Protein folding: HP model

Robin Emanuelsson

October 2018

Introduction

In nature there are 20 different amino acids that make up the proteins. After a protein has been made it folds in to a self avoided native state where the folded protein can be seen as a "key" for a certain "keyhole"in the body. If the "key" doesn't fit the protein wont do it's job. It has been proposed that the native state of the folded protein is the state with the lowest energy.
The HP model (H for hydrophobic and P for polar) is a widely used representation for protein folding. The amino acids are either a H or a P. The protein's will folds such that the energy is minimized, where the energy function is such that neighbouring non-connected H amino acids lowers the energy by -1. The model is based on the fact that it presumes that some amino acids are hydrophobic so they want to avoid water and the protein should, by this will to avoid water, fold in to the native state.

The algorithms

Monte Carlo simulations is commonly used in the understanding of thermodynamical problems. The folding of a protein can be seen as a phase change problem from a thermodynamic point of view. The simulations is a number of random, in our cases pivots, and the step is accepted with a probability that depends on the energy function. That is

$$\begin{aligned} &\text{if } E_2 \leq E_1 \text{ accept} \\ &\text{if } R < \exp(-(E_2 - E_1)\beta) \text{ accept} \\ &\text{else reject} \end{aligned} \tag{1}$$

where R is a random number between 0 and 1, β is number that is proportional to the temperature in the system and the energies are calculated with the number of non-connected neighbouring H amino acids.
In order for the energy to converge the temperature was lowered each step so that the probability to make a fold even though it wasn't energy efficient got lower for each iteration. The use probability for the system to make a fold in to a higher energy is so that the system can escape local minimums.

To find the degenerate energies a trick from statistical mechanics is used, namely that the distribution should be a Boltzmann distribution.

$$\frac{n_i}{N} = \frac{1}{Z} g \exp(-(E_i)\beta) \tag{2}$$

n_i is the number of a certain energy i we measure, N is the sum of all the number of energies we measure, g is the number of degeneracies for a given energy. We don't know

Z or g. But we do know what the sum over all g's is (the number of self avoiding walks) which we can use to renormalize to get Z and in turn g.

Results

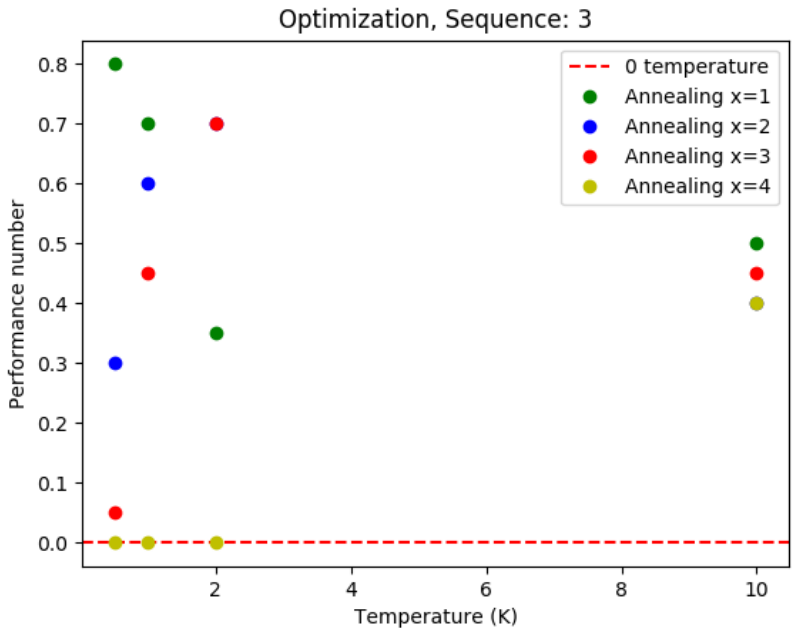
From the manual it was given

Tabell 1: Sequences with their sequence corresponding ground energy and degeneracy number.

Sequence	Seq. Number	Ground Energy	Degeneracy
HHPHHPHPPHHPHPPHPPPHHPHPH	1	-10	1
PHHHHPHHHPPPPPHHHPHHHPHHH	2	-11	1
PHHHPPPHPPHHPHHPPPPPHPPHH	3	-9	1
HPPHPHPHHHPPPPPHPHPHPHHPH	4	-10	14
HPPPHPHPHPPHPPPHHHPPPHHPP	5	-7	10

Annealing optimization

To find what temperature and way of lowering this temperature we ran the simulation 40 times, with N=30000 attempts at folding, for the five different sequences (note that the name in the figure is one less than then number in the table).
The result is how often we came to the know ground state. In the simulation we tried for zero temperature, lowering the temperature every fold with step $T_+ = T\frac{N-x}{N}$, where N is the number of folds and with the starting temperatures $T = 0.5, 1, 2, 10$



Figur 1: Performance, in percent of finding the know ground state, for the fourth sequence with the different temperatures and ways of lowering the temperature.

$(N-x)/N$	P $T = 0.5$ K	P $T = 1$ K	P $T = 2$ K	P $T = 10$ K
0 ($T = 0$ K)	0	0	0	0
1	0.8	0.7	0.35	0.5
2	0.3	0.6	0.35	0.5
3	0.05	0.45	0.7	0.45
4	0	0	0	0.4

Tabell 2: Table for the performance of the fourth sequence.

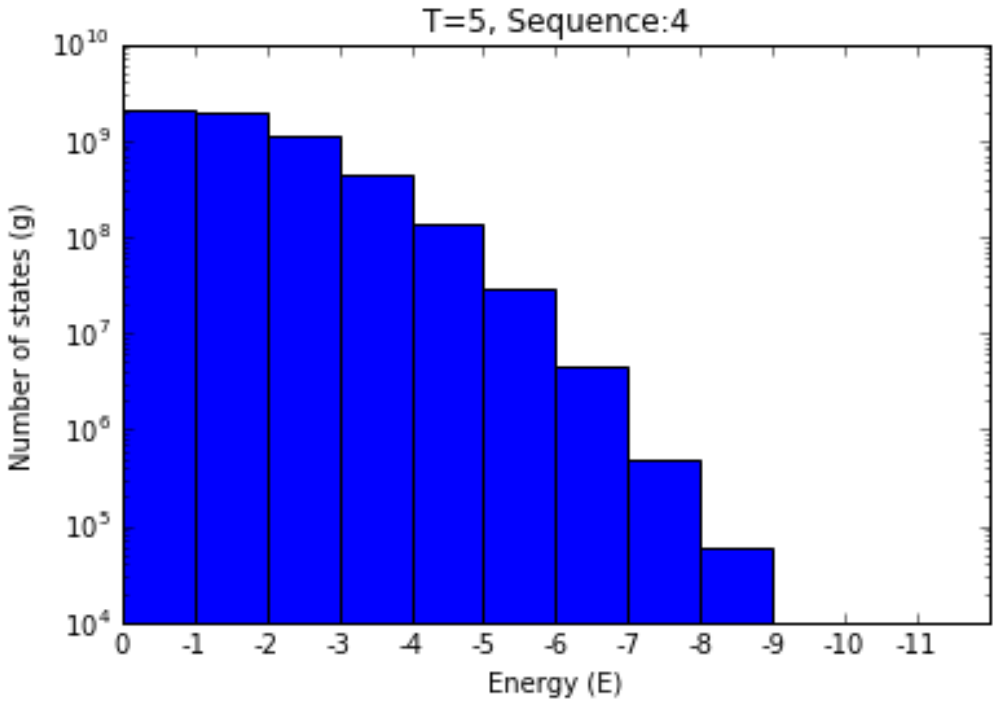
The anneling with $x = 3$ and for $T = 0.5$ performed the best, that is was most likely to find the ground state, but only for the said temperature. The overall best, with the given temperatures, was for $x = 1$. The rest of the sequences can be found in appendix A

Finding degeneracies

To find the degeneracies we used equation 1 for our regular simulation, with N=5000000 tries to pivot the sequence, with $T = 5$ With this information we use equation 2 to get the distribution of degenerate states. This was used with the five sequences in table 1. One of the ten histograms (the rest can be found in appendix B) is

Energy	g	n
0	$2.047435804 \cdot 10^9$	$2.160469 \cdot 10^6$
-1	$1.968795908 \cdot 10^9$	$1.700903 \cdot 10^6$
-2	$1.130586733 \cdot 10^9$	$7.99694 \cdot 10^5$
-3	$4.53454447 \cdot 10^8$	262600
-4	$1.33858455 \cdot 10^8$	63467
-5	$2.9029671 \cdot 10^7$	11269
-6	4578029	1455
-7	499594	130
-8	61020	13
-9	0	0

Tabell 3: The number of times n each energy was found in the fourth sequence and the calculated degeneracy for temperature $T = 5$



Figur 2: Histogram over the calculated degeneracies for the fourth sequence.

Finding designing sequences

Using the standard annealing equation 2 with, and a linear annealing (which was not found to be the best but still found the designing sequences), we tried to find designing sequences (sequences with only one unique ground states and in this case an energy of E=-11). This simulation also included a sorting algorithm so sort out identical sequences. The sequences were

$$0, 1, 0, 0, 1, a, 1, b, 1, 1, 1, 1, 1, 0, c, 1, d, 1, 1, 0, 0, 1$$

where a, b, c, d can be either a 0 or a 1, so sixteen different sequences in total. In the manual it was said that there was five different. The sequences that were found to be design sequences were

$$[a, b, c, d] : [0, 0, 0, 0], [0, 0, 0, 1], [0, 0, 1, 0], [0, 1, 0, 0], [1, 0, 0, 0] \tag{3}$$

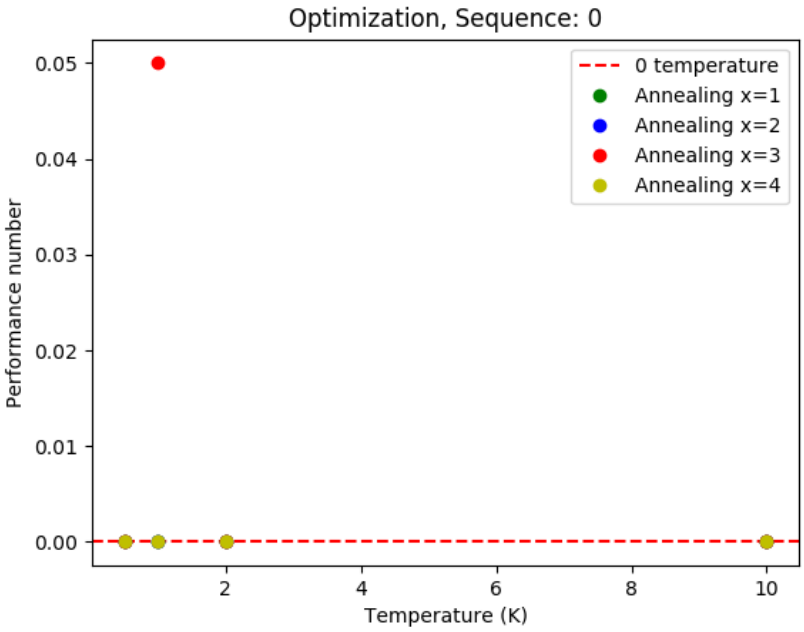
The simulation found two ground states but as it turned out they were the same but mirrored.

Discussion

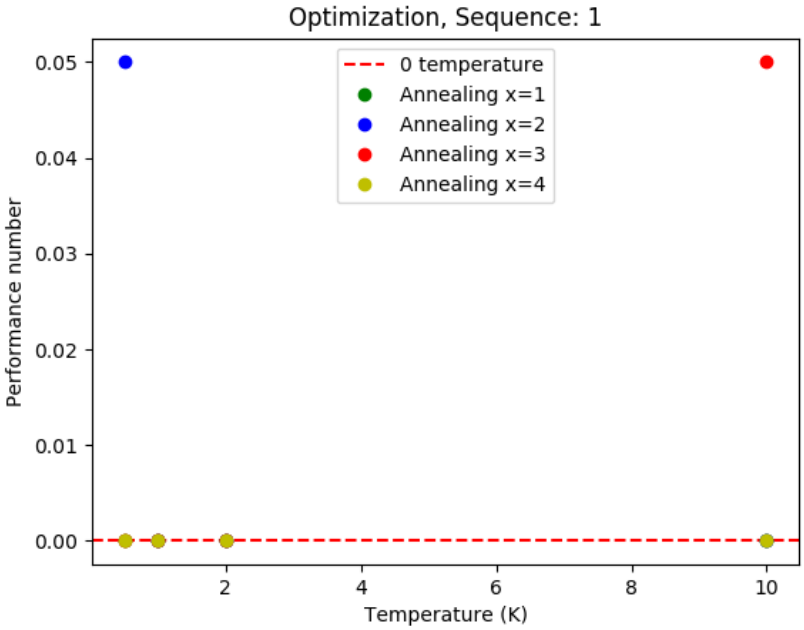
The very nature of the problems makes it very hard to get good results. The choice of T is quite tricky and the folding code take a long time. Simulation times of several hours is to expected which makes it hard try different parameters and see if the code is working as intended.
For the number of degeneracies we weren't able to find the ground state. This could possibly be solved by running more simulations and trying different temperatures.

For the optimization, different exponentially ways of lowering the temperature and doing more folding and using more initial temperatures could give a better result.
Finding the designing sequencers was successful, since we found manually that the two ground states which were found was the same. Adding a code to sort out mirror images could have been used. The best way of lowering the temperature was not use because it was successful and the best way of lowering the temperature was still not found.

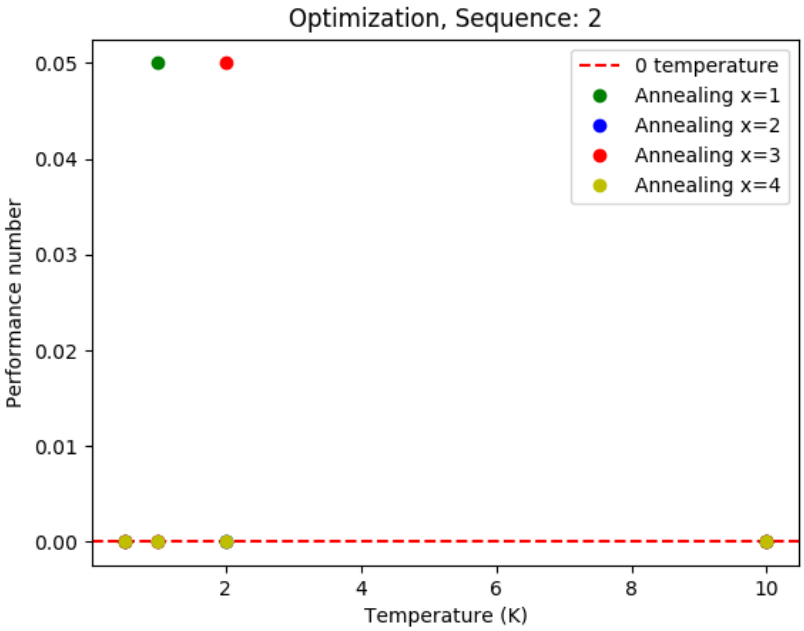
Appendix A



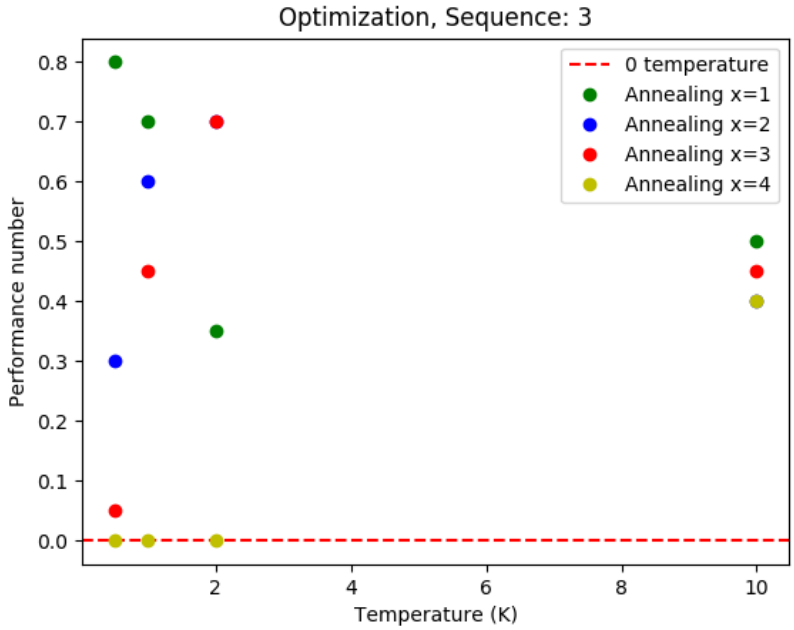
Figur 3: Performance for different T and different ways of lowering T for the first sequence.



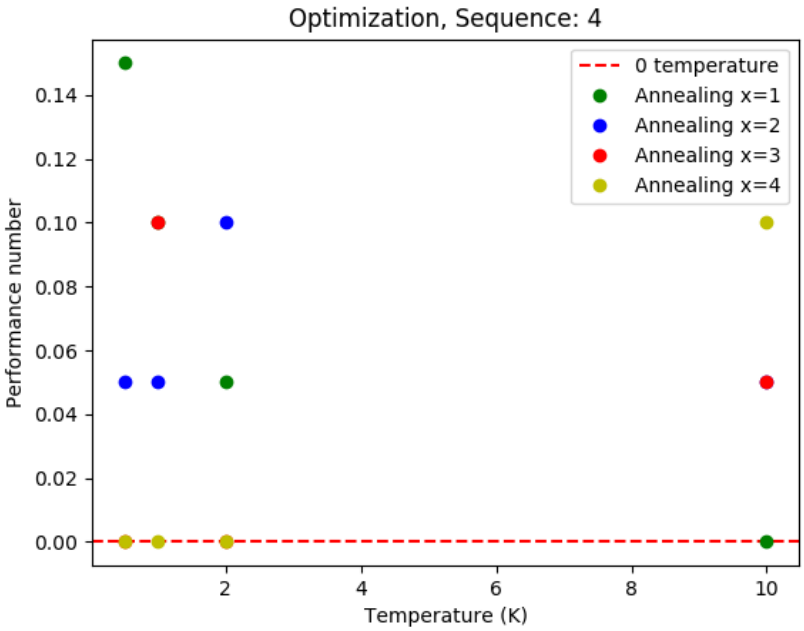
Figur 4: Performance for different T and different ways of lowering T for the second sequence.



Figur 5: Performance for different T and different ways of lowering T for the third sequence.

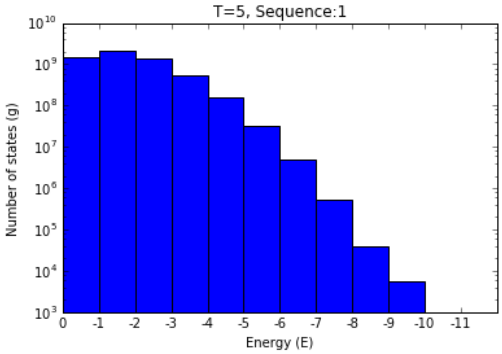


Figur 6: Performance for different T and different ways of lowering T for the fourth sequence.

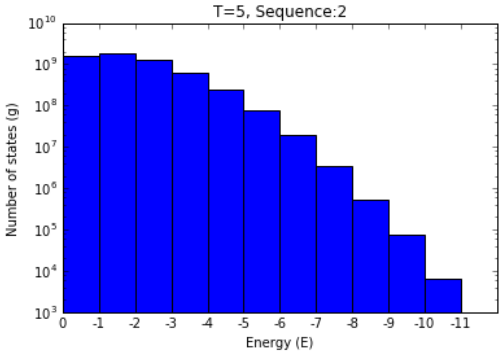


Figur 7: Performance for different T and different ways of lowering T for the fifth sequence.

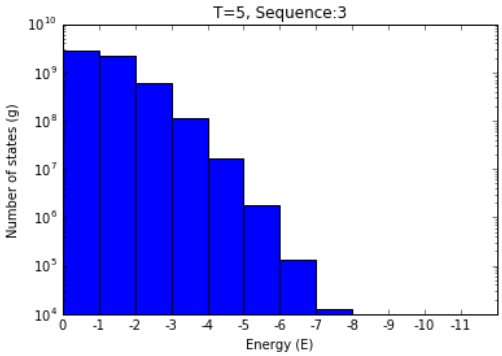
Appendix B



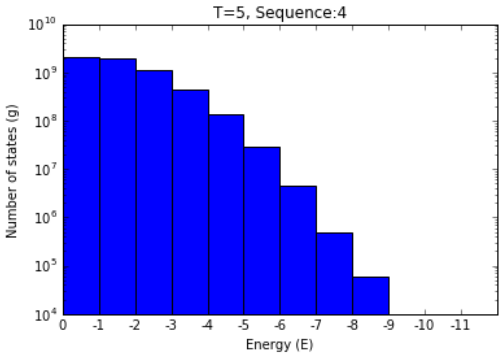
Figur 8: Histogram for $T = 5$ and the first sequence.



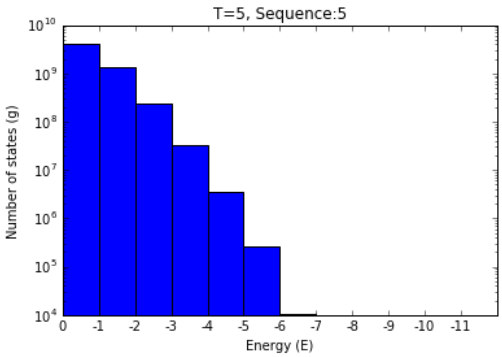
Figur 9: Histogram for $T = 5$ and the second sequence.



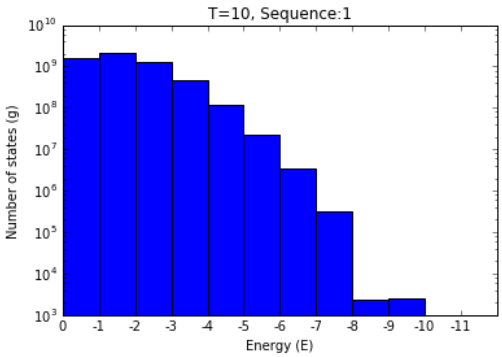
Figur 10: Histogram for $T = 5$ and the third sequence.



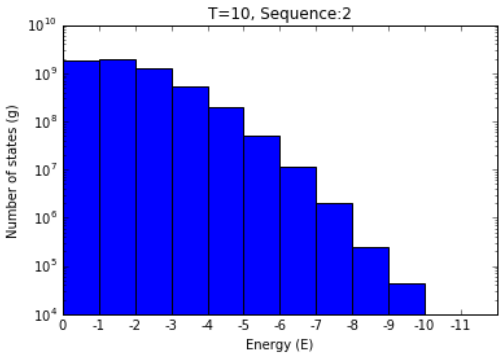
Figur 11: Histogram for $T = 5$ and the fourth sequence.



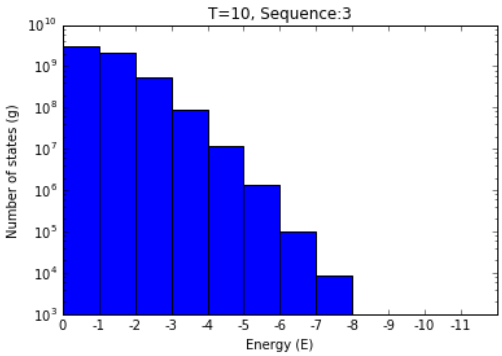
Figur 12: Histogram for $T = 5$ and the fifth sequence.



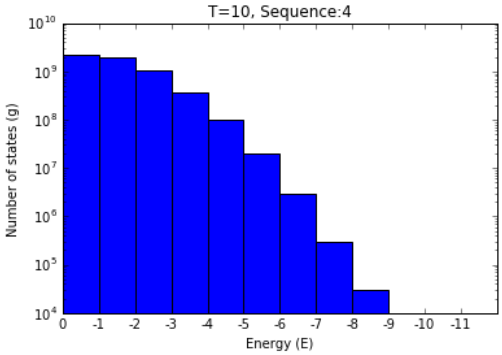
Figur 13: Histogram for $T = 10$ and the first sequence.



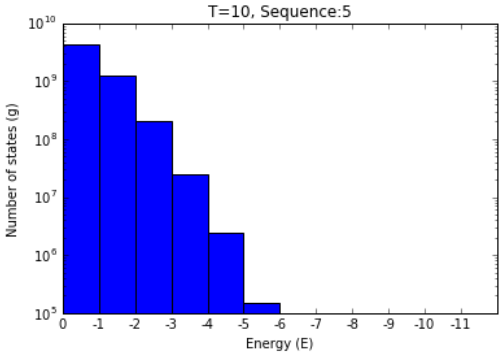
Figur 14: Histogram for $T = 10$ and the second sequence.



Figur 15: Histogram for $T = 10$ and the third sequence.



Figur 16: Histogram for $T = 10$ and the fourth sequence.



Figur 17: Histogram for $T = 10$ and the fifth sequence.

Appendix C

```
from __future__ import division, print_function

import numpy as np
import matplotlib.pyplot as plt
import time

from scipy.spatial.distance import pdist, squareform



class Switcher(object): #given to us. Not included to free up some space



def boltzmaN(hist,T):
    G=5768299665 #number of SAW. Given to us.
    u=[]
    g=[]
    for i in range(len(hist)): #Calculating the partition function
        u.append(hist[i]*np.exp(i/T))
    C=np.sum(u)
    for i in range(len(hist)):#calculating the number of degeneracies
        g.append(G*hist[i]/C*np.exp(i/T))
    return g

def accept(E1,E2,T):
    if E2<=E1:#accept lower energies
        return True
    kb=1
    p=np.exp(-(E2-E1)/(kb*T))
    r=np.random.rand()
    if r<p:#accept higher energies with a probability
        return True
    else:
        return False

start = time.time()
M=1
states=[1,1,1,14,10]
Eground=[-10,-11,-9,-10,-7]
temp=[1,10]
S = [[1,1,0,1,1,0,1,0,0,1,1,0,1,0,0,1,0,0,0,1,1,0,1,0,1],
     [0,1,1,1,1,0,1,1,1,0,0,0,0,0,1,1,1,0,1,1,1,0,1,1,1],
     [0,1,1,1,0,0,0,1,0,0,1,1,0,1,1,0,0,0,0,0,1,0,0,1,1],
     [1,0,0,1,0,1,0,1,1,1,0,0,0,0,0,1,0,1,0,1,0,1,1,0,1],
     [1,0,0,0,1,0,1,0,1,0,0,1,0,0,0,1,1,1,0,0,0,1,1,0,0]]

for T in temp: #loop for different temperatures
    Etot=[]
    l=0
```

```
for seq in S: #loop for different sequences
    Esave=[0] #[0,1,1,1,2] # x positions for sequence
    E=0
    pos=[]
    for i in range(len(seq)):
        pos.append((0,i))
    xpos=[0]*len(seq)
    ypos=[0]*len(seq)

    N=5000000
    for k in range(N): #loop for the pivoting and energy check
        for i in range(len(seq)):
            xpos[i]=pos[i][0]
            ypos[i]=pos[i][1]
        s=Switcher(seq,xpos,ypos)
        s.pivot()
        Enew=s.hh_contact()
        if accept(E,Enew,T)==True:
            pos=s.new_stat
            E=Enew
        Esave.append(Enew)
    Etot.append(Esave)
    print(l)
    l+=1

histtot=[]
for i in range(len(Etot)): #make histogram
    hist=[0]*12
    for j in range(N):
        E=np.abs(Etot[i][j])
        hist[E]+=1
    histtot.append(hist)



newhist=[boltzmaN(i,T) for i in histtot] #find the number of degenerate states
via boltzmann distribution
print(newhisto)
mybin=[0,-1,-2,-3,-4,-5,-6,-7,-8,-9,-10,-11]
it=1
for i in newhist: #plot
    plt.figure()
    plt.title("T={}, Sequence:{}".format(T,it))
    plt.xlabel("Energy (E)")
    plt.ylabel("Number of states (g)")
    pos = np.arange(len(mybin))
    width = 1.0     # gives histogram aspect to the bar diagram
    ax = plt.axes()
    ax.set_xticks(pos)
    ax.set_xticklabels(mybin)
    plt.bar(pos,i,1)
    plt.savefig("Histogram_T={},Sequence:{}.png".format(T,it))
```

```
            it+=1
end = time.time()
print(end-start)
```

Appendix D

```
from __future__ import division, print_function

import numpy as np
import matplotlib.pyplot as plt
import time
#import Trash
from scipy.spatial.distance import pdist, squareform



class Switcher(object): #given to us. Not included to free up some space

poslist=[[] for _ in range(16)]
deglist=[[] for _ in range(16)]
M=10
start = time.time()
k=0
for a in range(2): #looping all the 16 different sequences
    for b in range(2):
        for c in range(2):
            for d in range(2):
                for n in range(M):
                    deg=[]
                    seq = [0,1,0,0,1,a,1,b,1,1,1,1,1,0,c,1,d,1,1,0,0,1]
                    E=0
                    pos=[]
                    for i in range(len(seq)):#making sequance in to a stright line
                        pos.append((0,i))
                    xpos=[0]*len(seq)
                    ypos=[0]*len(seq)
                    T=1
                    N=30000
                    Tred=T/N
                    for n in range(N):#pivoting and checking the energy
                        for i in range(len(seq)):
                            xpos[i]=pos[i][0]
                            ypos[i]=pos[i][1]
                        s=Switcher(seq,xpos,ypos)
                        s.pivot()
                        Enew=s.hh_contact()
                        if Enew==-11:
                            poslist[k].append(s.new_stat)
                        if accept(E,Enew,T)==True:
                            pos=s.new_stat
                            E=Enew
                        T=T-Tred
                k+=1
for j in range(16):#removing equal configurations
    for i in poslist[j]:
        if i not in deglist[j]:
```

```
                deglist[j].append(i)

degstates=[len(x) for x in deglist]#printing the number of ground states
print(degstates)
end = time.time()
print(end-start)
```

Appendix E

```
start = time.time()
sumE=[]
M=20
states=[1,1,1,14,10]
Eground=[-10,-11,-9,-10,-7]
temp=[0.1,1,10]
S = [[1,1,0,1,1,0,1,0,0,1,1,0,1,0,0,1,0,0,0,1,1,0,1,0,1],
     [0,1,1,1,1,0,1,1,1,0,0,0,0,0,1,1,1,0,1,1,1,0,1,1,1],
     [0,1,1,1,0,0,0,1,0,0,1,1,0,1,1,0,0,0,0,0,1,0,0,1,1],
     [1,0,0,1,0,1,0,1,1,1,0,0,0,0,0,1,0,1,0,1,0,1,1,0,1],
     [1,0,0,0,1,0,1,0,1,0,0,1,0,0,0,1,1,1,0,0,0,1,1,0,0]]
N=30000

scoreTinf=[0,0,0,0,0]
a=0
for seq in S: #T=0
    for n in range(M):
        E=0
        pos=[]
        for i in range(len(seq)):
            pos.append((0,i))
        xpos=[0]*len(seq)
        ypos=[0]*len(seq)

        for k in range(N):
            for i in range(len(seq)):
                xpos[i]=pos[i][0]
                ypos[i]=pos[i][1]
            s=Switcher(seq,xpos,ypos)
            s.pivot()
            Enew=s.hh_contact()
            if Enew==Eground[a]:
                scoreTinf[a]+=1
                break
            if Enew>=E:
                pos=s.new_stat
                E=Enew
    a+=1

scorelinear=[[0]*len(S) for _ in range(len(temp))]
a=0
for T in temp:#linear annealing
    b=0
    for seq in S:
        for n in range(M):
            E=0
            Enew=0
            pos=[]
            for i in range(len(seq)):
                pos.append((0,i))
```

```
            xpos=[0]*len(seq)
            ypos=[0]*len(seq)
            Tnow=T
            Tred=Tnow/N
            for k in range(N):
                for i in range(len(seq)):
                    xpos[i]=pos[i][0]
                    ypos[i]=pos[i][1]
                s=Switcher(seq,xpos,ypos)
                s.pivot()
                Enew=s.hh_contact()
                if Enew==Eground[b]:
                    scorelinear[a][b]+=1
                    break
                if accept(E,Enew,Tnow)==True:
                    pos=s.new_stat
                    E=Enew
                Tnow-=Tred
        b+=1
    a+=1

scoreexpo=[[0]*len(S) for _ in range(len(temp))]
a=0
for T in temp: #exponential annealing
    b=0
    for seq in S:
        for n in range(M):
            E=0
            Enew=0
            pos=[]
            for i in range(len(seq)):
                pos.append((0,i))
            xpos=[0]*len(seq)
            ypos=[0]*len(seq)
            Tnow=T

            for k in range(N):
                Tred=Tnow/N
                for i in range(len(seq)):
                    xpos[i]=pos[i][0]
                    ypos[i]=pos[i][1]
                s=Switcher(seq,xpos,ypos)
                s.pivot()
                Enew=s.hh_contact()
                if Enew==Eground[b]:
                    scoreexpo[a][b]+=1
                    break
                if accept(E,Enew,Tnow)==True:
                    pos=s.new_stat
                    E=Enew
                Tnow-=Tred
        b+=1
```

```
    a+=1

finalscoreTinf=np.multiply(scoreTinf,1/M)
finalscorelinear=np.multiply(scorelinear,1/M)
finalscoreexpo=np.multiply(scoreexpo,1/M)


print(finalscoreTinf)
print(scorelinear)
print(finalscorelinear)
print(finalscoreexpo)
for i in range(len(S)): #Plotting
    plt.figure()
    plt.title("Optimization, Sequence: {}".format(i))
    plt.xlabel("Temperature (K)")
    plt.ylabel("Performance number")
    plt.axhline(finalscoreTinf[i], color="red",linestyle="--", label="0 temperatu
    plt.semilogx(temp,[x[i] for x in finalscorelinear],"bx",label="Linear Annealin
    plt.semilogx(temp,[x[i] for x in finalscoreexpo],"go",label="Exponential Anne
    plt.legend()
    plt.savefig("Optimization_Sequence{}".format(i))

end = time.time()
print(end-start)
```