**1，Description of the model**

IFRN's main function is to categorize all cells into several groups, with N cells in each group, and each cell in a group using different frequencies to reduce interference between them. M-cell IFRN transforms the original program, which could only handle a fixed number of cells and reuse factors, into a program capable of handling generic data.

**2，The main content of the program is as follows:**

2.1 Initialization and User Input:

The code begins by clearing previous variables and displaying basic information, including the program's name, author, and date. Then, the user is prompted to input parameters.

2.2 Creating Cell Layout:

The code uses principles of geometry to establish proximity relationships between cells. It generates a coordinate list representing the positions of cell centers and calculates the proximity relationships between cells. This ensures that the distance between them is sufficiently large to reduce interference.

The primary generation method for this part includes:

2.2.1. Generating the First Cell Center: Initially, we generate the center of the first cell at the origin (0, 0).

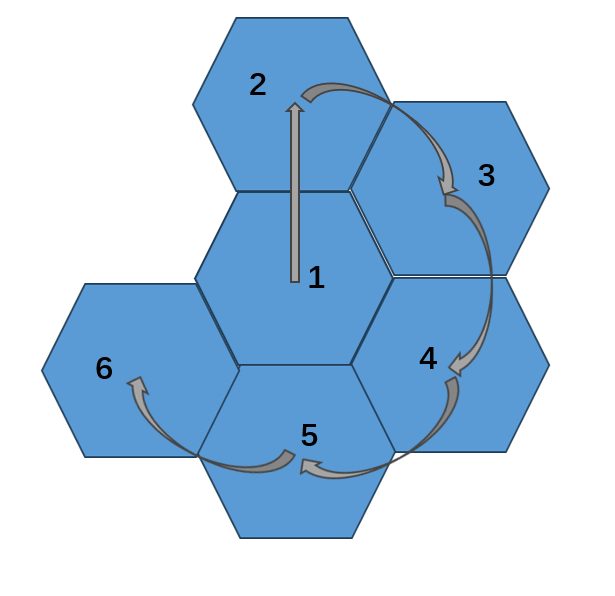
2.2.2. Defining Polar Coordinate Parameters: To generate the centers of other cells, we use a polar coordinate system. Two parameters are defined:

- Radii: The radius from the origin to the cell center, typically set to 1.

- Angles: An array containing six elements representing six directions starting from the origin, each separated by 60 degrees.

2.2.3. Generating the Centers of Other Cells: Using the parameters defined above, we employ a loop to sequentially generate the centers of cell centers. We continuously attempt to generate new x and y coordinates until a suitable position is found. These coordinates are located on concentric circles centered around the current cell center and are calculated based on the polar coordinate parameters. We then check whether the newly generated coordinates are too close to the existing cell centers. If they are, we consider the position too close and regenerate the coordinates.

2.2.4. Proximity Relationships and Distance Check: After generating the centers of all cells, we proceed to calculate proximity relationships between cells. We iterate through all pairs of cells, calculate their Euclidean distances, and mark cells with distances smaller than the `min\_distance\_threshold` as neighboring cells. The corresponding proximity relationships are recorded in a matrix.



**fig1. The approximate sequence of generating cells**

2.3 Frequency Allocation:

The code assigns frequencies to each cell. It starts by searching for frequencies used by neighboring cells of each cell and selects an unused frequency if available. If no available frequencies are found, a frequency is randomly chosen. This allocation strategy aims to minimize interference by ensuring neighboring cells use different frequencies, thus enhancing communication efficiency.

2.4 User Capacity Calculation:

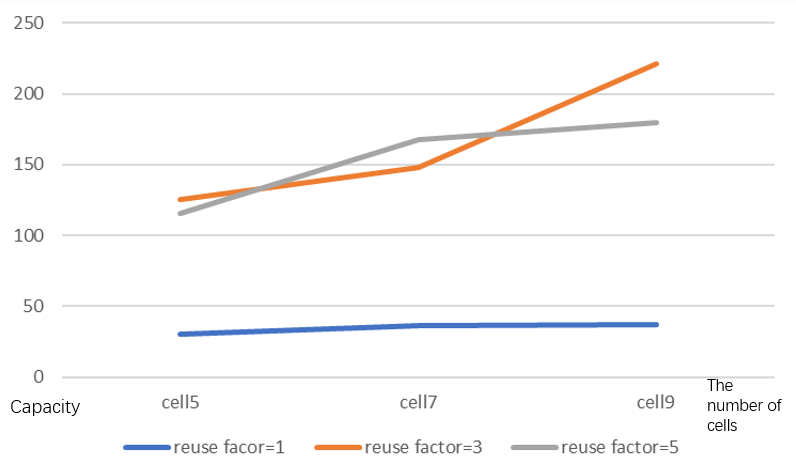
The code computes user capacity for each cell. It considers interference and the channel conditions of each user, resulting in the calculation of each cell's total capacity and the overall network capacity.

2.5 Output of Results:

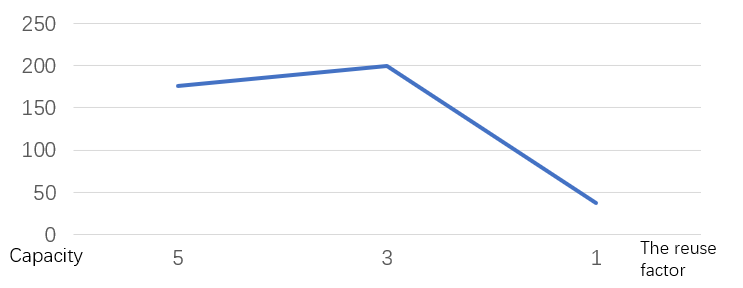
Finally, the code outputs the assigned frequencies for each cell and the total network capacity.

**3，Results**

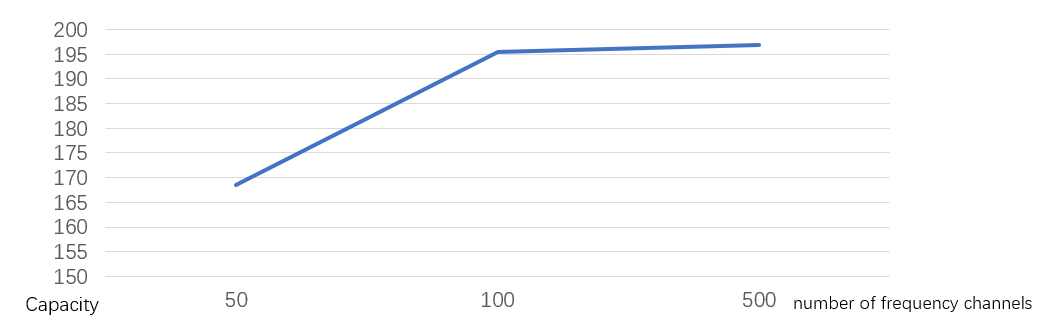
3.1， The relationship between capacity and the number of cells under different reuse factor scenarios.



3.2，The relationship between capacity and the reuse factor



3.3, The relationship between capacity and the number of frequency channels



**4，Matlab codes**

function final()

clc

clear all

disp('----------------------------------------------------------------------')

disp('-------------------------final presentation----------------------------')

disp('-------------------by Amber-GaoQi on 23/9/2---------------------')

disp('-----------------------------------------------------------------------')

%%

M=input('the number of cells: ');

N=input('reuse factor: ');

Nc=input('the number of frequency channels: ');

Um=input('average number of users in each cell: ');

snrdB=input('average SNR for each user in dB: ');

snr=10^(snrdB/10);

P=1;

sigma=sqrt(P/snr);

%% Create an adjacency matrix to represent neighboring relationships

min\_distance\_threshold = 0.9999999; % Minimum distance between cell centers

% Create an adjacency matrix to represent neighboring relationships, initially set to 0

adjacency\_matrix = zeros(M, M);

% Initialize lists for center point coordinates

x\_center\_positions = zeros(1, M);

y\_center\_positions = zeros(1, M);

% Generate the first center point (0, 0)

x\_positions(1) = 0;

y\_positions(1) = 0;

% Define polar coordinate parameters

radii = 1; % Radius

angles =[0, pi/3, 2\*pi/3, pi, 4\*pi/3, 5\*pi/3]; % Angles at 60-degree intervals

% Generate the remaining center points

for i = 2:M

n=2;

found\_position = false; % Add a flag variable to control the outer loop

while found\_position == false

% Generate x and y points, ensuring they are integers, around the origin in concentric circles

h=1+floor((i-2)/6);

if mod(n, 6) == 1

x\_positions(n) = x\_center\_positions(h)+(radii \* sin(angles(1)));

y\_positions(n) = y\_center\_positions(h) + (radii \* cos(angles(1)));

elseif mod(n, 6) == 2

x\_positions(n) = x\_center\_positions(h)+(radii \* sin(angles(2)));

y\_positions(n) = y\_center\_positions(h) + (radii \* cos(angles(2)));

elseif mod(n, 6) == 3

x\_positions(n) = x\_center\_positions(h)+(radii \* sin(angles(3)));

y\_positions(n) = y\_center\_positions(h)+ (radii \* cos(angles(3)));

elseif mod(n, 6) == 4

x\_positions(n) = x\_center\_positions(h)+(radii \* sin(angles(4)));

y\_positions(n) = y\_center\_positions(h) + (radii \* cos(angles(4)));

elseif mod(n, 6) == 5

x\_positions(n) = x\_center\_positions(h)+(radii \* sin(angles(5)));

y\_positions(n) = y\_center\_positions(h)+ (radii \* cos(angles(5)));

elseif mod(n, 6) == 0

x\_positions(n) = x\_center\_positions(h)+(radii \* sin(angles(6)));

y\_positions(n) = y\_center\_positions(h)+(radii \* cos(angles(6)));

end

% Check the distance between this center point and existing center points

too\_close = false;

for j = 1:i-1

distance = sqrt((x\_positions(n) - x\_center\_positions(j))^2 + (y\_positions(n) - y\_center\_positions(j))^2);

if distance < min\_distance\_threshold

too\_close = true;

n=n+1;

break;

end

end

if ~too\_close

% If the distance is suitable, add the coordinates of this center point to the list

x\_center\_positions(i) = x\_positions(n);

y\_center\_positions(i) = y\_positions(n);

n=n+1;

found\_position = true; % Set the flag variable to true

break;

end

if n > 1000 % Add a maximum number of attempts to prevent an infinite loop

error('Unable to generate suitable positions.');

end

end

if ~found\_position

error('Unable to generate suitable positions.');

end

end

% Output the positions of each cell

% disp('Cell positions:');

% disp([x\_center\_positions; y\_center\_positions]);

% Calculate neighboring relationships based on cell positions

for i = 1:M

for j = 1:M

if i ~= j

% Calculate the distance between cell i and cell j

distance = sqrt((x\_center\_positions(i) - x\_center\_positions(j))^2 + (y\_center\_positions(i) - y\_center\_positions(j))^2);

% If the distance is less than the threshold, set the corresponding position in the adjacency matrix to 1

if distance < 1.1

adjacency\_matrix(i, j) = 1;

end

end

end

end

% Output the adjacency matrix

disp('Adjacency matrix:');

disp(adjacency\_matrix);

%%

% Initialize a frequency list

fr = 1:N;

% Create an array to store the frequency used by each cell

grid\_fr = zeros(1, M);

% Start the allocation process

for i = 1:M

% Find the frequencies used by neighboring cells of the current cell

neighbor\_fr = grid\_fr(adjacency\_matrix(i, :) == 1);

% Find available frequencies that have not been used

available\_fr = setdiff(fr, neighbor\_fr);

% Randomly select an available frequency and assign it to the current cell

if ~isempty(available\_fr)

selected\_fr = available\_fr(randi(length(available\_fr)));

grid\_fr(i) = selected\_fr;

else

% If no available frequencies are found, select any frequency randomly

selected\_fr = fr(randi(length(fr)));

grid\_fr(i) = selected\_fr;

end

end

% Print the frequencies used by each cell

for i = 1:M

disp(['Cell ' num2str(i) ' uses frequency f' num2str(grid\_fr(i))]);

end

%%

% First, set all diagonal elements of the adjacency\_matrix to 1.

% Then, check the positions j in columns of adjacency\_matrix(i, :) == 1 representing cell j

% If they have the same frequency as cell i, change the 1 at position (i,j) in adjacency\_matrix to r

r = 0.7;

% Set diagonal elements from 0 to 1

adjacency\_matrix = adjacency\_matrix + eye(M);

% Check positions in each row where adjacency\_matrix(i, :) == 1

for i = 1:M

neighbors = find(adjacency\_matrix(i, :) == 1); % Find indices of neighboring cells

current\_frequency = grid\_fr(i); % Get the current cell's frequency

for j = neighbors

if i ~= j % Ensure that i and j are not the same cell

if grid\_fr(j) == current\_frequency % If the frequencies are the same

adjacency\_matrix(i, j) = r; % Change the 1 at (i, j) to r

end

end

end

end

rho = adjacency\_matrix;

% Traverse the adjacency matrix and change 1 to 0 except for the diagonal elements

for i = 1:M

for j = 1:M

if i ~= j && rho(i, j) == 1

rho(i, j) = 0;

end

end

end

%%

Nc\_ = zeros(1, N); % Initialize the Nc\_ array

for i = 1:N

if i ~= N

Nc\_(i) = round(Nc / N);

else

% Calculate Nc\_(N): Nc minus the sum of all previous Nc\_(i)

Nc\_(N) = Nc - sum(Nc\_(1:N-1));

end

% Generate a part of f(N, 1:Nc\_(N))

if i == 1

g(i, 1:Nc\_(i)) = 1:Nc\_(i);

else

g(i, 1:Nc\_(i)) = sum(Nc\_(1:i-1)) + 1:sum(Nc\_(1:i-1)) + Nc\_(i);

end

end

for i=1:M

num=grid\_fr(i);

f(i, 1:Nc\_(num))=g(num,1:Nc\_(num));

end

% Setting the number of active users for all the cells

for c=1:M

U(c)=poissrnd(Um);

Nchannel(c)=length( find( f(c , : )>0));%check

if U(c)>Nchannel(c)

fprintf('We cannot serve all the %d users in cell %d\n',Nchannel(c),c)

end

end

% Generate fading channels for each serving user from all base stations

for c1=1:M

for c2=1:M

for u=1:min(Nchannel(c2),U(c2))

h(c1,c2,u)=(randn+1i\*randn)/sqrt(2); % From cell c1 to user u in cell c2

end

end

end

% Compute the user capacity for each cell

for c=1:M

for u=1:min(Nchannel(c),U(c))

I(c,u)=0;

for c\_=1:M

if rho(c\_,c)~=0&&c\_~=c

if U(c\_)>=u&&Nchannel(c\_)>=u

I(c,u)=I(c,u)+rho(c\_,c)\*P\*abs(h(c\_,c,u))^2;

end

end

end

sinr(c,u)=P\*abs(h(c,c,u))^2/(I(c,u)+sigma^2);

capacity(c,u)=log2(1+sinr(c,u));

end

cell\_capacity(c)=sum(capacity(c, : ));

end

total\_net=sum(cell\_capacity);

fprintf('Capacity is %g\n',total\_net);

end