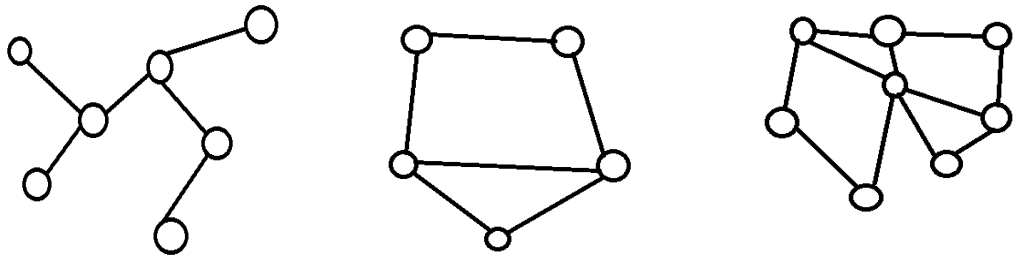
Robert Ramirez
ID: 420118
Math 350
Final Project

# Applications of Random Mazes and Graphs Generated via Markov Chain Monte Carlo Methods

## Introduction

**Mazes** are defined as complex series of pathways, usually meant to be navigated from one end to another. Mazes are familiar puzzles to almost everyone, and solving mazes is a simple aspect of both childhood and adulthood that opens the mind to critical thinking, and passes the time. They have widespread use in psychology, and are printed in magazines and newspapers and available immediately by the millions online. Mazes are often developed using random algorithms, but the most interesting and fascinating ones are still created by hand, and likely always will be. Leonard Euler was the first to study mazes mathematically, and in doing so he opened up the realm of topology. Thus, mazes are a fundamental aspect of advanced mathematics.

The common maze is more properly called a **perfect maze** – one which contains no loops. A perfect maze is analogous to a **tree** in graph theory. They are essentially one and the same thing, although by definition a maze is complex. However, a maze is defined as perfect if it is a tree. In graph theory, a tree is defined as a graph without cycles. We will consider any disjoint union of connected points to be its own **graph** of edges and vertices. A simple observation about the number of cycles, or **loops**, in a graph can be made from the following pictures of several graphs:

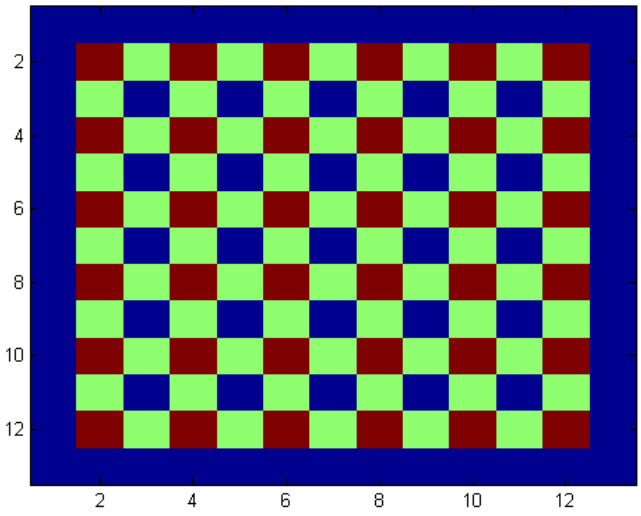


Notice that the number of loops is always equal to the number of edges minus the number of vertices, plus one, for a connected graph. This will be useful for us later.

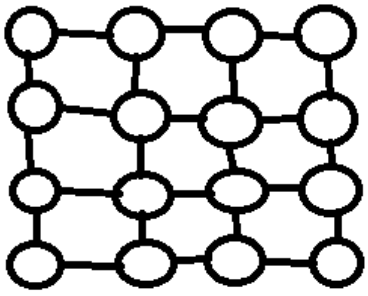
In order to study mazes, one must first have an efficient way of creating different mazes with the same properties many times. In other words, it would be ideal to have some kind of random maze generation algorithm. These algorithms usually require complex recursive methods and object oriented programming. A way around these methods that is perhaps more mathematically elegant is Markov Chain Monte Carlo to stepwise create a random maze. Alas, it proves to be very difficult to create a random perfect maze using nothing but simple non-recursive algorithms devoid of data structure beyond simple arrays. However, some comments on the necessity of studying perfect mazes and some observations using graph theory will prove to be a way around this dilemma.

Consider some problems studying arbitrary paths could be useful for. First, note that random mazes are essentially arbitrary, and that although there are a huge number of possible random mazes of any given size, they should all have the same properties if generated using the same algorithm, and thus statistical analyses can be performed on them. Problems random paths might be useful for investigating range from traffic flow and fluid flow throughout conduits or in general, to topology and game theory, to fire rescue, counterterrorism, and real-world searches of various kinds. The purpose of this paper is to investigate some of these applications. Noting that these are real-world problems, a perfect maze might not be necessary. We will thus focus our study on general graphs, but graphs that are complex and maze-like, and that are generated randomly and uniformly. Ideally, the Markov Chain we produce will have a stationary probability distribution. We will assume it does, but we will study its properties to give a more convincing argument for this.

In order to produce a useful graph, we begin with the following graph in the form of a complete grid. The vertices are colored red, and the edges connecting them to their neighbors are green:




Note that each vertex has at least 2 neighbors, and each can only connect to neighbors up, down, left, and right of it. Thus, with this representation of a graph, we have some empty (blue) spaces. An empty border is also included in this graph, in order to allow the outer vertices to have neighboring elements for purposes of programming. This grid-graph is equivalent to the following type of graph (note that the dimensions of the two graphs are different):



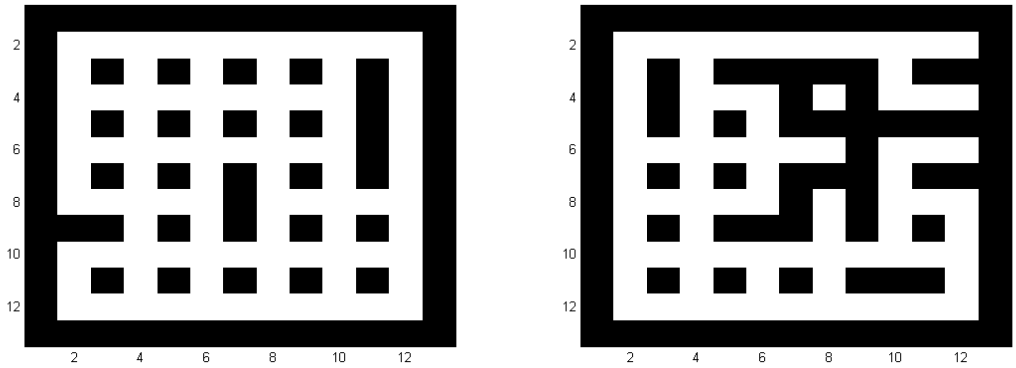
Each image we produce from now on shall be called a **grid**. A grid may contain multiple graphs, but if it contains only one, we will call the grid **connected**. A maze traditionally consists of a single graph. That is, a maze grid is usually connected. We will therefore do our best to uphold this condition, having already abandoned the condition of having no loops.

The Maze Generation Algorithm

Our goal now is to produce a random set of edges connecting a grid of vertices. We ideally want our grid to always be connected. Our Markov Chain will thus be defined as follows:

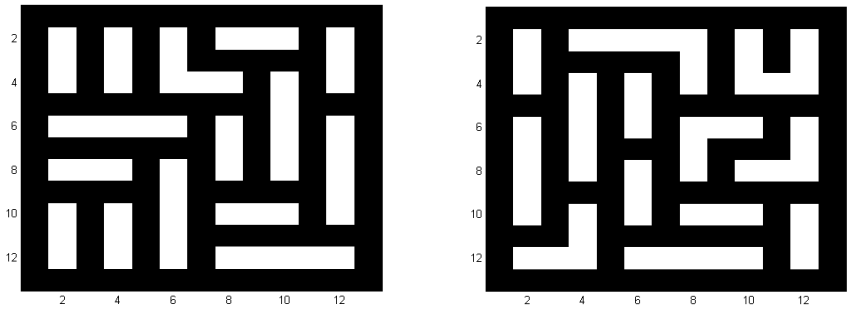
1. Remove an edge at random.
2. If removing that edge causes the graph to become disconnected, reject the step. Else, accept.

It is quite easy to randomly select an edge and to remove it. In our program, we will simply assign a value of 1 to a path space, and a value of 0 to a wall or boundary space. Because edges and vertices are both part of the path of a maze, they can visually appear the same. Thus our algorithm after a few steps begins to create images such as the following:

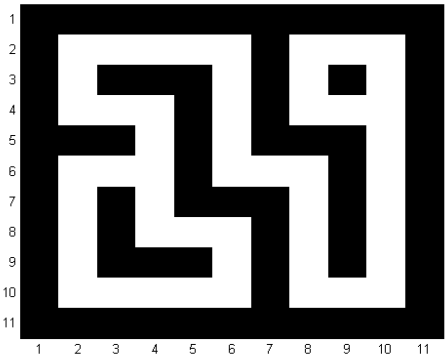


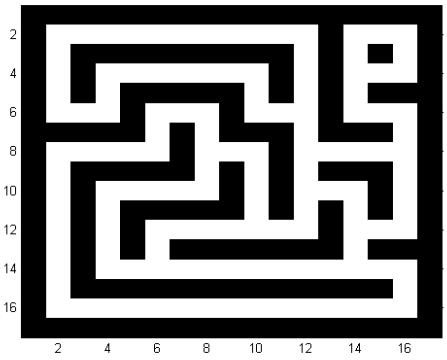

But the graph on the right has become disconnected! The problem of indirectly detecting if removing an edge causes the graph to become disconnected is not an easy one. It is usually solved using the same algorithms involved in generating random mazes, such as depth-first search or breadth-first search. To keep our algorithm simple, we must use some graph theory.

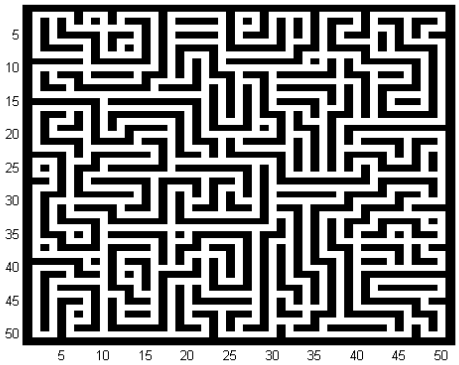
It can be hypothesized after examination of such graphs, that the best way to easily test if a graph is connected is by testing the number of connections each vertex has. At first thought, one connection per vertex should suffice, but it quickly becomes clear that this results in graphs such as the following:

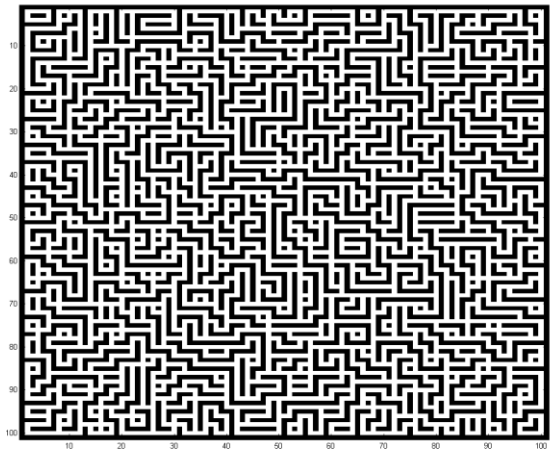


Three connections appears to result in graphs that do not differ by much from the original, complete grid, and four of course results in no changes at all. The ideal number appears to be two connections, at small scale, and then at large scale, as well. We give four random examples:









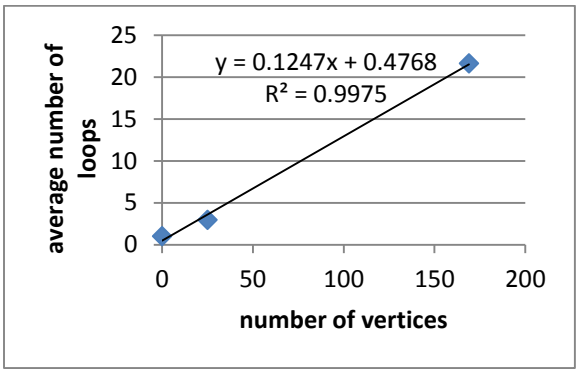
It should be noted that these images are actually images of matrices which store our grids. You may also notice the dimensions of the graphs. In increasing order, they are 11x11, 17x17, 51x51, and 101x101. We are using square, n x n (call them size n) grids, and in order to have a complete border, the size must always be an odd number. The number of vertices is given by $((n-1)/2)^2$. Thus, a graph of size n=101 is 50 vertices wide and has 2500 vertices. Take a closer look at the size 101 grid. In the top left and bottom right, and at a couple other locations near the boundary, there are disjoint loops. But looking at the other graphs and the scarcity of these disjoint loops, it would seem that the most of the time, two connections is sufficient for connectedness of the entire graph. Further analysis suggests that the only disjoint loops that occur with a significant probability are small loops. We could statistically model this, but it would have to be done manually.

Disconnects are fairly uncommon, and they are typically insignificant. In addition, increasing the size of the grid decreases the fraction of the graph that is disconnected (naturally). For our simulations, then, we would be better off using large graphs. Notice, however, that the number of loops is rather large, and in particular, there are a significant number of small, "trivial" loops in each graph. We call them trivial for the following reasons:

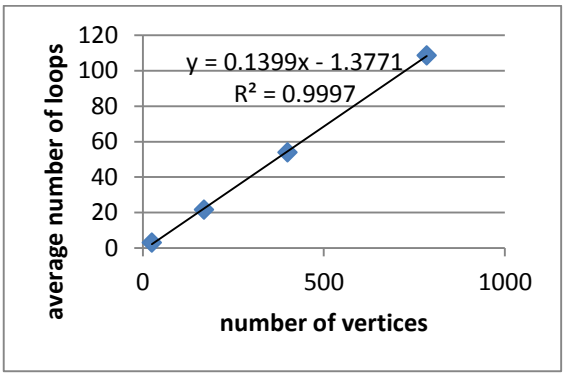
1. They are the result of the program doing nothing.
2. They are the smallest possible loops.

3. They are easily avoided while solving the maze and it is immediately clear that they are unnecessary to travel around.

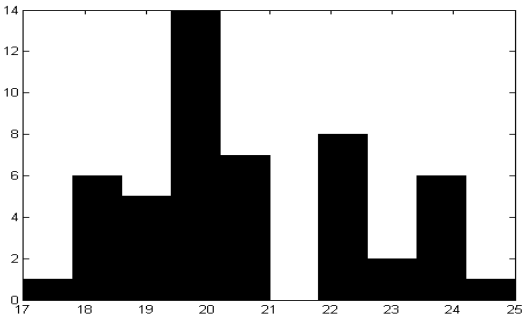
We can calculate the average number of loops each graph has for a given size using a simulation. The graph and trend line are displayed below, along with a histogram for n=27. In addition, we can analyze the number of loops in a given graph over time (1 step = 1 unit time).



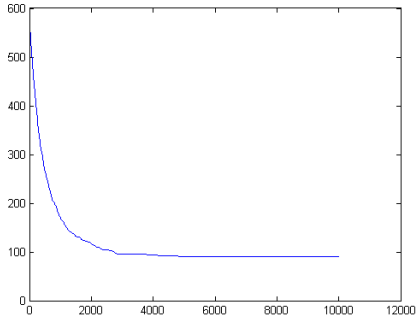

Predicted graph after 2 simulations using (0,1) as a third point.




Actual graph after all 4 simulations




# of loops histogram for n=27




# of loops graph over time for n=51

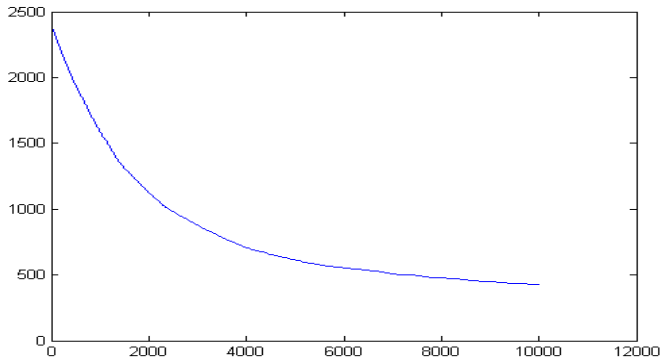
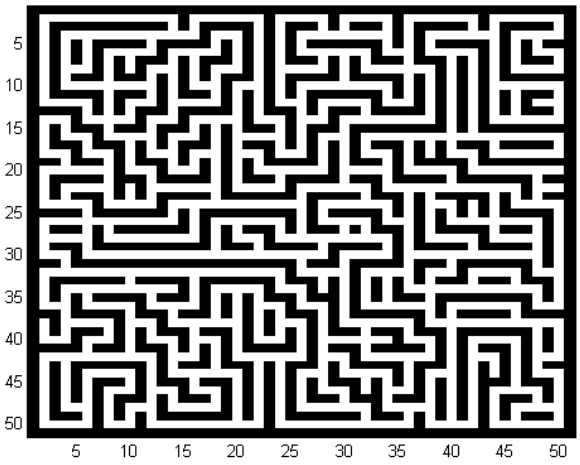



Figure A: The number of loops levels off smoothly over time for n=101, but 10,000 are not enough steps to reach the minimum number of loops to achieve a finished maze.

Comparing the predicted and actual graphs and trend lines of the final number of loops with respect to the number of vertices in one dimension tell us that our Markov Chain creates mazes of the same "type" each time. That is, our mazes are all indeed random samplings from the space of mazes of the type we have created with our algorithm. From the other graphs we glean three things. First, there are a huge number of loops. The other things are that the number of loops approaches a minimum as the number of steps increases, and the final number of loops is

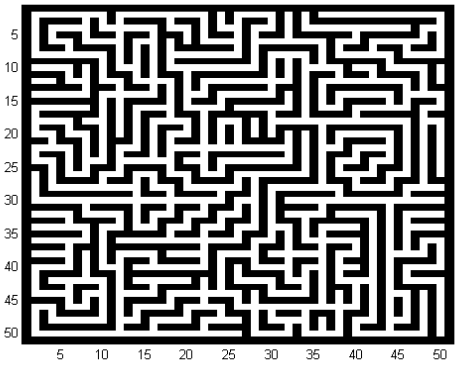
relatively constant for a given size graph. This means that for our simulations, 10000 steps is enough to guarantee a "completed" graph. For simulations on much larger graphs of the order n=101, 20,000 seems more appropriate (Figure A). This is important for analyzing the time it takes for a graph of a given size to become complete and allows us to gauge the number of steps required per simulation.

You may notice that the number of trivial loops is not insignificant. It behooves us to attempt to eliminate these loops for the sake of creating a more mazelike graph. We first break the trivial loops at a random point, and then if they are disconnected from the main graph, we connect them to a random vertex. The code for doing this is included on page 11. The appendices contain all the code that was used in this investigation. A maze generated with this new algorithm is shown below:




Notice there are now a few dead-ends, which are the remnants of the broken trivial loops. The added code also connects any (former) trivial loops to the rest of the graph. This code is convenient because it is easy to detect trivial loops – their immediate neighbors are all connections (valued 1). We will call this type of maze type 2, and the original one with a lot of loops type 1.

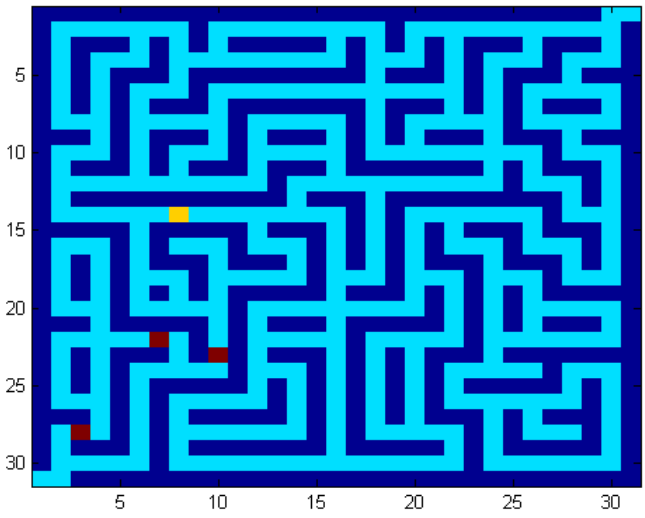
For aesthetic purposes, we include an extension of the type 2 maze code on page 12 in an attempt to eliminate more loops. This (type 3) maze creates large disconnects quite often, however – too many disconnects for general applications – so we only give a sample graph below and will not analyze or use this type of maze in applications.



Applications

For these applications, we will be analyzing type 1 and type 2 mazes of size n=31 (225 vertices). We will analyze the application of these mazes to a real-world situation. Consider the following scenario: A team of elite men is on an operation pursuing a deadly terrorist. The terrorist has been inside a building for a long time, but he gets informed that the SWAT team is coming for him. Shortly after, the pursuing team enters the building, having given the terrorist a 500 step head start. The SWAT team doesn't know the floor plan of the building, and the terrorist doesn't know which way to go to avoid the SWAT, and both teams move at random – the terrorist is trying to escape to the back exit and the other team is trying to apprehend the terrorist. If the terrorist makes it to the exit without running into a SWAT member, he will escape.

We will place the "terrorist" particle at the central vertex. These mazes have 15 vertices per dimension – if there are an odd number of vertices in each dimension there will be a central one. Let us also add an entrance and an exit to the maze. Because the vertices at the corners can have at most two connections, and all vertices must have at least two, we know that the corners of the maze with always have connections. This is useful to know, because it means we can place entrances and exits at these locations with complete confidence that they will connect to the graph. All particles will do random walks on the graph subject to the conditions we have imposed. How many men are required to guarantee with .9 probability that the terrorist will be caught? The code for this simulation is included on page 15. A maze with entrances and exits and 3 particles in pursuit mid-simulation is shown below:




And the results:

1 SWAT officer (30 runs):

Death fraction = 0.7667
Escape fraction =0.2333

2 SWAT officers (30 runs):

Death fraction =0.7667
Escape fraction =0.2333

3 SWAT officers (30 runs):

Death fraction =0.8333
Escape fraction =0.1667

4 SWAT officers (30 runs)
Death fraction =0.8333
Escape fraction =0.1667

5 SWAT officers (30 runs)
Death fraction =0.8333
Escape fraction =0.1667

6 SWAT officers (30 runs)
Death fraction =0.9333
Escape fraction =0.0667

So a team of SIZE 6 is sufficient, even in large buildings! Even a single SWAT officer would have a reasonable success rate. If we take dead ends to be locked doors, loops that link with the main graph via a single pathway to be rooms, and the rest of the pathways to be halls, our graphs model a typical building fairly accurately. Finally, we ask if the terrorist is more likely to reach the exit in a graph with no dead-ends, or one with dead-ends (even if they are short). We will consider only the case for 1 SWAT officer:

Death fraction=.6666
Escape fraction=.3333

This agrees with what we might expect. It is better to pursue a target inside a building with dead-ends, then. Simulations with these mazes could be used to predict the outcomes of counterterrorism missions or help plan them. For example, we have just shown that it is better to lure a terrorist into a building with locked doors to capture him! The applications of these data for fire rescue or SWAT operations are significant. Mazes of this type will also likely have use in studying traffic flow on city streets (including neighborhoods and parking lots and garages, because they are modified grids), as well as real-world searches of any kind. The possibilities are endless.

## Appendices

### Maze Generation Code

```
function [I, k, L1, L2] = mazegen3(n, S)
%N=10000; %number of steps
N = S;
%n = 57;%odd number - ((n-1)/2)^2 is number of vertices
I = zeros(n,n,N);
j = 0;
numLoops = zeros(1,N+1);
while 2*j<n-2
    j = j + 1;
    I(:,2*j,1) = 1;
    I(2*j,:,1) = 1;
end
I(1,:,1) = 0;
I(n,:,1) = 0;
I(:,n,1) = 0;
I(:,1,1) = 0;
numVerts = ((n-1)/2)*((n-1)/2);
numEdges = sum(sum(I(:,:,1))) - numVerts;
numLoops(1,1) = numEdges - numVerts + 1;

%THIS CREATES A TYPE 1 GRAPH

for k=2:N
    numVerts = ((n-1)/2)*((n-1)/2);
    numEdges = sum(sum(I(:,:,k-1))) - numVerts;
    numLoops(1,k-1) = numEdges - numVerts + 1;
    if mod(k,5000) == 0
        k;
    end
    I(:,:,k) = I(:,:,k-1);
    row=1;
    col=1;
    while
row==1||row==n||col==1||col==n||(mod(row,2)==0&&mod(col,2)==0)||(mod(row,2)==
1&&mod(col,2)==1)
        row=ceil(rand*n);
        col=ceil(rand*n);
    end
    I(row,col,k) = 0;
    for i=2:n-1
        for j=2:n-1
            Q = [i, j+1
                i+1, j
                i-1, j
                i, j-1];
            if
(I(Q(1,1),Q(1,2),k)+I(Q(2,1),Q(2,2),k)+I(Q(3,1),Q(3,2),k)+I(Q(4,1),Q(4,2),k))
<2
                %if
(I(Q(1,1),Q(1,2),k)+I(Q(2,1),Q(2,2),k)+I(Q(3,1),Q(3,2),k)+I(Q(4,1),Q(4,2),k))
==0
```

```
                %    I(row,col,k) = I(row,col,k-1);
                %elseif (rand>.25);
                    I(row,col,k) = I(row,col,k-1);
                %end
            end
        end
    end
end

%Now count the loops

%xxx = n*n - 4*(n-1)
%sum(sum(I(:,:,k)))
%(n-2)*(n-2) = area of maze
%(n-1)/2 = number of vertices in a row or col
numVerts = ((n-1)/2)*((n-1)/2);

numEdges = sum(sum(I(:,:,k))) - numVerts;
numLoops(1,N) = numEdges - numVerts + 1;
L1 = numLoops(1,N);
```

```
%THIS CREATES A TYPE 2 GRAPH

% if a loop is just a small square, break it. Increases diversity
% of graph without increasing probability of disconnecting it.
for i=2:n-1
    for j=2:n-1
        Q = [i, j+1
            i+1, j
            i-1, j
            i, j-1];
        F = [i, j+2
            i+2, j
            i-2, j
            i, j-2];
        C = [i-2, j-1
            i-2, j+1
            i-1, j-2
            i-1, j+2
            i+2, j-1
            i+2, j+1
            i-1, j-2
            i-1, j+2];
        if
(I(Q(1,1),Q(1,2),k)+I(Q(2,1),Q(2,2),k)+I(Q(3,1),Q(3,2),k)+I(Q(4,1),Q(4,2),k))
==4%if surrounded by connections
            if mod(i,2) == 1 || mod(j,2) == 1 %if it's an edge
                t = ceil(rand*4); %choose a random neighboring connection
(neighboring edge)
                I(Q(t,1),Q(t,2),k)=0; %and disconnect it(remove it), breaking
the loop.
                if
(I(F(1,1),F(1,2),k)+I(F(2,1),F(2,2),k)+I(F(3,1),F(3,2),k)+I(F(4,1),F(4,2),k))
==0
                    t=ceil(rand*8); %choose random connection
                    while C(t,1)==1||C(t,2)==1||C(t,1)==n||C(t,2)==n%making
sure you don't fill in a border
                        t=ceil(rand*8);%choose a random connection (edge)
that is probably going to connect the disjoint part of the graph
                    end
                    I(C(t,1),C(t,2),k)=1; %and connect it
                end
            end
        end
    end
end
```

```
%THIS CREATES A TYPE 3 GRAPH (does not work quite as expected)

Theory: If is a black edge, and sum around +2 circle surrounding it is 2,
%then it's a small loop (non-trivial) in the shape of a rectangle (I
%THINK) --- IDEA! Test only the 4 surrounding it in the outer +2 region (the
four +2 neighboring edges),
%and then break the one that =1 (there will only be one if it's this type
%of loop - the sum will be 1. Then locate the one that equals 1). This might
eliminate most of those loops altogether.
for i=2:n-1
    for j=2:n-1
        if I(i,j,k) == 0
            Q = [i, j+1
                i+1, j
                i-1, j
                i, j-1];
            if
(I(Q(1,1),Q(1,2),k)+I(Q(2,1),Q(2,2),k)+I(Q(3,1),Q(3,2),k)+I(Q(4,1),Q(4,2),k))
==3
                for t=1:4
                    if I(Q(t,1),Q(t,2),k)==0
                        row = Q(t,1);
                        col = Q(t,2);
                            while I(row,col,k) == 0&&1<row<n&&1<col<n
                                Q = [row, col+1
                                    row+1, col
                                    row-1, col
                                    row, col-1];
                                rowprev = row;
                                colprev = col;
%                               G = [I(Q(1,1),Q(1,2),k)==0
%                                   I(Q(2,1),Q(2,2),k)==0
%                                   I(Q(3,1),Q(3,2),k)==0
%                                   I(Q(4,1),Q(4,2),k)==0];
%                               if sum(G) == 3
%                                   for z=1:4
%                                       if G(z,1)==0
                                            row = Q(t,1);
                                            col = Q(t,2);
%                                       end
%                                   end
%                               end
                                if row==1||col==1||row==n||col==n
                                    break
                                end
                            end
                        if 1<row<n&&1<col<n
                            Q = [rowprev, colprev+1
                                rowprev+1, colprev
                                rowprev-1, colprev
                                rowprev, colprev-1];
                            if
(I(Q(1,1),Q(1,2),k)+I(Q(2,1),Q(2,2),k)+I(Q(3,1),Q(3,2),k)+I(Q(4,1),Q(4,2),k))
==3
                                r = rand;
                                if r > .5
                                   I(row,col,k) = 0;
```

```
                                else
                                    I(i,j,k) = 0;
                                end
                            end
                        end
                    end
                end
            end
        end
    end
end


%recalculate loops one more time, again, assuming graph is completely
%connected
numVerts = ((n-1)/2)*((n-1)/2);
numEdges = sum(sum(I(:,:,k))) - numVerts;
numLoops(1,N+1) = numEdges - numVerts + 1;
L2 = numLoops(1,N+1);

percentDec = (L1-L2)/L1;

%makes entrance and exit lower left and top right - NO! AT ALL CORNERS!
I(n,1,k) = 1;
I(n,2,k) = 1;
I(1,n-1,k) = 1;
I(1,n,k) = 1;

% I(1,1,k) = 1;
% I(1,2,k) = 1;
% I(n,n-1,k) = 1;
% I(n,n,k) = 1;
%axis equal
%colormap gray %(-1 = black, 0 = dark grey, 1 = light grey, 2 = white)
%(-1 = blue, 0 = cyan, 1 = yellow, 2 = red)
%imagesc(I(:,:,k))
%plot(numLoops);
end
```

Simulation (most comments are for modifying the number of particles. Currently the code will have 3 particles searching for 1):

```
function
[row1,col1,row2,col2,row3,col3,row4,col4,row5,col5,row6,col6,row7,col7] =
mazesimulation(n)
%n=31;
S = 10000;
[I, k, L1, L2] = mazegen3(n, S);
row1 = (n+1)/2;
col1 = (n+1)/2;
% row1 = n;
% col1 = 1;
%I(rowp,colp,k) = 2;
row2=0;
col2=0;
row3=0;
col3=0;
row4=0;
col4=0;
row5=0;
col5=0;
row6=0;
col6=0;
row7=0;
col7=0;
%rowg=1;
%colg=1;
%while mod(rowg,2) == 1||mod(colg,2)==2
%    rowg = ceil(rand*n);
%    colg = ceil(rand*n);
%end
%I(rowg,colg,k) = 3;
count = 0;
while
(row1~=row2||col1~=col2)&&(row1~=row3||col1~=col3)&&(row1~=row4||col1~=col4)&
&(row1~=row5||col1~=col5)&&(row1~=row6||col1~=col6)&&(row1~=row7||col1~=col7)
&&count<100000&&(row1~=1||col1~=n)%&&(rowp~=n||colp~=1))
            I(row1,col1,k) = 1;
        if row2~=0&&col2~=0&&row2~=n+1&&col2~=n+1
            I(row2,col2,k) = 1;
        end
        if row3~=0&&col3~=0&&row3~=n+1&&col3~=n+1
            I(row3,col3,k) = 1;
        end
        if row4~=0&&col4~=0&&row4~=n+1&&col4~=n+1
            I(row4,col4,k) = 1;
        end
        if row5~=0&&col5~=0&&row5~=n+1&&col5~=n+1
            I(row5,col5,k) = 1;
        end
        if row6~=0&&col6~=0&&row6~=n+1&&col6~=n+1
            I(row6,col6,k) = 1;
```

```
        end
        if row7~=0&&col7~=0&&row7~=n+1&&col7~=n+1
            I(row7,col7,k) = 1;
        end
    if count==51
        row2=n;
        col2=1;
        row3=n;
        col3=1;
        row4=n;
        col4=1;
%         row5=n;
%         col5=1;
%         row6=n;
%         col6=1;
%         row7=n;
%         col7=1;
%         row7=n;
%         col7=1;
%         row7=n;
%         col7=1;
    end
    B1 = [];
    B2 = [];
    B3 = [];
    B4 = [];
%     B5 = [];
%     B6 = [];
%     B7 = [];
%     B7 = [];
%     B7 = [];
    Q1 = [row1, col1+1
        row1+1, col1
        row1-1, col1
        row1, col1-1];
    Q2 = [row2, col2+1
        row2+1, col2
        row2-1, col2
        row2, col2-1];
    Q3 = [row3, col3+1
        row3+1, col3
        row3-1, col3
        row3, col3-1];
    Q4 = [row4, col4+1
        row4+1, col4
        row4-1, col4
        row4, col4-1];
%     Q5 = [row5, col5+1
%         row5+1, col5
%         row5-1, col5
%         row5, col5-1];
%     Q6 = [row6, col6+1
%         row6+1, col6
%         row6-1, col6
%         row6, col6-1];
%     Q7 = [row7, col7+1
%         row7+1, col7
```

```
%          row7-1, col7
%          row7, col7-1];
%      Q7 = [row7, col7+1
%          row7+1, col7
%          row7-1, col7
%          row7, col7-1];
%      Q7 = [row7, col7+1
%          row7+1, col7
%          row7-1, col7
%          row7, col7-1];
    for t=1:4
        if Q1(t,1)~=0&&Q1(t,2)~=0&&Q1(t,1)~=n+1&&Q1(t,2)~=n+1
            if I(Q1(t,1),Q1(t,2),k)~=0
                B1 = [B1 t];
            end
        end
    end
    r=ceil(rand*size(B1,2));
    row1 = Q1(B1(r),1);
    col1 = Q1(B1(r),2);
    if
(row1~=row2||col1~=col2)&&(row1~=row3||col1~=col3)&&(row1~=row4||col1~=col4)&
&(row1~=row5||col1~=col5)&&(row1~=row6||col1~=col6)&&(row1~=row7||col1~=col7)
        if count>50
            for t=1:4
                if Q2(t,1)~=0&&Q2(t,2)~=0&&Q2(t,1)~=n+1&&Q2(t,2)~=n+1
                    if I(Q2(t,1),Q2(t,2),k)~=0
                        B2 = [B2 t];
                    end
                end
            end
            r=ceil(rand*size(B2,2));
            row2 = Q2(B2(r),1);
            col2 = Q2(B2(r),2);
            for t=1:4
                if Q3(t,1)~=0&&Q3(t,2)~=0&&Q3(t,1)~=n+1&&Q3(t,2)~=n+1
                    if I(Q3(t,1),Q3(t,2),k)~=0
                        B3 = [B3 t];
                    end
                end
            end
            r=ceil(rand*size(B3,2));
            row3 = Q3(B3(r),1);
            col3 = Q3(B3(r),2);
            for t=1:4
                if Q4(t,1)~=0&&Q4(t,2)~=0&&Q4(t,1)~=n+1&&Q4(t,2)~=n+1
                    if I(Q4(t,1),Q4(t,2),k)~=0
                        B4 = [B4 t];
                    end
                end
            end
            r=ceil(rand*size(B4,2));
            row4 = Q4(B4(r),1);
            col4 = Q4(B4(r),2);
%             for t=1:4
%                 if Q5(t,1)~=0&&Q5(t,2)~=0&&Q5(t,1)~=n+1&&Q5(t,2)~=n+1
%                     if I(Q5(t,1),Q5(t,2),k)~=0
```

```
%                         B5 = [B5 t];
%                     end
%                 end
%             end
%             r=ceil(rand*size(B5,2));
%             row5 = Q5(B5(r),1);
%             col5 = Q5(B5(r),2);
%             for t=1:4
%                 if Q6(t,1)~=0&&Q6(t,2)~=0&&Q6(t,1)~=n+1&&Q6(t,2)~=n+1
%                     if I(Q6(t,1),Q6(t,2),k)~=0
%                         B6 = [B6 t];
%                     end
%                 end
%             end
%             r=ceil(rand*size(B6,2));
%             row6 = Q6(B6(r),1);
%             col6 = Q6(B6(r),2);
%             for t=1:4
%                 if Q7(t,1)~=0&&Q7(t,2)~=0&&Q7(t,1)~=n+1&&Q7(t,2)~=n+1
%                     if I(Q7(t,1),Q7(t,2),k)~=0
%                         B7 = [B7 t];
%                     end
%                 end
%             end
%             r=ceil(rand*size(B7,2));
%             row7 = Q7(B7(r),1);
%             col7 = Q7(B7(r),2);
%             for t=1:4
%                 if Q7(t,1)~=0&&Q7(t,2)~=0&&Q7(t,1)~=n+1&&Q7(t,2)~=n+1
%                     if I(Q7(t,1),Q7(t,2),k)~=0
%                         B7 = [B7 t];
%                     end
%                 end
%             end
%             r=ceil(rand*size(B7,2));
%             row7 = Q7(B7(r),1);
%             col7 = Q5(B7(r),2);
%             for t=1:4
%                 if Q7(t,1)~=0&&Q7(t,2)~=0&&Q7(t,1)~=n+1&&Q7(t,2)~=n+1
%                     if I(Q7(t,1),Q7(t,2),k)~=0
%                         B7 = [B7 t];
%                     end
%                 end
%             end
%             r=ceil(rand*size(B7,2));
%             row7 = Q7(B7(r),1);
%             col7 = Q5(B7(r),2);
        end
    count = count + 1
    end
I(row1,col1,k) = 2;
if row2~=0&&col2~=0&&row2~=n+1&&col2~=n+1
    I(row2,col2,k) = 3;
end
if row3~=0&&col3~=0&&row3~=n+1&&col3~=n+1
    I(row3,col3,k) = 3;
end
```

```
if row4~=0&&col4~=0&&row4~=n+1&&col4~=n+1
    I(row4,col4,k) = 3;
end
%     if row5~=0&&col5~=0&&row5~=n+1&&col5~=n+1
%         I(row5,col5,k) = 3;
%     end
%     if row6~=0&&col6~=0&&row6~=n+1&&col6~=n+1
%         I(row6,col6,k) = 3;
%     end
%     if row7~=0&&col7~=0&&row7~=n+1&&col7~=n+1
%         I(row7,col7,k) = 3;
%     end
    axis equal
    imagesc(I(:,:,k));
end
% if rowp==rowg&&colp==rowg
%     die = 1;
% elseif (rowp==n&&colp==n)||(rowp==1&&colp==1)
%     escape = 1;
% end
if row2~=0&&col2~=0&&row2~=n+1&&col2~=n+1
    I(row2,col2,k) = 3;
end
if row3~=0&&col3~=0&&row3~=n+1&&col3~=n+1
    I(row3,col3,k) = 3;
end
if row4~=0&&col4~=0&&row4~=n+1&&col4~=n+1
    I(row4,col4,k) = 3;
end
% if row5~=0&&col5~=0&&row5~=n+1&&col5~=n+1
%     I(row5,col5,k) = 3;
% end
% if row6~=0&&col6~=0&&row6~=n+1&&col6~=n+1
%     I(row6,col6,k) = 3;
% end
% if row7~=0&&col7~=0&&row7~=n+1&&col7~=n+1
%     I(row7,col7,k) = 3;
% end
% I(row7,col7,k) = 3;
% I(row7,col7,k) = 3;
I(row1,col1,k) = 2;
axis equal
imagesc(I(:,:,k));
```

The test code that implements the maze generation Markov Chain algorithm is below

```
N = 1;
numLoops1 = zeros(1,N); %num before modifying
numLoops2 = zeros(1,N); %num after modifying
for i=1:N
    i
 [I, k, L1, L2] = mazegen3(51, 10000);
 numLoops1(1,i) = L1;
 numLoops2(1,i) = L2;
end
avg1 = mean(numLoops1)
avg2 = mean(numLoops2);
x = avg1-avg2;
x/avg1;
%note that the difference avg1-avg2 gives the average number of small
%square loops (call them trivial loops).
%hist(numLoops1)
%avg = 12.18 loops for size n=21 (10x10 vertices), 10000 steps eech, 50 runs
%avg = 28.38 for n=31 (15x15)
axis equal
colormap gray
imagesc(I(:,:,k))
```

And the code controlling the simulations:

```
N = 30;
n=31;
die = 0;
escape = 0;
for i=1:N
    i
    [row1,col1,row2,col2,row3,col3,row4,col4,row5,col5,row6,col6,row7,col7] =
mazesimulation(n);
    if
(row1==row2&&col1==col2)||(row1==row3&&col1==col3)||(row1==row4&&col1==col4)|
|(row1==row5&&col1==col5)||(row1==row6&&col1==col6)||(row1==row7&&col1==col7)
        die = die + 1;
    elseif (row1==1&&col1==n)%||(rowp==n&&colp==1)
        escape = escape + 1;
    end
end

row1;
col1;
dfrac=die/(die+escape) %Small chance of not summing to 1 because the
efrac=escape/(die+escape) %simulation stops after a certain number of steps
```