	-1	-1	-1	-1	-1	-1
1	1	1	1	1	1	1
-1	-1	-1	1	-1	-1	-1
-1	1	-1	1	1	1	1
-1	1	-1	-1	-1	-1	-1

The first step of my approach to generating mazes is to generate a fixed path, which starts at a fixed point at the center of the graph, defined to be the start of the maze, and creates a random path to another point, defined to be the end of the maze. This fixed path can be thought of as one guaranteed solution of the maze. An example of a fixed path I generated for a 50-by-50 grid graph is on the top of the next page.



Fixed path for a maze. The green square represents the start of the maze, and the blue square represents the end.

To create a maze, I simulated a Markov chain of states X_0, X_1, …, with X_0 defined to be the maze with only the fixed path white (-1), and all other spaces black (+1). The transitions for the Markov chain were determined as follows:

i. Let $\xi=X_n$ be the spin configuration at time step n;

ii. Pick a vertex v of the graph with equal probabilities. If v is on the fixed path, pick again;

iii. Set $X_{n+1}(\omega) = \xi(\omega)$ for all $\omega \neq v$, and $X_{n+1}(v) = s$, where s is picked with probability $\frac{1}{1+e^{\beta\Phi(v)}}$ if s=-1, and 1- $\frac{1}{1+e^{\beta\Phi(v)}}$ if s=+1.

The function $\Phi(v)$ is determined below.

The above procedure uses the properties of the Ising model and the Gibbs sampler method. To help determine the "energy function" $u(\xi)$ of a maze, which is to be minimized in a good maze, I list a few properties that would make a grid graph (with a fixed path in place) a good maze:

1) There should be long paths branching out from the fixed path, which should either lead to dead ends or loop back onto themselves.
2) One should be able to access all the white space on the maze.
3) There should not be large clusters of black space or white space.
4) There should not be simple alternate solutions, i.e. paths that lead to the end of the maze that are significantly shorter than the fixed path.

I could not find an exact theory to address all these characteristics. Rather, I focused on some of the more local, small scale characteristics of good mazes, which are:

1) The majority of vertices have two vertices of the same color and two vertices of the opposite color adjacent to them. Thus, having many vertices of this form should give the maze a low "energy".

2) Dead ends are created when a white vertex has one white vertex and three black vertices adjacent to it. There also seem to be many black vertices that are surrounded by three white and one black. These types of vertices should contribute a moderately low "energy".
3) Also present are vertices that have three vertices of the same color and one vertex of a different color adjacent to them. These are important to have in white vertices, which need to "branch out" to create interesting mazes.

To try to capture these characteristics, I define a maze's energy function as follows:

$$u(\xi) = \sum_{v} h(v)$$

This is summed over all vertices v of the maze, where h(v) depends on the color of v and the color of the vertices directly adjacent to v. Reasoning from the above characteristics of a maze, I initially defined h(v) to be:

h(v)=

		Number of black (+1) vertices adjacent to v				
		0	1	2	3	4
Color of v	Black (+1)	1	-1	-2	-1	1
	White (-1)	1	-1	-2	-1	1

This definition assigns low energy to vertices with two of the same color adjacent, and relatively low energy to vertices with one or three of the same color adjacent. Thus, I reasoned that this h(v) should produce mazes that follow the small-scale trends described above.

The algorithm described above creates a Markov chain that is irreducible, i.e. any state can reach any other in a finite number of steps, and aperiodic, i.e. any state can reach any other state in an even or odd number of steps (this can be concluded from the fact that any state can transition to itself). If such a chain follows the detailed balance condition

$л_\beta(\xi)P(\xi,\eta) = л_\beta(\eta)P(\eta,\xi)$ for any 2 configurations ξ and η, then the chain will converge to a single stationary distribution in the Gibbs sampler algorithm. This stationary distribution represents the mazes with the most desirable features, determined by the definition of h(v).

The Gibbs sampler method is a special case of the Metropolis-Hastings MCMC algorithm. It always accepts a given transition, and selects a transition with a uniform conditional probability [5]. In the case of the Ising model, for each step in the Markov chain, the Gibbs sampler method dictates that a vertex be chosen randomly, and that its value be chosen according to the probability $л_\beta(X(v) = \pm 1 | X(\omega) = \xi(\omega)\ for\ all\ \omega \neq v)$ for vertices v and ω. We now

show that the proposed model follows the detailed balance condition (this closely follows the method described in [6]):

Define $\xi_{\pm}(v) = \pm 1$ and $\xi_{\pm}(\omega) = \xi\ (\omega)$

For the proposed algorithm, $P(\xi, \eta)$=0 unless ξ and η are the same or differ only in one vertex, v. Clearly, the detailed balance condition holds for ξ=η. For $\xi \neq \eta$, the detailed balance condition becomes $л_\beta(\xi_+)P(\xi_+, \xi_-) = л_\beta(\xi_-)P(\xi_-, \xi_+)$.

Note that $\frac{л_\beta(\xi_+)}{л_\beta(\xi_-)} = \frac{e^{-\beta u(\xi_+)}}{e^{-\beta u(\xi_-)}} = e^{\beta\Phi(v)}$

Where $\Phi(v) = u(\xi_-) - u(\xi_+) = h(v = -1) - h(v = +1) + h(\uparrow_-) - h(\uparrow_+) + h(\downarrow_-) - h(\downarrow_+) + h(\rightarrow_-) - h(\rightarrow_+) + h(\leftarrow_-) - h(\leftarrow_+)$ Where, for example, $\uparrow$ refers to the vertex directly above v, and $h(\uparrow_-)$ is the energy value of that vertex when v has a value of -1.

We thus have:

$$л_\beta(X(v) = \pm 1 | X(\omega) = \xi(\omega)\ for\ all\ \omega \neq v) = \frac{P(X(v) = \pm 1 \cap X(\omega) = \xi(\omega)\ for\ all\ \omega \neq v)}{P(X(\omega) = \xi(\omega)\ for\ all\ \omega \neq v)}$$

$$= \frac{л_\beta(\xi_\pm)}{л_\beta(\xi_+) + л_\beta(\xi_-)}$$

And

$$\frac{л_\beta(\xi_-)}{л_\beta(\xi_+)+л_\beta(\xi_-)} = (1 + \frac{л_\beta(\xi_+)}{л_\beta(\xi_-)})^{-1} = (1 + e^{\beta\Phi(v)})^{-1} = \frac{1}{1+e^{\beta\Phi(v)}}$$

The Gibbs sampler transitions probabilities are thus

$$л_\beta(X(v) = s | X(\omega) = \xi(\omega)\ for\ all\ \omega \neq v) = \begin{cases} \frac{1}{1+e^{\beta\Phi(v)}}\ if\ s = -1 \\ 1 - \frac{1}{1+e^{\beta\Phi(v)}}\ if\ s = +1 \end{cases}$$

With the Markov chain defined as above, $P(\xi_+, \xi_-) = P(\xi_-) = \frac{1}{1+e^{\beta\Phi(v)}}$, and $P(\xi_-, \xi_+) = P(\xi_+) = 1 - \frac{1}{1+e^{\beta\Phi(v)}} = \frac{e^{\beta\Phi(v)}}{1+e^{\beta\Phi(v)}}$. Thus, $\frac{P(\xi_-, \xi_+)}{P(\xi_+, \xi_-)} = e^{\beta\Phi(v)}$, and the detailed balance condition is satisfied.

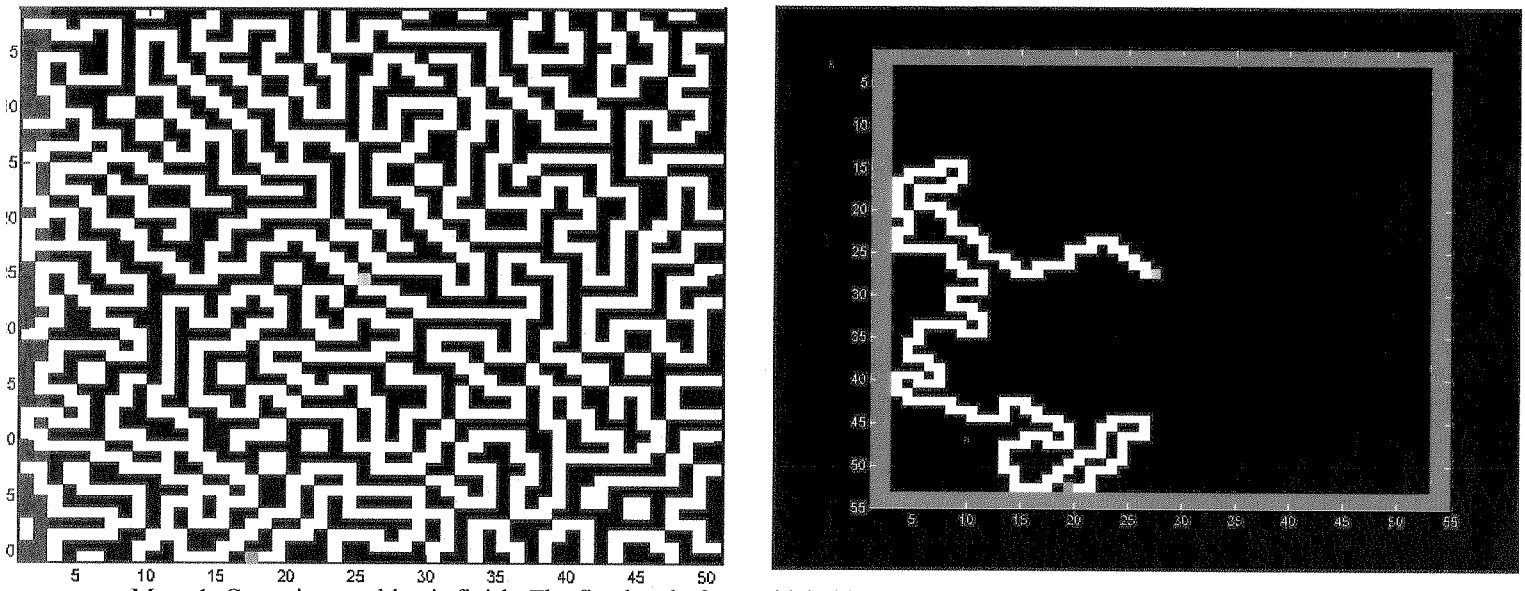
3 Simulation and Results

I now show and describe the results of my maze generation algorithm. For the MATLAB code that I used to create the following mazes, see Appendix 1. My first somewhat successful mazes used a "temperature parameter" of β=3, and an "energy function" of

h(v)=

		Number of black (+1) vertices adjacent to v				
		0	1	2	3	4
Color of v	Black (+1)	2	0	-.5	-.5	2
	White (-1)	1	0	-2.5	-1	1

This is a slight modification of my theoretically-predicted h(v). I made the adjustments to my initial prediction empirically, running the simulation for slightly different h(v) values until I came upon one that gave the best mazes. One example of a maze I generated with this h(v) is given below:



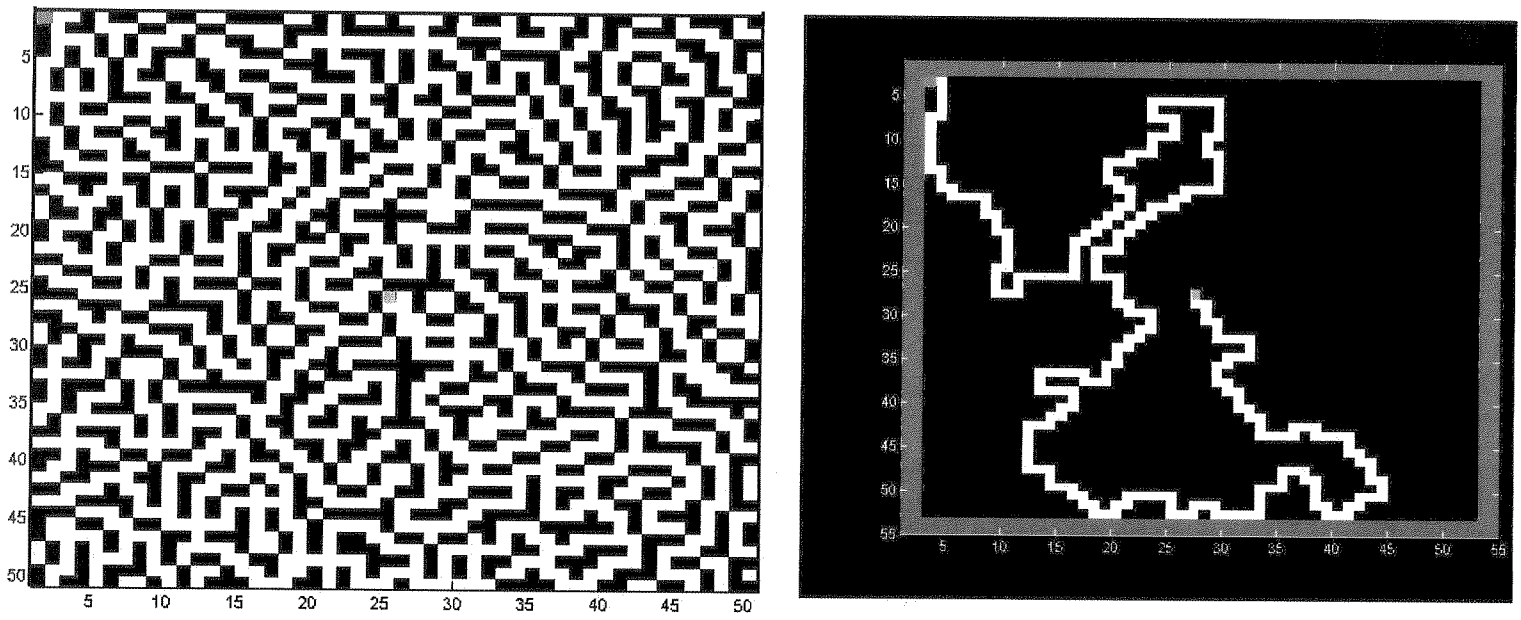
Maze 1: Green is start, blue is finish. The fixed path, from which this maze was generated, is on the left.

This maze generally shows the qualities of a decent maze. It is mostly made of straight white paths, and has several routes branching from the correct path that lead to dead ends. One problem with this maze is its unused white space. Many of the paths in the upper-right of the maze are inaccessible, and there are some strange shapes, such as lone white rectangles, which make the maze seem unprofessional.

From experimenting with random values of h(v), I was able to generate some good mazes. One of them is shown on the next page. It uses $\beta=3$, and

h(v)=

		Number of black (+1) vertices adjacent to v				
		0	1	2	3	4
Color of v	Black (+1)	2	0	.5	1	1
	White (-1)	1	1	-1	0	1



Maze 2: Green is start, blue is finish. The fixed path is on the left.

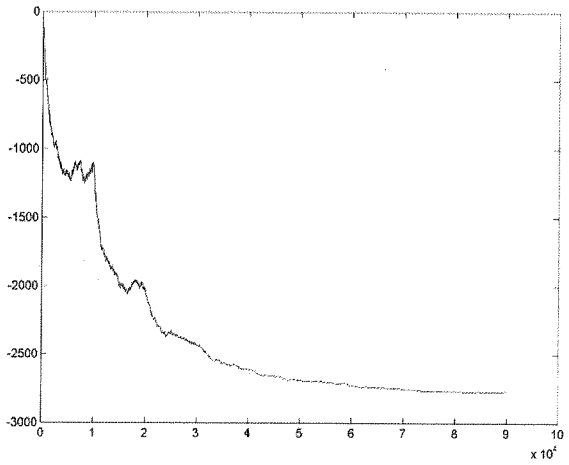
This maze is somewhat challenging, as it has many long paths that lead to dead ends. Most paths are accessible, though there are still some isolated white areas.

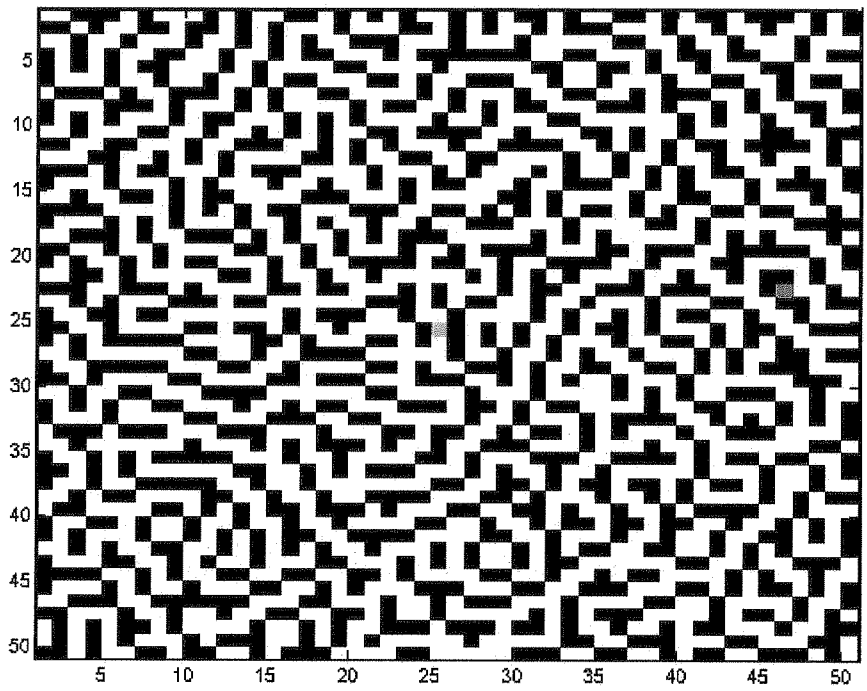
Next, instead of using a fixed β parameter, I experimented with "cooling down" the system by slowly increasing β as the Markov chain was run (for the MATLAB code used to do this, see Appendix 2). For the following maze, I increased β from 0 to 4 in increments of .5, each β being used for 10000 steps in the chain. I returned to an h(v) value near my theoretical prediction,

h(v)=

		Number of black (+1) vertices adjacent to v				
		0	1	2	3	4
Color of v	Black (+1)	2	-.5	.5	.5	1
	White (-1)	1	1	-1	0	1

One maze generated from this "cooling down" process is shown on the next page. The plot of its "energy" as a function of time step in the chain is shown below. We see the process successfully produces as "low energy" maze.





Maze 3: Green is start, blue is finish.

This is a fairly challenging maze where most paths are accessible.

4 Discussion/Conclusions

Maze generation using Gibbs sampling proved to be a rather inexact science. Most of the challenge lied in determining the qualities of a good maze, and mathematically implementing them through my "energy function". There are subtle differences between a challenging maze and an easy one, and I found that a given energy function could produce both types equally often, since a good maze has many available paths, but one path that happens to create a shortcut to the end could ruin the maze.

One general problem with my mazes was that they were made up of jagged paths, while most professionally-made mazes use long, straight paths. Jagged paths are not as aesthetically appealing, nor do they efficiently use space. I do not think my energy function could be adjusted to create longer straight paths though, since it only takes into account small-scale relations between the vertices of the maze. It defines energy based only on the color of individual vertices and those adjacent to them, and not on their relation to vertices further away.

Nevertheless, I was able to create some challenging mazes. The MCMC algorithm was fast, and could be easily run a few times so that I could pick and choose the best mazes of the ones I generated. Overall, this exploration showed that the theory of Gibbs sampling, based in Ising model energy considerations, could be applied to grid-graphs to create mazes. Perhaps by defining the energy of graph in a more sophisticated manner, this theory could be extended to more reliably create professional-looking, challenging mazes.

References

[1] Stack Overflow (2012). "Algorithm to Generate a Segment Maze." http://stackoverflow.com/questions/2641964/algorithm-to-generate-a-segment-maze

[2] Wilson, Robin J. (1985). *Introduction to Graph Theory* (3rd ed). New York, NY: Longman Inc.

[3] "Prim's Algorithm." Wikipedia. http://en.wikipedia.org/wiki/Prim%27s_algorithm

[4] Feres, Renato. Math 350 Lecture Notes, Fall 2012.

[5] Walsh, B. (2004). "Markov Chain Monte Carlo and Gibbs Sampling." MIT Lecture Notes. http://web.mit.edu/~wingated/www/introductions/mcmc-gibbs-intro.pdf

[6] Feres, Renato. Math 350 Homework 10, Fall 2012

Appendix 1: MATLAB code for maze generation

```
%specify maze dimensions
r=50;
s=50;
%create fixed path
numberspaces=0;
while numberspaces<=r*s/10
M=zeros(r+4,s+4);
M(3:(r+2),3:(s+2))=ones(r,s);
currx=ceil((r+3)/2);
curry=ceil((s+3)/2);
M(currx,curry)=-1;
for j=1:10*r*s
    nextspot=ceil(rand*4);
    if nextspot==1
        if M(currx,curry+1)==0||M(currx,curry+1)==-1||M(currx,curry+2)==-
1||M(currx-1,curry+1)==-1||M(currx+1,curry+1)==-1
        else
            curry=curry+1;
            M(currx,curry)=-1;
        end
    elseif nextspot==2
        if M(currx,curry-1)==0||M(currx,curry-1)==-1||M(currx,curry-2)==-
1||M(currx+1,curry-1)==-1||M(currx-1,curry-1)==-1
        else
            curry=curry-1;
            M(currx,curry)=-1;
        end
    elseif nextspot==3
        if M(currx-1,curry)==0||M(currx-1,curry)==-1||M(currx-2,curry)==-
1||M(currx-1,curry+1)==-1||M(currx-1,curry-1)==-1
        else
```

```
            currx=currx-1;
            M(currx,curry)=-1;
        end
    else
        if M(currx+1,curry)==0||M(currx+1,curry)==-1||M(currx+2,curry)==-
1||M(currx+1,curry+1)==-1||M(currx+1,curry-1)==-1
        else
            currx=currx+1;
            M(currx,curry)=-1;
        end
    end
end
numberspaces=sum(sum(M==-1));
end
%change fixed path to 0's
M=M+ones(r+4,s+4);
M=M-(M==1);
M=M-(M==2);
%run markov chain
beta=2;
steps=100000;
%h=[0 -1 -2 0 2;1 -1 -3 -2 1;0 0 0 0 0];
h=[3 -1 -1 1 1;1 1 -3 0 1;0 0 0 0 0];
for k=1:steps
    randv1=ceil(r*rand);
    randv2=ceil(s*rand);
    if M(randv1+2,randv2+2)==0
    else
        M(randv1+2,randv2+2)=1;
        sign=M(randv1+2,randv2+2);
        upsign=M(randv1+2,randv2+3);
        downsign=M(randv1+2,randv2+1);
        rightsign=M(randv1+3,randv2+2);
        leftsign=M(randv1+1,randv2+2);
        upup=M(randv1+2,randv2+4);
        upright=M(randv1+3,randv2+3);
        upleft=M(randv1+1,randv2+3);
        downdown=M(randv1+2,randv2);
        downright=M(randv1+3,randv2+1);
        downleft=M(randv1+1,randv2+1);
        rightright=M(randv1+4,randv2+2);
        rightup=M(randv1+3,randv2+3);
        rightdown=M(randv1+3,randv2+1);
        leftleft=M(randv1,randv2+2);
        leftup=M(randv1+1,randv2+3);
        leftdown=M(randv1+1,randv2+1);
        vup=upsign+3*(upsign~=1);
        vdown=downsign+3*(downsign~=1);
        vright=rightsign+3*(rightsign~=1);
        vleft=leftsign+3*(leftsign~=1);
        A=sum([upsign downsign rightsign leftsign]==1);
        B=sum([upup upright upleft sign]==1);
        C=sum([downdown downright downleft sign]==1);
        D=sum([rightright rightup rightdown sign]==1);
        E=sum([leftleft leftup leftdown sign]==1);
        Phi=h(2,A+1)-h(1,A+1)+h(vup,B)-h(vup,B+1)+h(vdown,C)-
h(vdown,C+1)+h(vright,D)-h(vright,D+1)+h(vleft,E)-h(vleft,E+1);
```

```
        probminus=1/(1+exp(beta*Phi));
        if rand<=probminus
            M(randv1+2,randv2+2)=-1;
        else
            M(randv1+2,randv2+2)=1;
        end
    end
end
%visualize maze
M=M(3:(r+2),3:(s+2));
M=M-ones(r,s);
M=M+3*(M==-2)+2*(M==-1);
M(currx-2,curry-2)=.5;
M(ceil((r+3)/2)-2,ceil((s+3)/2)-2)=.5;
[b,c] = size(M);                              %# Get the matrix size
imagesc((1:c)+0.5,(1:b)+0.5,M);               %# Plot the image
colormap(gray);
```

Appendix 2: MATLAB code for generation with cooling, and energy plotting

```
%run markov chain
betavector=0:.5:4;
stepsvector=10000*ones(1,9);
% steps=100000;
%h=[0 -1 -2 0 2;1 -1 -3 -2 1;0 0 0 0 0];
h=[1 -.5 .5 0 1;1 1 -1 0 1;1 1 -1 0 1];
totalu=0;
for o=1:length(betavector)
    u(1)=totalu(length(totalu));
for k=1:stepsvector(o)
    randv1=ceil(r*rand);
    randv2=ceil(s*rand);
    currentspace=M(randv1+2,randv2+2);
    if M(randv1+2,randv2+2)==0
        u(k+1)=u(k);
    else
        M(randv1+2,randv2+2)=1;
        sign=M(randv1+2,randv2+2);
        upsign=M(randv1+2,randv2+3);
        downsign=M(randv1+2,randv2+1);
        rightsign=M(randv1+3,randv2+2);
        leftsign=M(randv1+1,randv2+2);
        upup=M(randv1+2,randv2+4);
        upright=M(randv1+3,randv2+3);
        upleft=M(randv1+1,randv2+3);
        downdown=M(randv1+2,randv2);
        downright=M(randv1+3,randv2+1);
        downleft=M(randv1+1,randv2+1);
        rightright=M(randv1+4,randv2+2);
        rightup=M(randv1+3,randv2+3);
        rightdown=M(randv1+3,randv2+1);
        leftleft=M(randv1,randv2+2);
        leftup=M(randv1+1,randv2+3);
        leftdown=M(randv1+1,randv2+1);
        vup=upsign+3*(upsign~=1);
```

```
        vdown=downsign+3*(downsign~=1);
        vright=rightsign+3*(rightsign~=1);
        vleft=leftsign+3*(leftsign~=1);
        A=sum([upsign downsign rightsign leftsign]==1);
        B=sum([upup upright upleft sign]==1);
        C=sum([downdown downright downleft sign]==1);
        D=sum([rightright rightup rightdown sign]==1);
        E=sum([leftleft leftup leftdown sign]==1);
        Phi=h(2,A+1)-h(1,A+1)+h(vup,B)-h(vup,B+1)+h(vdown,C)-
h(vdown,C+1)+h(vright,D)-h(vright,D+1)+h(vleft,E)-h(vleft,E+1);
        probminus=1/(1+exp(betavector(o)*Phi));
        if rand<=probminus
            M(randv1+2,randv2+2)=-1;
        else
            M(randv1+2,randv2+2)=1;
        end
        newspace=M(randv1+2,randv2+2);
        if currentspace==newspace
            u(k+1)=u(k);
        elseif currentspace>newspace
            u(k+1)=u(k)+Phi;
        else
             u(k+1)=u(k)-Phi;
        end
    end
end
totalu=[totalu u];
end
%visualize maze
M=M(3:(r+2),3:(s+2));
M=M-ones(r,s);
M=M+3*(M==-2)+2*(M==-1);
M(currx-2,curry-2)=.5;
M(ceil((r+3)/2)-2,ceil((s+3)/2)-2)=.5;
[b,c] = size(M);                          %# Get the matrix size
imagesc((1:c)+0.5,(1:b)+0.5,M);           %# Plot the image
colormap(gray);
pos=1:length(totalu);
% plot(pos,totalu)
```