MCC Summer School 2005 — Genetic Algorithm Lab

Gus Hart, Dept. of Physics and Astronomy, Northern Arizona University

June 22, 2005

1 Overview

Genetic algorithms, or more generally "evolutionary approaches," can be effective in solving optimization problems for highly correlated problems in a discrete search space. In today's lecture we described how an evolutionary approach can be applied to the problem of truncating a cluster expansion. For your lab, you will write a program to search a simple Hamiltonian using a genetic algorithm. In principle, this lab doesn't have anything to do with cluster expansion specifically but that doesn't matter. You will take exactly the same approach as used in optimizing the cluster expansion—you'll just apply it to a case that is simpler to handle. It's not in anyway a "dumbed down" version of the problem.

You've been given several MATLAB files to build your program around. If you prefer to program in MATHEMATICA, C, C++, Fortran90/95, PERL, Python, etc., you should find no problem translating the MATLAB files to another language—they are pretty simple.

2 Today's model Hamiltonian

The Hamiltonian that we will use in the program for this lab is a simple one, easy to conceptualize, but with a rugged energy landscape so that it will be hard to guess the answer intuitively and hard to search by more "standard" search techniques.

Here's our model: consider an $N \times N$ square lattice. Particles can sit on the lattice sites and they will interact with each other. The interaction can be negative (favorable, attractive) or positive (repulsive). The type and strength of the interaction of two particles will depend, not on distance, but on where two particles are sitting. Said another way, every site is connected (coupled) to every other site. Some combinations of sites have favorable interactions and some have repulsive interactions.

In math, the energy of your system is:

$$E(\vec{\xi}) = (\xi_1, \xi_2, \cdots, \xi_N) \begin{pmatrix} J_{1,1} & J_{1,2} & \cdots & J_{1,N} \\ J_{2,1} & J_{2,2} & \cdots & J_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ J_{N,1} & J_{N,2} & \cdots & J_{N,N} \end{pmatrix} \begin{pmatrix} \xi_1 \\ \xi_2 \\ \vdots \\ \xi_N \end{pmatrix} \tag{1}$$

where $\vec{\xi}$ represents the occupation of sites—a string of zeroes and ones (an entry for each site on the lattice)—and $\{J_{i,j}\}$ are the coupling constants (i.e., the strengths of the interaction between the ith and jth sites). Note how easy this is to program. Given a set of $\{J_{i,j}\}$'s, this is a one-liner in a "matrix/vector-friendly" language like MATLAB or Fortran90/95.

3 The Optimization problem

The optimization problem we aim to solve, of course, is to minimize the energy. Specifically, we have N^2 sites, and given some number of particles $N_{\text{part}} < N^2$, where should we place the particles to minimize the energy?

We'll approach this problem with a genetic algorithm. Start with a "population" of randomly generated guesses (candidate solutions), evaluate the energy of each one, selectively pick individuals to "mate," using any kind of crossover generate "offspring," and iterate until you have found the minimum.

4 What you have been given

Several MATLAB m-files have been provided and are shown here. Electronic copies of these files are on lab web page.

The following file sets up the matrix of $\{J_{i,j}\}$'s:

```
function [connections] = initialize_hamiltonian(N)
connections = rand(N,N)-1/2;  % Random matrix, each entry between -1/2 and 1/2
connections = (connections + connections');% Make the matrix symmetric
                                           % Not necessary, but physically
                                           % a very sensible thing...
```

And this file computes the energy, given an occupation vector:

```
function energy = compute_energy(conn_mat,occ)
% Compute the energy of a NxN fully-connected chain

energy = occ*conn_mat*occ' ;
```

This program, that uses the two preceding function m-files, just calculates the energy for all possible combinations of site occupations, and then makes a histogram plot. Use it as an example of how to use the two functions. (Don't set N too big or it will run forever...)

```
% Funny hamiltonian for practicing searches (simulated annealing, genetic
% algorithm, etc.)
clc

%rand('state',0);  % Restart the random number generator if you wish
k = 0;  % Number of permutations found so far
N = 16; % Number of sites in the model
Nocc = ceil(N/2); % Number of occupied sites

Jmatrix = initialize_hamiltonian(N); % Set up the interactions
Nperms = nchoosek(N,Nocc);  % How many possible permutations
fprintf('Search space: %d\n',Nperms)
energies = zeros(1,Nperms);  % Initialize the array for energies
for i =1:2^N % In this loop is a simple-minded, brute-force method
             % for generating all possible permutations of 0's and 1's
    binstr = dec2bin(i);           % Convert the number to binary
    occupation = str2num(binstr'); % Convert it to a string
    occupation = [zeros(N-length(occupation),1); occupation]; % Pad with extra zeroes
    if Nocc~=sum(occupation) % Skip over cases with the
        continue             % wrong number of occupied sites
    end
    k = k + 1;
    energies(k) = compute_energy(Jmatrix,occupation');
end

hist(energies,20)
```

Finally, you don't need this, but here is a program that optimizes using not a genetic algorithm but a simulated annealing (Monte Carlo) approach and compares the answer to a brute-force, direct enumeration approach.

```
% Application of simulated annealing to our "funny hamiltonian"
clc

%rand('state',0);  % Restart the random generator if you want
k = 0; % Number of permutations found
N = 12; %Number of sites in the model
Jmatrix = initialize_hamiltonian(N); % Set up the interactions
Nocc = ceil(N/2); % Number of occupied sites
Nperms = nchoosek(N,Nocc);
fprintf('Search space: %d\n',Nperms)
energies = zeros(1,Nperms);
for i =1:2^N % In this loop is a simple-minded, brute-force method
             % for generating all possible permutations of 0's and 1's
    binstr = dec2bin(i);
    occupation = str2num(binstr');
    occupation = [zeros(N-length(occupation),1); occupation];
    if Nocc~=sum(occupation)
        continue
    end
    occupation;
    k = k + 1;
    energies(k) = compute_energy(Jmatrix,occupation');
end

oldguess = [zeros(N-Nocc,1); ones(Nocc,1)];
k = 0; %iteration counter
kT = 6; % Temperature
dt = .99;

energy = compute_energy(Jmatrix,oldguess');
%while kT>1e-5
while k < 500
    site1=ceil(rand(1)*N); % Select two particle
    site2=ceil(rand(1)*N); % locations to switch
    while oldguess(site1)==oldguess(site2)  % Keep selecting sites till we've selected bot
        site2=ceil(rand(1)*N);
    end
    newguess = oldguess;
    temp = oldguess(site1);
    oldguess(site1) = oldguess(site2);
    oldguess(site2) = temp;
    deltaE = compute_energy(Jmatrix,oldguess') - energy;
    if deltaE < 0
        newguess = oldguess;
        %disp('jump down')

    elseif rand(1) < exp(-deltaE/kT)
        newguess = oldguess;
        %disp('jump up')
    else
        oldguess = newguess;
        %disp('no jump')
    end
    k = k + 1;
    energy = compute_energy(Jmatrix,newguess');
    MCe(k) = energy;
    kT = kT*dt;
    newguess';
end
plot(MCe,'.-')
%min(energies)
%energy
fprintf('Global min: %15.7f\n Found min: %15.7f\n',min(energies),energy)
if (min(energies)<energy)
    disp('Not converged')
else
    disp('Converged')
end
```

5 Things to think about and to try

Before you start programming, consider this last detail—the model we have described above is not really constrained to a 2D lattice. Actually, it's just a one-dimensional "chain" of sites, but each site is linked to every other site. Making this realization may make it easier for you to write the program.

1. Before writing your program, you may want to run the two examples first and experiment around a little with them. They illustrate how to use the functions provided (`compute_energy` and `initialize_hamiltonian`).
2. Write a program that uses the Hamiltonian described above and employs a genetic algorithm to find the state of lowest energy.
3. How can you test to see if you are finding a global minimum? And not just a "local" minimum?
4. Does your success rate decrease with increasing system size?
5. What fraction of the solution space is explored by your program before it finds the minimum?
6. How is the performance of the GA affected by mutation rate, population size, etc.?

Extra things to try:

1. Instead of fixing the number of particles, let the number of particles vary. What number of particles and which arrangement of particles minimizes the energy? The number of possibilities is *much* higher in this case. But is the problem harder, or easier? How can you be sure?
2. How does the performance of the GA compare to simulated annealing?
3. Can you modify the Hamiltonian somehow to make it more "correlated" and thus harder for a simulated annealing approach?