Peer Assessments (https://class.coursera.org/smac-001/human_grading/)
/ Homework session 9: Simulated Annealing for sphere packings and the travelling salesman problem
Help (https://class.coursera.org/smac-001/help/peergrading?url=https%3A%2F%2Fclass.coursera.org%2Fsmac-001%2Fhuman_grading%2Fview%2Fcourses%2F971628%2Fassessments%2F13%2Fresults%2Fmine)

Submission Phase

1. Do assignment ✓ (/smac-001/human_grading/view/courses/971628/assessments/13/submissions)

Evaluation Phase

2. Evaluate peers ✓ (/smac-001/human_grading/view/courses/971628/assessments/13/peerGradingSets)

Results Phase

3. See results ✓ (/smac-001/human_grading/view/courses/971628/assessments/13/results/mine)

Your effective grade is 15

Your unadjusted grade is 15, which is simply the grade you received from your peers.

See below for details.

In this homework session 9 of **Statistical Mechanics: Algorithms and Computations**, you will study a crucial application of statistical mechanics, namely the **simulated annealing** method. After a short introduction, using two preparation programs (one of them purely imaginary), you will consider the packing problem of **disks on a sphere**, following Newton and Gregory (1694) and then the **travelling salesman problem**.

A Here, we resume a mathematical result relevant to simulated annealing.

A1 Download (cut-and-paste) the below **Preparation program 1**, which simulates a particle in a one-dimensional potential

```
V = -4.0 *x ** 2   -0.5 * x ** 3 +  x ** 4
```

with two minima. The global minimum is at x = 1.614, and is this minimum that we want to "find" using Monte Carlo methods. The second (local) minimum is located at x = -1.239.

```
import math, random

def V(x):
    pot =  -4.0 * x ** 2   -0.5 * x ** 3 +  x ** 4
    return pot

gamma = 0.0125
n_iter = 100
n_plus = 0
for iteration in range(n_iter):
    T = 2.0
    x = 0
    delta = 0.1
    step = 0
    n_accept = 0
    while T > 0.0001:
        step += 1
        if step == 100:
            T *= (1.0 - gamma)
            if n_accept < 20:
               delta *= 0.5
            step = 0
            n_accept = 0
        x_new = x + random.uniform(-delta, delta)
        if random.uniform(0.0, 1.0) < math.exp(- (V(x_new) - V(x)) / T):
            x = x_new
            n_accept += 1
    if x > 1.58 and x < 1.62: n_plus += 1
print n_plus / float(n_iter), x, gamma
```

Study **Preparation program 1**, in particular its **annealing schedule** where, after each 100 iterations, the temperature is rescaled to

```
T -> T * (1 - gamma)   (Simulated annealing step)
```

compute (by simply running the program, once you have understood it) the **approximate probability** with which the correct solution x \sim 1.614 is found for gamma = 0.5, 0.25, 0.125 ... 0.002 (approximately). Convince yourself that

1. for finite annealing rate gamma, the correct solution is not necessarily found in the limit t -> infinity.
2. the correct solution is found with probability 1 for t -> infinity in the limit of vanishing annealing rate (gamma -> 0).

Supply a table of your results (probability vs gamma for very large times), **or upload a graph** (**not required**).

NB: Both your observations here are backed by mathematical theorems: For smooth potentials, simulated annealing is guaranteed to find the ground-state in the limit of infinite time in the limit of infinitely slow annealing. However, these theorems are not really useful in practice.

Prob = Probability of having the correct x (x $\sim$ 1.614).

Gamma	Prob	x
0.5	0.49	1.61390202908
0.25	0.51	1.61373882203
0.125	0.61	1.61433189791
0.0625	0.62	1.61353508976
0.03125	0.65	1.61315379712
0.015625	0.84	1.6193326257
0.007813	0.86	1.61438599875
0.0039065	0.97	1.6175102714
0.001953	0.98	1.61664941718

One can observe that the probability increases with lowering of gamma, and in the limit of vanishing gamma ($\sim$ 0.002), the correct solution is found with probability close to 1.

Evaluation/feedback on the above work

Note: this section can only be filled out during the evaluation phase.

In this section, your fellow student should simply show that he/she has run the program and understood the general lessons of it.

There are two issues:
1/ for finite gamme the probability to hit the global (right) minimum is smaller than 1.
2/ for gamma -> 0 the probability to hit the global (right) minimum seems to go to 1.

The following answer would have obtained full score:

For finite gamma, the probability is smaller than 1.

and here is the table of this probability as a function of gamma:

gamma	probability to hit global minimum
0.500000	0.488000
0.250000	0.526000
0.125000	0.608000
0.062500	0.598000
0.031250	0.684000
0.015625	0.800000
0.007812	0.896000
0.003906	0.950000
0.001953	0.982000

It seems to go to one for small gamma.

POINTS - DESCRIPTION

give 0 points if really nothing has been done (question not treated)
give 1 point if the program has been run, but not really run for small gamma
give 2 points if the program has been run and the general lessons understood.

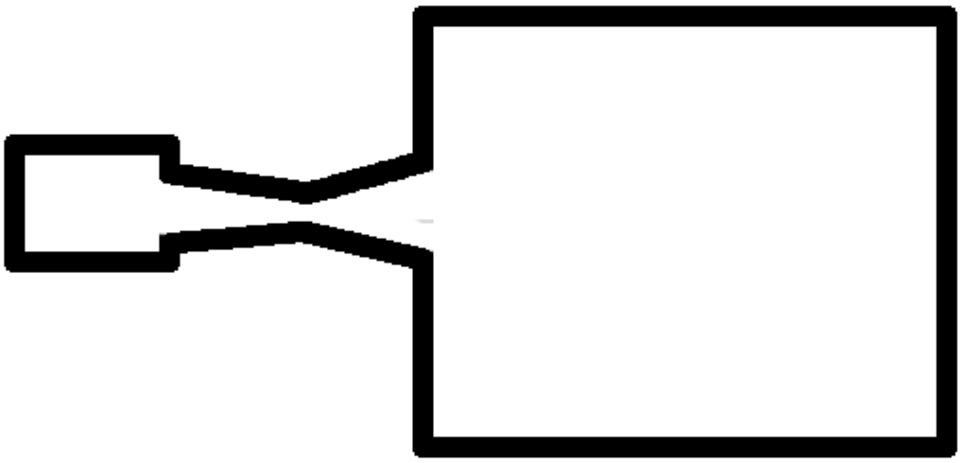
NB: There is a **final-week homework bonus**: If some work was done (program run for different gammas), your fellow student should get full points.

Score from your peers: **2**

peer 1 → *[This area was left blank by the evaluator.]*

peer 2 → *[This area was left blank by the evaluator.]*

A2 We now imagine a "Gedankenexperiment" (thought experiment) of a hard disk in the container shown below. An actual implementation is neither required nor expected. The container has two boxes connected through a **hard bottleneck**. All its contours, and in particular the bottleneck, are made of **hard walls**.



We imagine performing Monte Carlo simulation with displacements (x,y) -> (x + delx, y + dely), where delx, dely (that can be positive or negative) are much smaller than the dimensions of the container.

After each 100 iterations, the disk radius is increased

```
sigma -> sigma * (1 + gamma)  with gamma > 0 (simulated annealing step)
```

if this is possible (if there is an overlap, no action is taken). We want to find the largest disk fitting into the container.

Convince yourself that **simulated annealing for this case** does **NOT FIND WITH PROBABILITY 1** the optimal solution (disk to the right), even for gamma -> 0 in the limit t -> infinity. **Write down a short explanation** of why this is so.

In the above system, there is a big barrier due to the hard bottleneck. So, it the disk gets trapped in the left container, it is unlikely to escape the barrier and move to the right box also for small gamma. Therefore, we do not find with probability 1 the optimal solution even for gamma -> 0 in the limit t -> infinity.

Evaluation/feedback on the above work

Note: this section can only be filled out during the evaluation phase.

The following answer would have obtained full score:

Just before the size of the disk becomes larger than the bottle-neck, the disk has a finite probability to be in the left box. It is for this reason that it
will have a finite probability to be in the left box, even in the limit t -> \infty, independent of the annealing schedule.

POINTS - DESCRIPTION

NB:There is a **final-week homework bonus**: if some work was done, your fellow student should get full points.

Score from your peers: **1**

peer 1 → *[This area was left blank by the evaluator.]*

peer 2 → *[This area was left blank by the evaluator.]*

B Here we perform **simulated annealing for the disk-packing problem**. For simplicity, please download (cut-and-paste) the below program.
Note also that we only output configurations that are above a given density min_density.

```
import random, math

def unit_sphere():
    x = [random.gauss(0.0, 1.0) for i in range(3)]
    norm =  math.sqrt(sum(xk ** 2 for xk in x))
    return [xk / norm for xk in x]

def minimum_distance(positions, N):
    dists = [math.sqrt(sum((positions[k][j] - positions[l][j]) ** 2 \
             for j in range(3))) for l in range(N) for k in range(l)]
    return min(dists)

def resize_disks(positions, r, N, gamma):
    Upsilon = minimum_distance(positions, N) / 2.0
    r = r + gamma * (Upsilon - r)
    return r

N = 13
gamma  = 0.5
min_density = 0.78
for iteration in range(100):
    print iteration
    sigma  = 0.25
    r = 0.0
    positions = [unit_sphere() for j in range(N)]
    n_acc = 0
    step = 0
    while sigma > 1.e-8:
        step += 1
        if step % 500000 == 0:
            eta = N / 2.0 * (1.0 - math.sqrt(1.0 - r ** 2))
            print r, eta, sigma, acc_rate
        k = random.randint(0, N - 1)
        newpos = [positions[k][j] + random.gauss(0, sigma) for j in range(3)]
        norm = math.sqrt(sum(xk ** 2 for xk in newpos))
        newpos = [xk / norm for xk in newpos]
        new_min_dist = min([math.sqrt(sum((positions[l][j] - newpos[j]) ** 2 \
                       for j in range(3))) for l in range(k) + range(k + 1, N)])
        if new_min_dist > 2.0 * r:
            positions = positions[:k] + [newpos] + positions[k + 1:]
            n_acc += 1
        if step % 100 == 0:
            acc_rate = n_acc / float(100)
            n_acc = 0
            if acc_rate < 0.2:
                sigma *= 0.5
            elif acc_rate > 0.8 and sigma < 0.5:
                sigma *= 2.0
            r = resize_disks(positions, r, N, gamma)
            R = 1.0 / (1.0 / r - 1.0)
            eta = 1.0 * N / 2.0 * (1.0 - math.sqrt(1.0 - r ** 2))
    print 'final density: %f (gamma = %f)' % (eta, gamma)
    if eta > min_density:
        f = open('N_' + str(N) + '_final_'+ str(eta) + '.txt', 'w')
        for a in positions:
           f.write(str(a[0]) + ' ' + str(a[1]) + ' ' + str(a[2]) + '\n')
        f.close()
```

Note the role of the parameter in this program.

R = 1 / (1/r -1)   relation between outer-sphere radius R and disk radius r

eta = N * (1 - sqrt(1 - r^2)) / 2   surface area of the spherical caps formed by the disks.

B1 Run the simulated annealing program for N = 13 disks on a sphere. Experiment with its various parameters, especially with the annealing rate gamma, and the control parameter sigma, that sets the step width of the Markov chain.
NB: Remember that step-width adjustment during a Monte Carlo calculation used for integration is FORBIDDEN (see the discussion in thread...).

Use somewhat smaller values of gamma to recover the conjectured optimal solution for N=13, that has density eta = 0.79139, and a sub-optimal solution with density eta = 0.78639. **Communicate the parameter gamma you found useful**. **Upload (cut-and-paste) the positions x,y,z** as output by the program, **for both configurations**.

Answer the following question: **For small annealing rates**, do you **ALWAYS** recover the optimal solution (case discussed in section A1) or do you find that you continue to find different solutions even for small annealing rates (case discussed in section A2)?

Optional (no points to be gained): Illustrate the optimal solution using, for example, a modified version of the pure Python program given in tutorial 9 (example_pylab_visualization.py). Print it out on glossy paper.

NB: While it is proven that R < 1 for N = 13, it has not been proven that the solution with eta = 0.79139 is actually the best one.

gamma = 0.025

Positions file for optimal solution (eta = 0.79139) attached: N_13_final_0.791392804901 (https://s3.amazonaws.com/coursera-uploads/user-4ee8c8fa0a6b395f7b4d5436/971628/asst-13/4d79ffd0c27511e3b581516b3b1ac593.txt)

```
0.00261685763212 0.704015518314 0.710179767404
-0.677903838797 -0.531854846617 -0.507520253265
0.724925190564 -0.549581386277 0.415263492187
0.0184826931047 -0.998710534662 -0.0472827453843
-0.0239302607526 0.978329546916 -0.205666332321
0.682417710626 0.450305127199 -0.575787600285
0.841002167035 0.398481107989 0.365961967447
-0.0223380521148 -0.0384995830221 -0.999008905633
0.026770286253 -0.214280524207 0.976405248204
0.67404848457 -0.505355312143 -0.538771425503
-0.682889339428 -0.577175862371 0.447805955735
-0.725396737686 0.422710652842 -0.543245135209
-0.838162133059 0.365567940736 0.404776876084
```

Positions file for sub-optimal solution (eta = 0.78639) attached: N_13_final_0.786386055227 (https://s3.amazonaws.com/coursera-uploads/user-4ee8c8fa0a6b395f7b4d5436/971628/asst-13/5ba9ead0c28811e3b581516b3b1ac593.txt)

```
-0.961281847395 0.180896829509 -0.207878683233
-0.103539661326 0.796283912718 -0.595996198712
-0.239786951884 -0.780128277687 0.577842615302
0.773868078643 0.548595017979 -0.316499136026
-0.454277224575 0.0965614860649 0.885611699698
-0.47129657149 0.835356574314 0.28294687744
0.949140325538 -0.308493625122 0.0629628914297
0.432236205011 0.707162710643 0.559546927213
-0.37799905417 -0.055962597623 -0.92411303568
0.549003053146 -0.241934646902 -0.800039545438
0.453716880059 -0.194310624721 0.869703612658
-0.639924119689 -0.713569272029 -0.285159630831
0.283505839016 -0.937081552545 -0.203721876891
```

Even for small annealing rates, one does not necessarily always recover the optimal solution. We continue to find different solutions even for small annealing rates as the case discussed in section A2.

Attached is the run log for 100 iterations: B_N_13_runlog (https://s3.amazonaws.com/coursera-uploads/user-4ee8c8fa0a6b395f7b4d5436/971628/asst-13/8e60ad90c2d511e3834b1fa346ab3378.txt)

We see that sub-optimal densities such as 0.775224, 0.786386, 0.784624, 0.773760 etc. are obtained as well.

Evaluation/feedback on the above work

Note: this section can only be filled out during the evaluation phase.

B1 Here you evaluate whether your fellow student can run

The following answer would have obtained full score:

I used gamma = 0.02, and found configurations with 0.791392810853 and

```
0.786386058319
```

Density: 0.791392810853

```
0.437453564251 0.125353021734 -0.890461115977
0.280268350008 -0.502601275947 -0.817827371393
0.918090759752 0.391113507264 0.0643395779766
-0.950186539848 -0.2554665524 -0.178556377919
-0.312364104514 0.638248242215 0.703610579455
0.459618394076 0.435814950204 -0.773832191761
0.830454295085 -0.555835164055 -0.037322033369
0.178101910295 -0.752715235677 0.633797667658
0.477024487279 0.124536368199 0.870022029337
0.175060060059 0.983605210514 -0.043298559128
-0.637165099069 -0.261066331556 0.725165503217
-0.72201192536 0.673143353494 -0.159927500092
-0.283735299075 -0.941437867604 -0.182178542919
```

Density 0.786386058319

```
0.46579807742 -0.439627730643 -0.767958077971
-0.431144040932 -0.899828901724 0.0665038614818
0.989323204974 -0.066661516437 -0.12959875897
-0.834514164358 0.364893331649 -0.412842543839
0.458549501518 -0.845893600114 -0.272390109849
0.608429381494 0.793088414055 0.0287133283827
0.425362657736 -0.125495752062 -0.896279769725
-0.325717007502 0.931672907843 0.160916201218
0.658218055813 0.15599439193 0.736488113066
0.0390138565171 -0.563771534671 0.825008833707
-0.262864899754 0.33731161872 0.903948514216
-0.909141336749 -0.141495965506 0.39171535783
0.00538810832763 0.709016971397 -0.705170832181
```

One finds different solutions even for very small annealing rates.

POINTS - DESCRIPTION
give 0 points if question not treated
give 1 point if one solution was found (and uploaded)
give 2 points if two solutions were found (and uploaded)
give 3 points for the two solutions and an understanding that very small annealing rates do not help.

NB:There is a **final-week homework bonus**: If some work was done (for N=13, two solutions found), your fellow student should get full points.

Score from your peers: **3**

peer 1 → *[This area was left blank by the evaluator.]*

peer 2 → *[This area was left blank by the evaluator.]*

B2 Find the optimal solution for *N* = 15 (it should have density eta = 0.80731). **Upload** (cut-and-paste) the positions *x,y,z* as output by the program.

Optional (zero points to be gained): In fact, for N=15, prove that there are TWO optimal solutions at the same density. (Write a program checking that only in one of them, the 5-connected disks touch each other.) **NB: Purely optional, no points gained**.

Positions file for optimal solution for N=15 (eta = 0.80731) attached: N_15_final_0.807314332098 (https://s3.amazonaws.com/coursera-uploads/user-4ee8c8fa0a6b395f7b4d5436/971628/asst-13/b87ed7f0c2d511e3aaaaf1c9ef263eb4.txt)

```
-0.501791168226 -0.591227948815 -0.631391428538
-0.907879434139 0.175033489851 -0.380943841659
-0.836166971432 -0.509502866934 0.203055717654
0.427097661887 -0.414651253698 0.803524688493
-0.212905043055 0.199772417479 -0.956432132384
-0.3322927311 0.870157170901 -0.363879154099
-0.0878243750122 -0.994193523842 0.0621781014843
0.26279738172 0.472115356492 0.841453876529
0.376004971308 -0.466353307578 -0.800708969641
0.749144893456 -0.662401406402 -0.00251105658916
0.932680635264 0.158957964545 0.32378881715
0.462572980783 0.884293821713 0.0636449081189
-0.419263103365 -0.116193419847 0.900398544724
-0.710026430377 0.578055132494 0.402137702737
0.775545398497 0.304269090418 -0.553127160322
```

Evaluation/feedback on the above work

Note: this section can only be filled out during the evaluation phase.

B2 Here is first the solution for the **optional part**: a program that allows to compute the connectivity of the disks. Download and run this program to see how it works.

```
import os, math

def dist(a,b):
    return math.sqrt( sum((a[k] - b[k]) **2 for k in range(3)))

N, eta_opt = 15, 0.80731
#N, eta_opt = 19, 0.81096

d_ref = math.sqrt(1.0 - (1.0 - 2.0 * eta_opt / float(N)) ** 2) * 2.0
mypath = os.getcwd()   # current working directory, where the data files
 are
files = [f for f in sorted(os.listdir(mypath)) if f[0:4] == 'N_%i' % N i
f f[-1] == 't']
for filename in files:
    print  '====================', filename, '===================='
    f = open(filename, 'r')
    pos = []
    for line in f:
        a, b, c = line.split()
        pos.append([float(a), float(b), float(c)])
    f.close()
    mat_mat = [[0 for i in range(N)] for j in range(N)]
    for i in range(N):
        for j in range(i):
            if dist(pos[i], pos[j]) < d_ref * 1.0001:
                mat_mat[i][j] = 1
                mat_mat[j][i] = 1
    connectivity = [sum(mat_mat[i]) for i in range(N)]
    for j in range(N):
        print 'disk %2i [connectivity %i] is connected with ' % (j, conn
ectivity[j]) + \
      ' '.join(['%2i [%i],' % (k, connectivity[k]) for k in range(N) if
mat_mat[j][k]])
    print
```

The following answer would have obtained full score:

I did find the solution for N=15, it has density 0.807314339866
And here is the set of x,y,z coordinates:
0.592946153131 -0.199447363905 -0.780151016482
0.968807728461 0.233511739227 -0.0829689876895
-0.469657269505 -0.316066822284 0.824332343811
-0.495697441307 0.0136167870855 -0.868388524677
-0.238680778659 0.717548271632 0.654336277289
-0.0976171874744 -0.763266926819 -0.638666175035
-0.205096560032 -0.949354495413 0.238036642353
-0.84406126026 -0.494113144222 -0.208357360404
0.181088692651 0.6037630407 -0.776322791163
-0.471431312039 0.857382542143 -0.206513182286
0.394043008564 0.918104394267 0.0425961104892
0.58705123766 0.359864450783 0.725167857413
-0.922505886677 0.25395365341 0.290672377369

0.405481035362 -0.524020662413 0.748990971458
0.6187827393 -0.780413814131 -0.0897897558623

POINTS - DESCRIPTION

Give 0 points if question not treated.
Give 1 point if some work done but optimal density not reached.
Give 2 points if optimal solution found and coordinates uploaded.

NB:There is a **final-week homework bonus**: If the optimal solution was found (for N=15), your fellow student should get full points.

Score from your peers: **2**

peer 1 → *[This area was left blank by the evaluator.]*

peer 2 → *[This area was left blank by the evaluator.]*

B3 Find the optimal solution for **N = 19** (it should have density eta = 0.81096). **Upload** (cut-and-paste) the positions x,y,z as output by the program.

Optional (zero points to be gained): **Show that there are infinitely many (sic) optimal solutions**, as one of the 19 disks is not jammed. (Write a program checking that one of the 19 disks is one-connected. This "rattler" has only one shortest distance).

NB: Configurations of parts **B2** and **B3** were found by D. A. Kottwitz (1991) using a very complicated optimization program.

Positions file for optimal solution for N=19 (eta = 0.81096) attached: N_19_final_0.81096092614 (https://s3.amazonaws.com/coursera-uploads/user-4ee8c8fa0a6b395f7b4d5436/971628/asst-13/b1f85a40c2d611e3aaaaf1c9ef263eb4.txt)

```
0.999327622217 -0.0366492328931 0.00106639722235
0.681345125601 0.232210821137 0.694151967776
0.436196647096 -0.878702471352 -0.193944455715
-0.579840472807 0.795942813917 0.173954198192
-0.0636305203403 -0.626092087418 -0.777148541113
0.117870802165 0.973922004484 -0.193861814647
-0.604115512114 -0.776544029769 -0.178951998741
0.0467973120997 0.146615702045 -0.988085951471
-0.992785086181 0.11862809082 0.0174685066642
0.663901721321 -0.283754892133 -0.691894258986
0.692630761641 0.504750140916 -0.515257142865
-0.719031780671 -0.424362518097 0.550372375414
0.0286545843952 -0.291513441413 0.956137452602
-0.707439822381 -0.144426629163 -0.691859701455
-0.0749162563204 -0.898631410774 0.432260502601
0.110510604576 0.796178873488 0.594883692572
-0.495505951324 0.307140939457 0.812488951009
-0.48222348586 0.627900815725 -0.610901854063
0.652639446503 -0.555728149294 0.515002890236
```

Evaluation/feedback on the above work

Note: this section can only be filled out during the evaluation phase.

B3 Here, we first explain the optional parts of the question: Please find here a program that computes the connectivity of the disks (this program is the same as the one used in section **B3**, for N = 19).

```
import os, math

def dist(a,b):
    return math.sqrt( sum((a[k] - b[k]) **2 for k in range(3)))

#N, eta_opt = 15, 0.80731
N, eta_opt = 19, 0.81096

d_ref = math.sqrt(1.0 - (1.0 - 2.0 * eta_opt / float(N)) ** 2) * 2.0
mypath = os.getcwd()   # current working directory, where the data files
 are
files = [f for f in sorted(os.listdir(mypath)) if f[0:4] == 'N_%i' % N i
f f[-1] == 't']
for filename in files:
    print  '====================', filename, '===================='
    f = open(filename, 'r')
    pos = []
    for line in f:
        a, b, c = line.split()
        pos.append([float(a), float(b), float(c)])
    f.close()
    mat_mat = [[0 for i in range(N)] for j in range(N)]
    for i in range(N):
        for j in range(i):
            if dist(pos[i], pos[j]) < d_ref * 1.0001:
                mat_mat[i][j] = 1
                mat_mat[j][i] = 1
    connectivity = [sum(mat_mat[i]) for i in range(N)]
    for j in range(N):
        print 'disk %2i [connectivity %i] is connected with ' % (j, conn
ectivity[j]) + \
      ' '.join(['%2i [%i],' % (k, connectivity[k]) for k in range(N) if
mat_mat[j][k]])
    print
```

Here is output of the program, it shows that one disk is unconnected.

```
==================== N_19_final_0.810960929026.txt ====================
disk  0 [connectivity 5] is connected with  3 [4],  5 [4],  6 [3],  8 [4],  9 [3],
disk  1 [connectivity 3] is connected with  2 [5],  3 [4], 14 [4],
disk  2 [connectivity 5] is connected with  1 [3],  3 [4],  4 [4],  5 [4], 15 [3],
disk  3 [connectivity 4] is connected with  0 [5],  1 [3],  2 [5],  6 [3],
disk  4 [connectivity 4] is connected with  2 [5],  5 [4],  7 [3], 15 [3],
disk  5 [connectivity 4] is connected with  0 [5],  2 [5],  4 [4],  8 [4],
disk  6 [connectivity 3] is connected with  0 [5],  3 [4], 11 [4],
disk  7 [connectivity 3] is connected with  4 [4], 12 [4], 17 [3],
disk  8 [connectivity 4] is connected with  0 [5],  5 [4],  9 [3], 18 [3],
disk  9 [connectivity 3] is connected with  0 [5],  8 [4], 16 [4],
disk 10 [connectivity 5] is connected with 11 [4], 12 [4], 14 [4], 16 [4], 17 [3],
disk 11 [connectivity 4] is connected with  6 [3], 10 [5], 14 [4], 16 [4],
disk 12 [connectivity 4] is connected with  7 [3], 10 [5], 14 [4], 15 [3],
disk 13 [connectivity 0] is connected with
disk 14 [connectivity 4] is connected with  1 [3], 10 [5], 11 [4], 12 [4],
```

disk 15 [connectivity 3] is connected with 2 [5], 4 [4], 12 [4],
disk 16 [connectivity 4] is connected with 9 [3], 10 [5], 11 [4], 18 [3],
disk 17 [connectivity 3] is connected with 7 [3], 10 [5], 18 [3],
disk 18 [connectivity 3] is connected with 8 [4], 16 [4], 17 [3],

The following answer would have obtained full score:

I did find the solution at eta = 0.810960929026**,** and here are the x,y,z values:

```
-0.442099078635 -0.851237356028 -0.282742583232
-0.84915508001 0.473594222949 0.233760907944
-0.684135528059 -0.156930947676 0.712271898161
-0.959271868651 -0.279913890214 -0.0380223102993
0.0781912198823 -0.250870209398 0.964857642956
-0.268746716408 -0.819368098422 0.506370537955
-0.606614879259 -0.222157346066 -0.763324637262
0.614070020126 0.306125963176 0.72746471052
0.320731470116 -0.946621725421 -0.0322278301745
0.171091150401 -0.627236525546 -0.759804026888
0.421934957981 0.788464236641 -0.447543337312
-0.185102718143 0.459770117005 -0.86853233863
0.248389665708 0.909692128826 0.33281046366
0.724624534378 -0.470468229703 0.503566211156
-0.34328104244 0.913521333972 -0.218258787404
-0.218812073518 0.491240658019 0.843091864741
0.533842531909 0.0908408327363 -0.840690248684
0.938699307079 0.344525750584 -0.0120672313835
0.86319256631 -0.408507920515 -0.296681432422
```

POINTS - DESCRIPTION

Give 0 points if the question was not treated
Give 1 point if some simulations were made but optimal density was not found
Give 2 points if the optimal density was found, and coordinates uploaded

NB:There is a **final-week homework bonus**: If the optimal density was found (for N=19), your fellow student should get full points.

Score from your peers: **2**

peer 1 → *[This area was left blank by the evaluator.]*

peer 2 → *[This area was left blank by the evaluator.]*

C Here we consider the travelling salesman problem (TSP), one of the classic problems in combinatorial optimization: given N cities, with distances d(i,j) = d(j,i) between them, find the shortest closed tour visiting all of them. The TSP is a prominent example of the class of NP complete problems, for which it is very difficult to find a solution, but this solution is easy to check. We will apply simulated annealing to the TSP in two dimensions. Simulated annealing provides a great "first approach" to this problem.

C1 Before simulated annealing, let us try a direct-sampling approach. For this, download (cut-and-paste) the below program.

```
import random, math, pylab

def dist(x, y):
    return math.sqrt((x[0] -y[0]) ** 2 + (x[1] - y[1]) ** 2)

def tour_length(cities, N):
    return sum (dist(cities[k + 1], cities[k]) for k in range(N - 1)) + dist(cities[0], c
ities[N - 1])

N = 20
random.seed(12345)
cities = [(random.uniform(0.0, 1.0), random.uniform(0.0, 1.0)) for i in range(N)]
random.seed()
energy_min = float('inf')
for iter in xrange(1000000):
    random.shuffle(cities)
    energy =  tour_length(cities, N)
    if energy < energy_min:
        print energy
        energy_min = energy
        new_cities = cities[:]
cities = new_cities[:]
for i in range(1,N):
    pylab.plot([cities[i][0], cities[i - 1][0]], [cities[i][1], cities[i - 1][1]], 'bo-')
pylab.plot([cities[0][0], cities[N - 1][0]], [cities[0][1], cities[N - 1][1]], 'bo-')
pylab.title(str(energy_min))
pylab.axis('scaled')
pylab.axis([0.0, 1.0, 0.0, 1.0])
pylab.savefig('TSP_configuration.png')
pylab.show()
```

Study this program. Notice that any tour is given by a random permutation of the cities. Notice also that the random number generator is initialized by a seed in order to generate an 'instance' (positions of the N cities), but is then randomized. Subsequent runs of the program produce the same instance, but then the program uses different random numbers at each run.

Run the program several times for **N=10**, for 1 Million iterations each (You may change the seed). Do you get the impression that you have obtained the optimal tour length? **Explain briefly your observations and upload the "optimal" tour (graphics file).**

Run the program several times for **N=20**, for 1 Million iterations each (You may change the seed). Do you get the impression that you have obtained the optimal tour length? **Explain briefly your observations and upload the "optimal" tour (graphics file).**

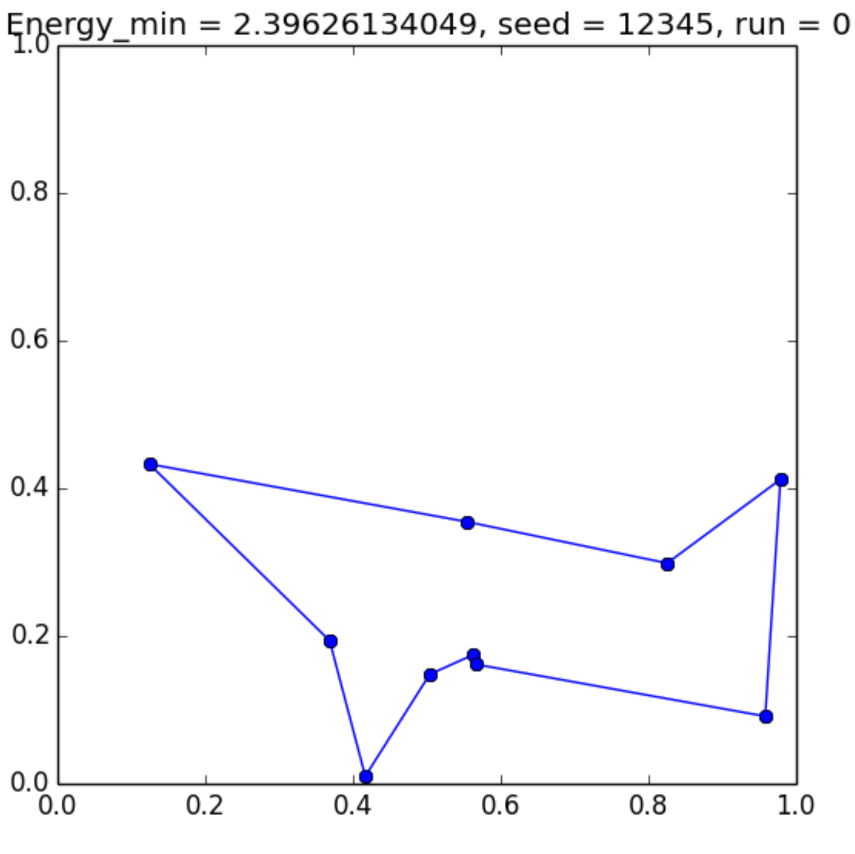
Simulations were done with four different seeds to check the convergence to optimal solution.

Flow:
1. Choose N as 10 or 20.
2. Choose a seed to generate the positions of N cities.
3. For the chosen seed and cities configuration, run 10 times the program (each time for 1 million iterations) to generate ten possible solutions.
4. Combine the 10 runs in a movie file (.gif) to get an overview of the solutions.
5. If the solution is same for all 10 runs, we have found an optimal solution. Otherwise, not.

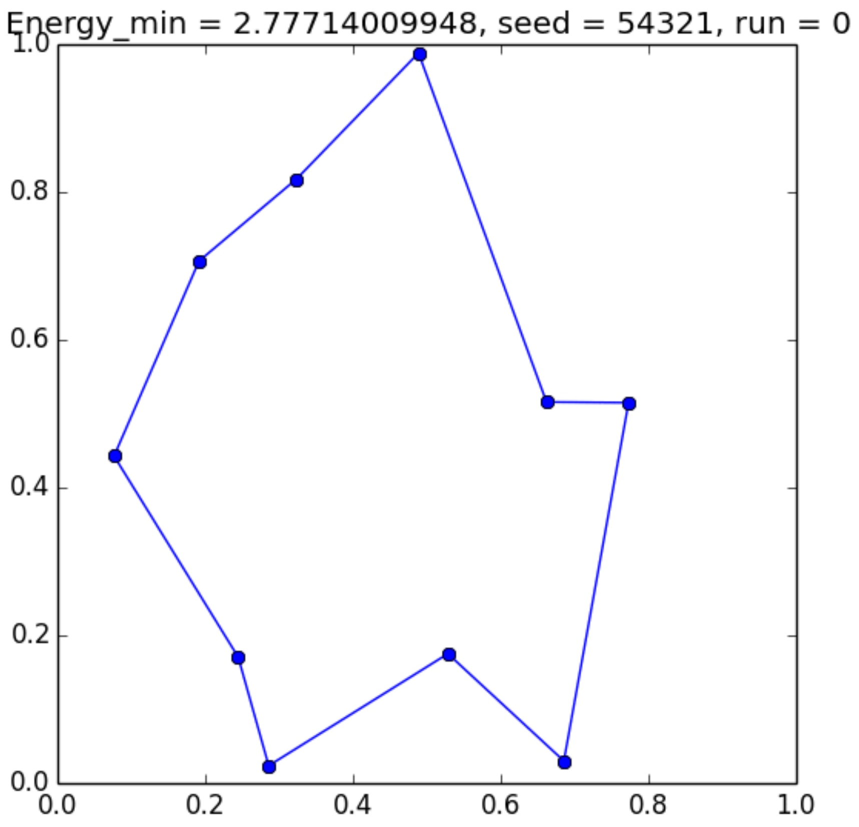
Results: For all 10 runs for N=10, the solution does not change. Therefore, we have the optimal solution for this case.
For the N=20 case, each of the 10 runs gives different solutions. Therefore, we can not be sure if we have converged to the optimal solution for this case.

"Optimal" tour plots for N=10:

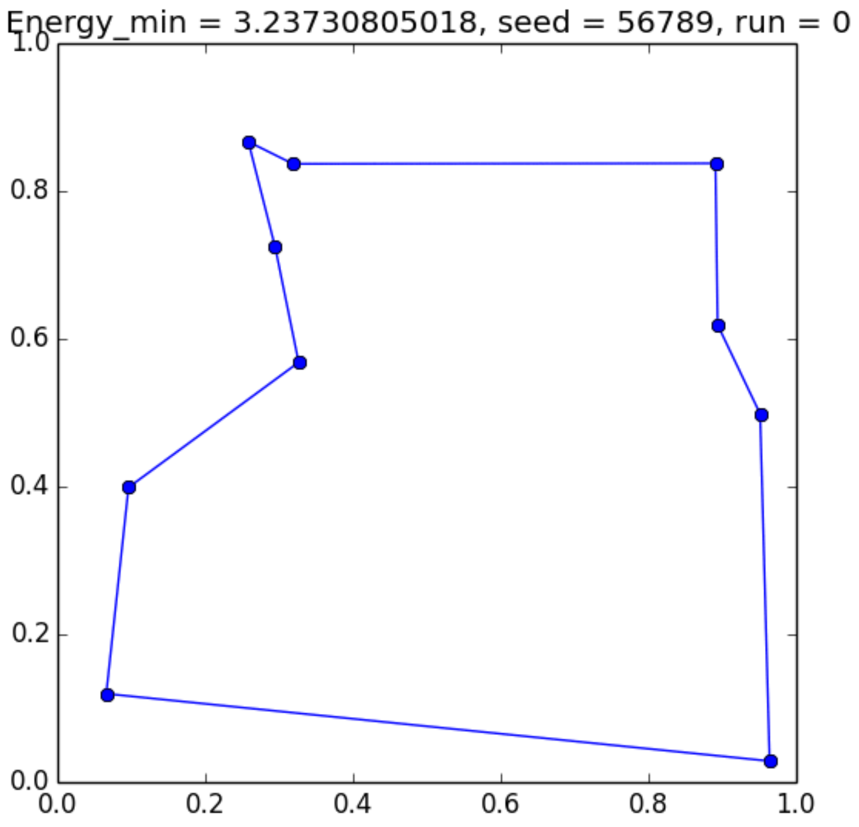
Seed = 12345



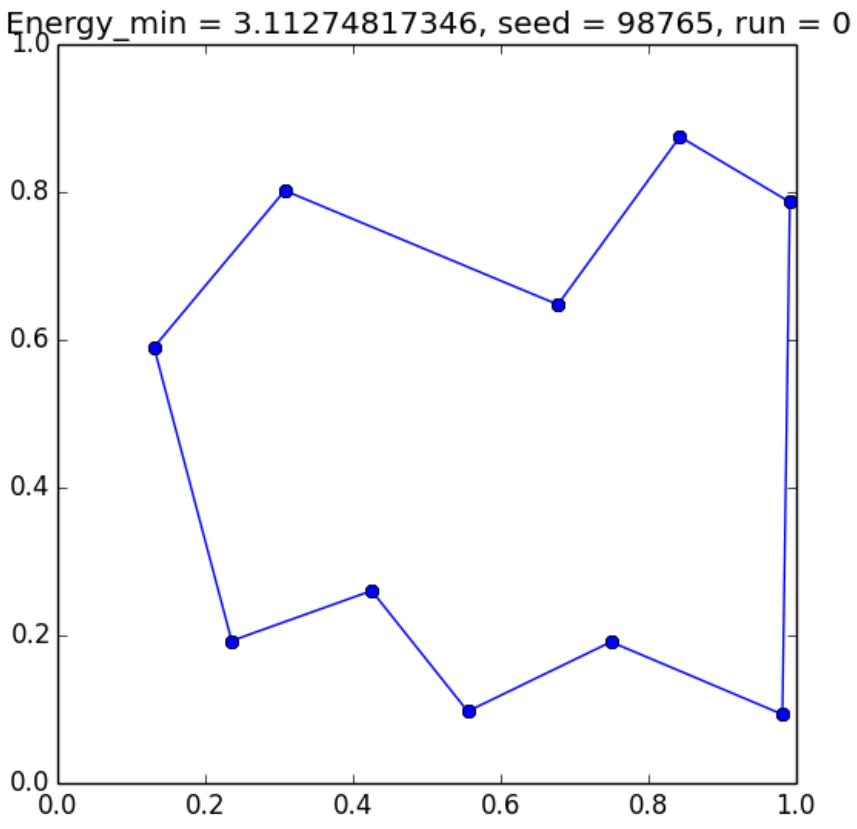
Seed = 54321



Seed = 56789

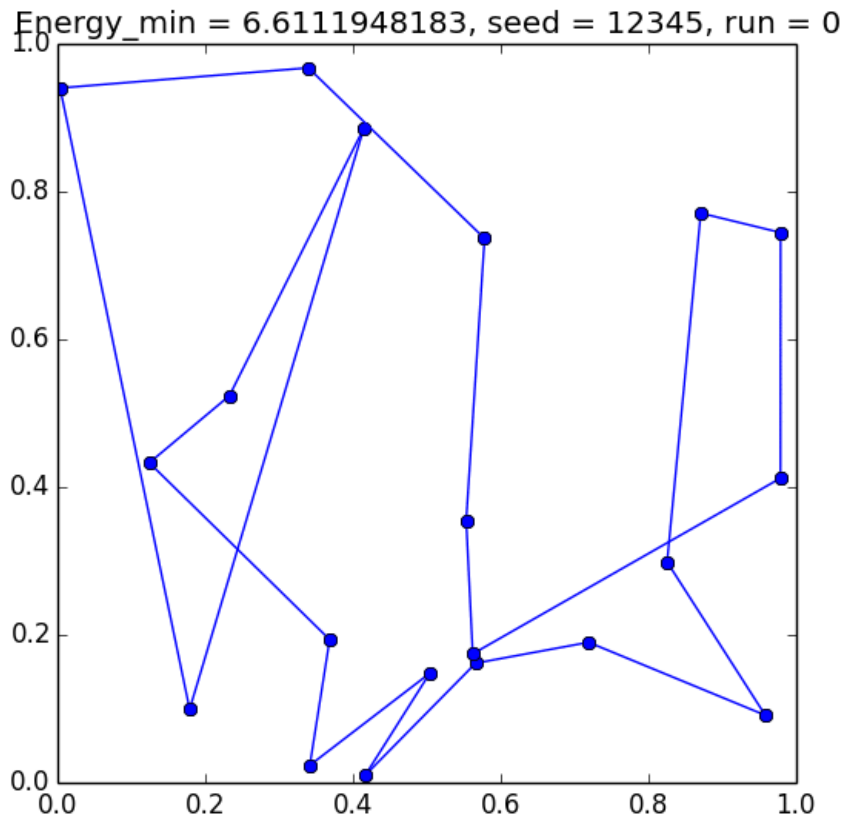


Seed = 98765

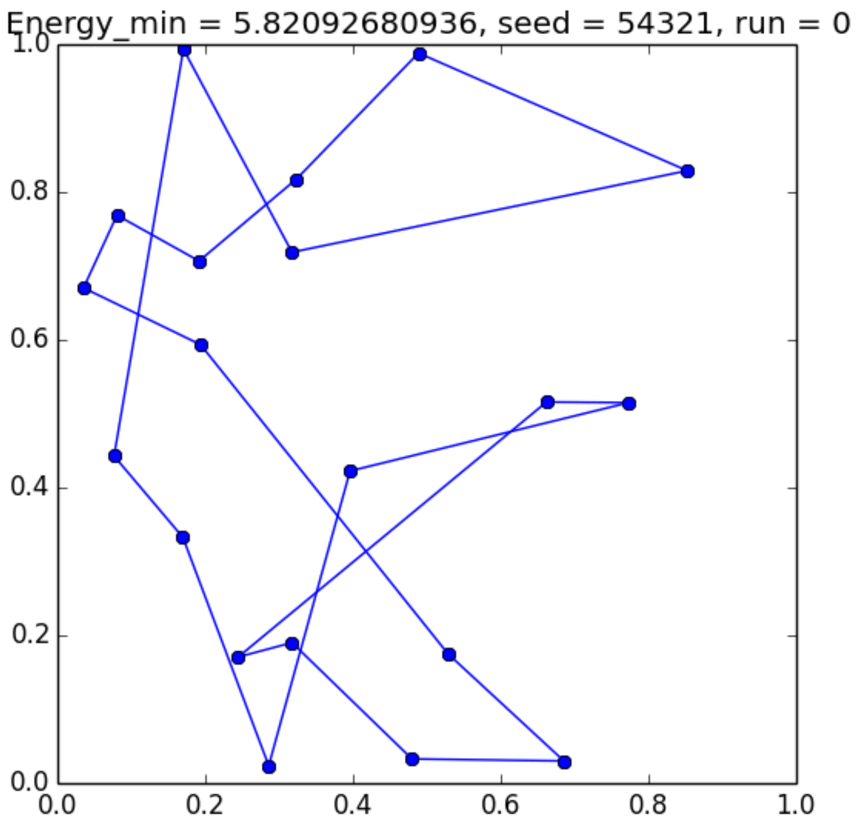


"Sub-optimal" tour plots for N=20:

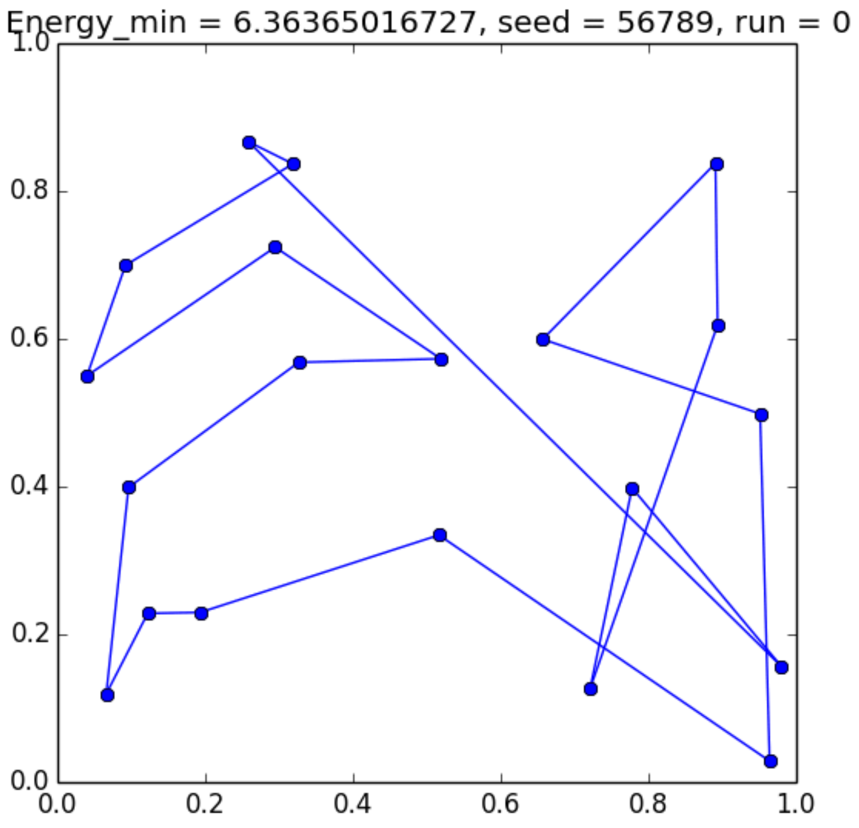
Seed = 12345



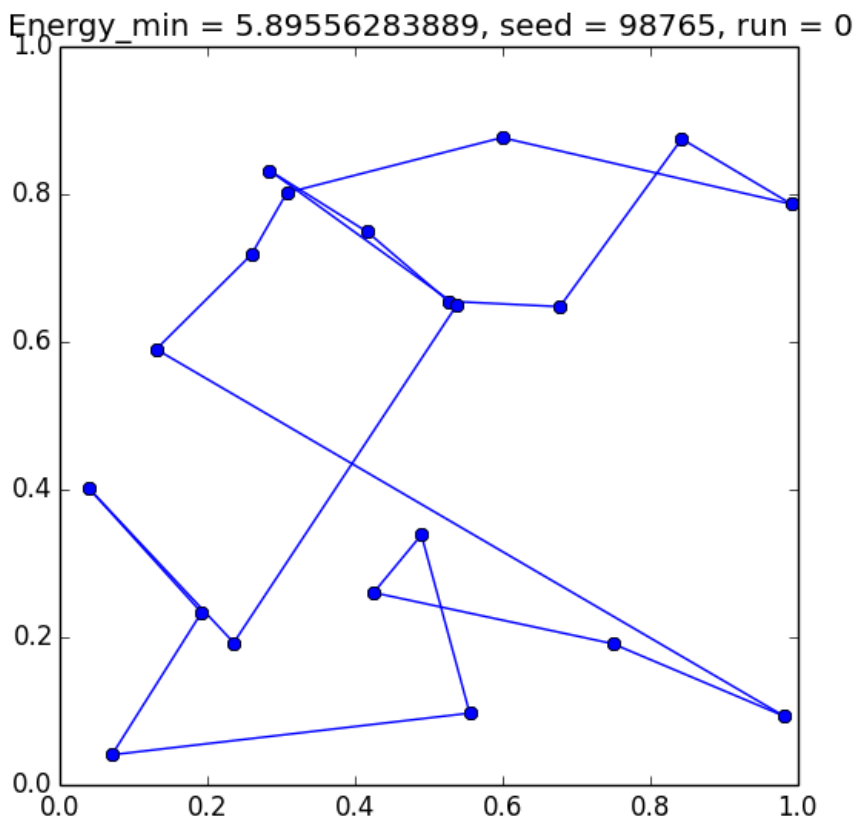
Seed = 54321



Seed = 56789



Seed = 98765



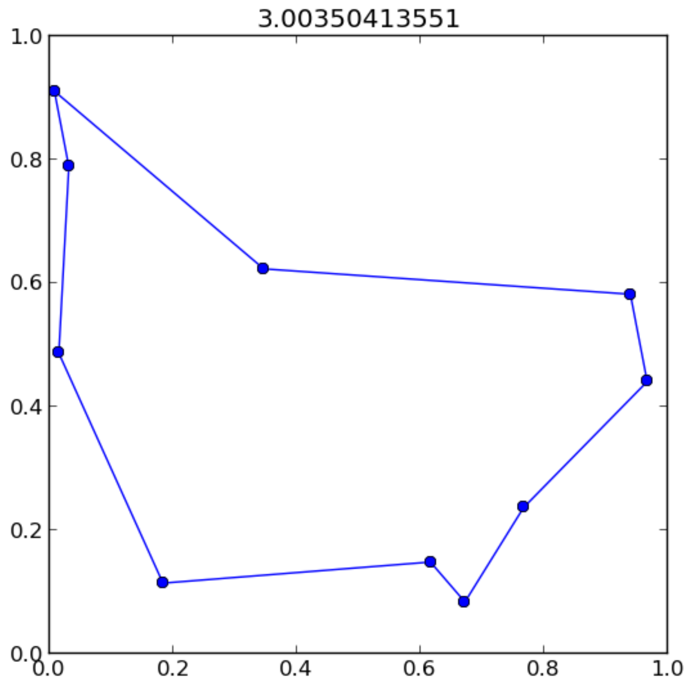
Evaluation/feedback on the above work

Note: this section can only be filled out during the evaluation phase.

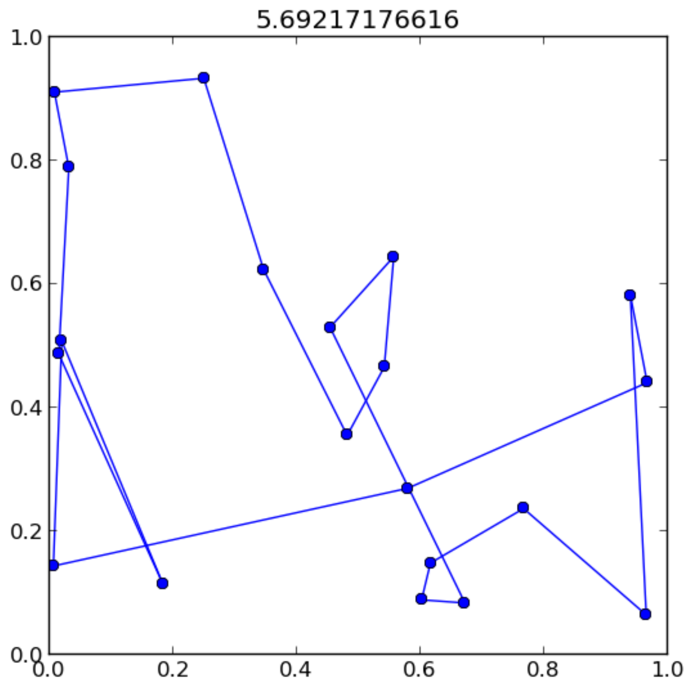
C1 Here you check whether your fellow student is able to download run and modify a simple program. All of the below is fairly elementary:

The following answer would have obtained full score:

Here is the best solution for N=10 that I found (for the random seed 1234). It looks like the optimal tour.



and here is the best direct-sampling solution for N=20, again for the seed 1234. There are crossings, and it is clearly not the optimal tour.



POINTS - DESCRIPTION

give 0 points if the question was not treated.
give 1 point if the direct sampling program was run for N=10.
give 2 point if the direct-sampling program was run for N=20.

NB:There is a **final-week homework bonus**: If some work was done (for N=10 and N=20), your fellow student should get full points.

Score from your peers: **2**

peer 1 → *[This area was left blank by the evaluator.]*

peer 2 → Nice. I wish you would say how you converted your pictures into .gif-s, perhaps on the forums?

C2 Now let us do **simulated annealing**, using as an energy the **length of the tour**. Download (cut-and-paste) the below program, that you are free to run and to modify.

```
import random, math, pylab

def dist(x, y):
    return math.sqrt((x[0] -y[0]) ** 2 + (x[1] - y[1]) ** 2)

def tour_length(cities, N):
    return sum (dist(cities[k + 1], cities[k]) for k in range(N - 1)) + dist(cities[0], c
ities[N - 1])

N = 20
random.seed(12345)
cities = [(random.uniform(0.0, 1.0), random.uniform(0.0, 1.0)) for i in range(N)]
random.seed()
random.shuffle(cities)
beta = 1.0
n_accept = 0
best_energy = 10000.
energy =  tour_length(cities, N)
for iter in xrange(1000000):
    if n_accept == 100:
        beta *=  1.005
        n_accept = 0
    p = random.uniform(0.0, 1.0)
    if p  < 0.2:
        i = random.randint(0, N / 2)
        cities = cities[i:] + cities[:i]
        i = random.randint(0, N / 2)
        a = cities[:i]
        a.reverse()
        new_cities =  a + cities[i:]
    elif p < 0.6:
        new_cities = cities[:]
        i = random.randint(1, N - 1)
        a = new_cities.pop(i)
        j = random.randint(1, N - 2)
        new_cities.insert(j, a)
    else:
        new_cities = cities[:]
        i = random.randint(1, N - 1)
        j = random.randint(1, N - 1)
        new_cities[i] = cities[j]
        new_cities[j] = cities[i]
    new_energy =  tour_length(new_cities, N)
    if random.uniform(0.0, 1.0) < math.exp(- beta * (new_energy - energy)):
        n_accept += 1
        energy = new_energy
        cities = new_cities[:]
        if energy < best_energy:
           best_energy = energy
           best_tour = cities[:]
    if iter % 100000 == 0: print energy, iter, 1.0 / beta
```

```
cities = best_tour[:]
for i in range(1,N):
    pylab.plot([cities[i][0], cities[i - 1][0]], [cities[i][1], cities[i - 1][1]], 'bo-')
pylab.plot([cities[0][0], cities[N - 1][0]], [cities[0][1], cities[N - 1][1]], 'bo-')
pylab.title(str(best_energy))
pylab.axis('scaled')
pylab.axis([0.0, 1.0, 0.0, 1.0])
pylab.savefig('simulated_annealing_best_path_N%i.png' % N)
```

Three types of moves are implemented in this program. **Make sure you understand these moves** (no need to write them up).

Run the program for the same instance as in **C1**, for *N*=10. **Do you find the same solution**? Or even do you find **a better solution**? **Upload the graphics file** of the solution you found.

Run the program for the same instance as in **C1**, for *N*=20. **Do you find the same solution**? Or even do you find **a better solution**? **Upload the graphics file** of the solution you found.

Run the program for *N*=50. Do you think that the program finds the optimal solution or can you see, by visual inspection, that the solution found is sub-optimal? **Upload the graphics file** of the solution you found.

NB: Don't hesitate to experiment with larger values of *N*, different moves, etc.

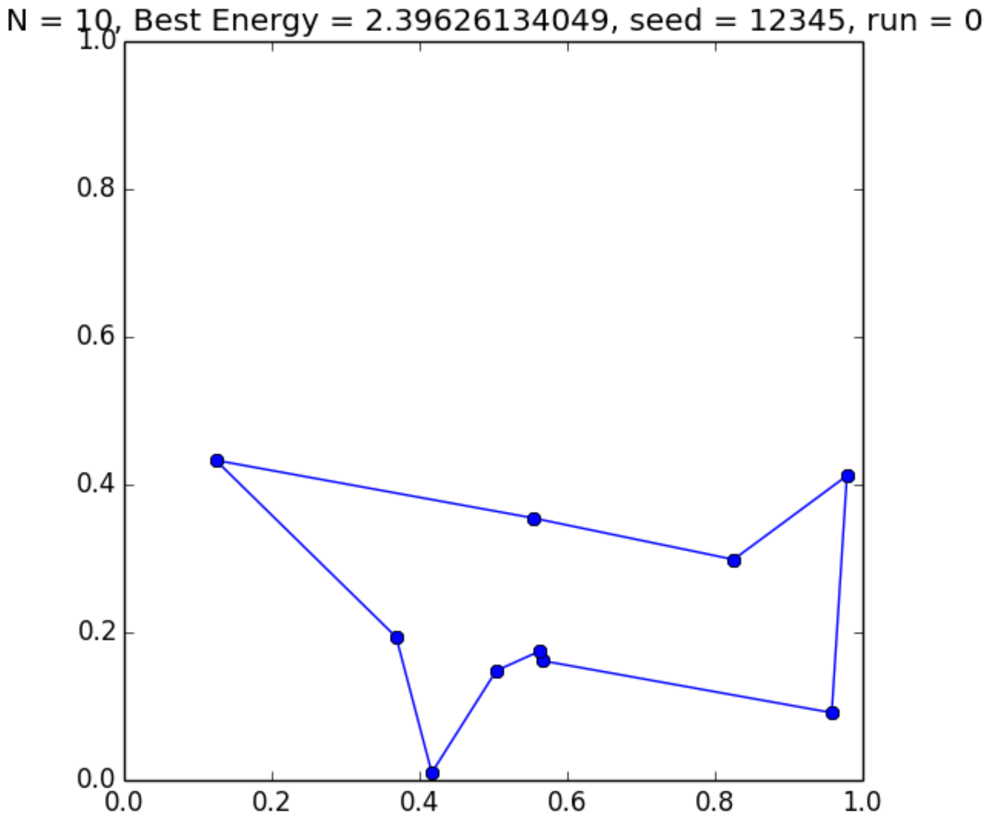
Simulations were done with four different seeds to check the convergence to optimal solution.

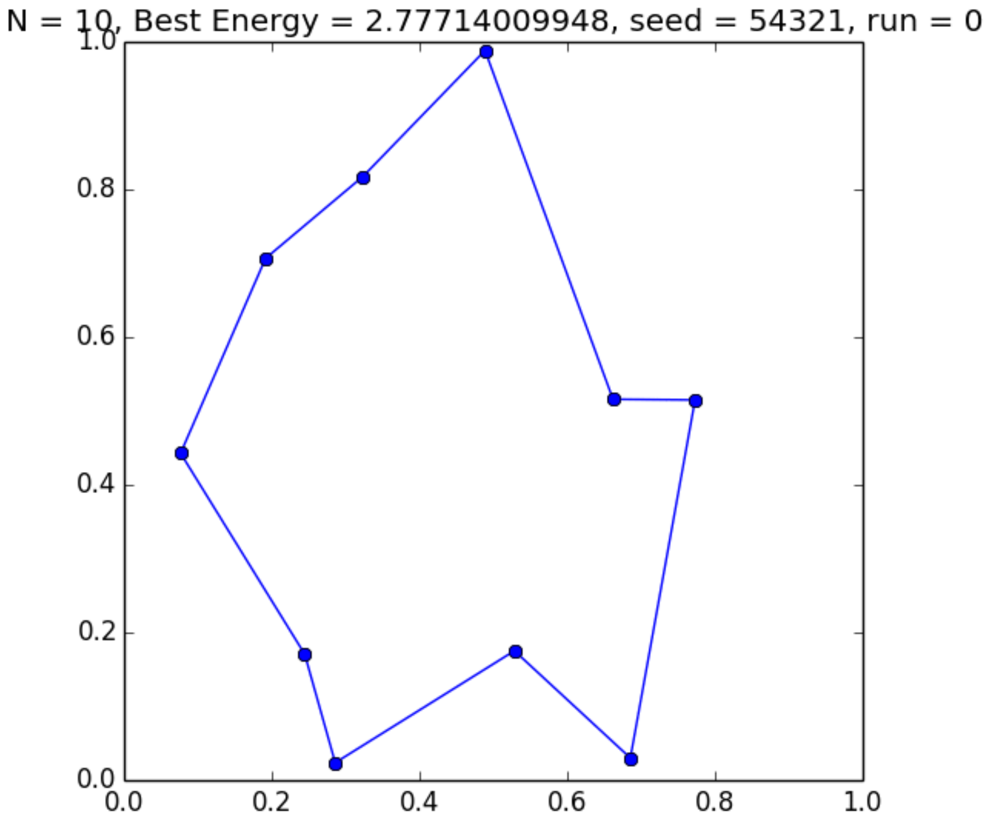
Flow:
1. Choose N as 10, 20 or 50.
2. Choose a seed to generate the positions of N cities.
3. For the chosen seed and cities configuration, run 10 times the program (each time for 1 million iterations) to generate ten possible solutions.
4. Combine the 10 runs in a movie file (.gif) to get an overview of the solutions.
5. If the solution is same for all 10 runs, we have found an optimal solution. Otherwise, not.

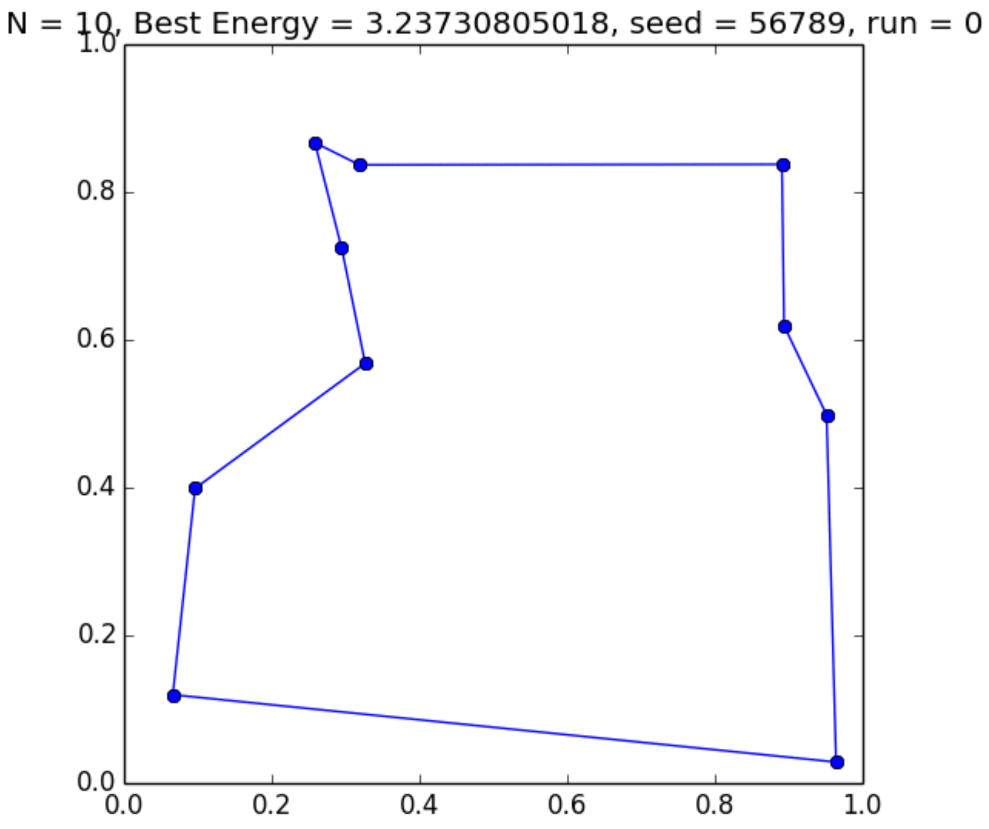
Results: For all 10 runs for N=10 and N=20, the solution does not change. Therefore, we have the optimal solution for these two cases.
For the N=50 case, we have some small variations between the runs (except the seed 56789). It's not as diverse as N=20 case in C1 above. Therefore, we can be reasonably sure regarding the optimal solution. Also, visually, the solution looks (almost) optimal.

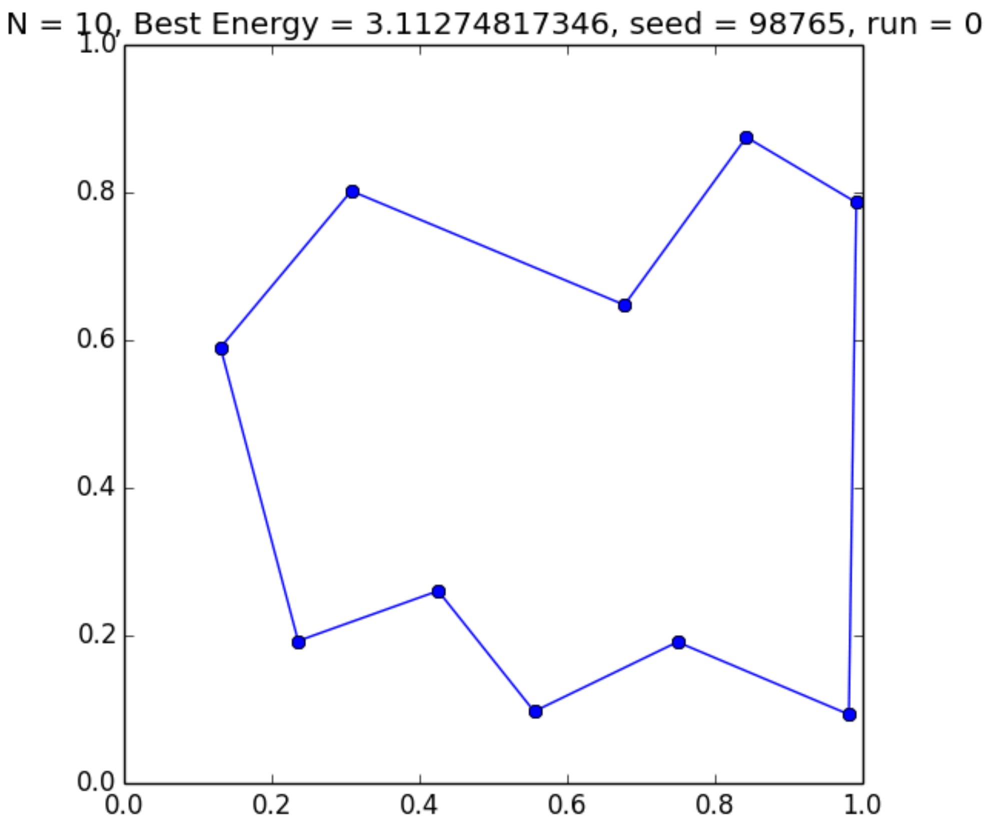
For N=10 case, the solutions in C1 match with those found here. For N=20 case, the solutions in C1 were clearly sub-optimal. The simulated annealing algorithm does provide optimal solutions for N=20 as confirmed by the fact that the plots do not change in between runs and also that they look visually optimal.

N=10 case for the four seeds:

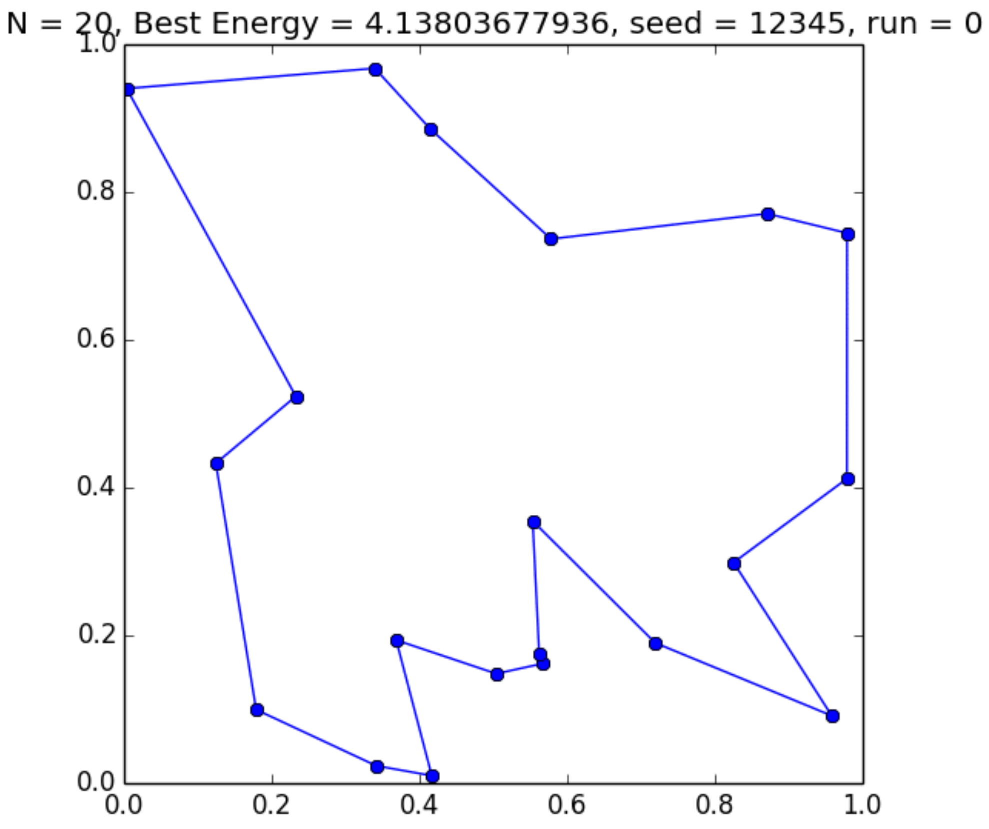

N = 10, Best Energy = 2.39626134049, seed = 12345, run = 0
0.0
0.2
0.4
0.6
0.8
1.0

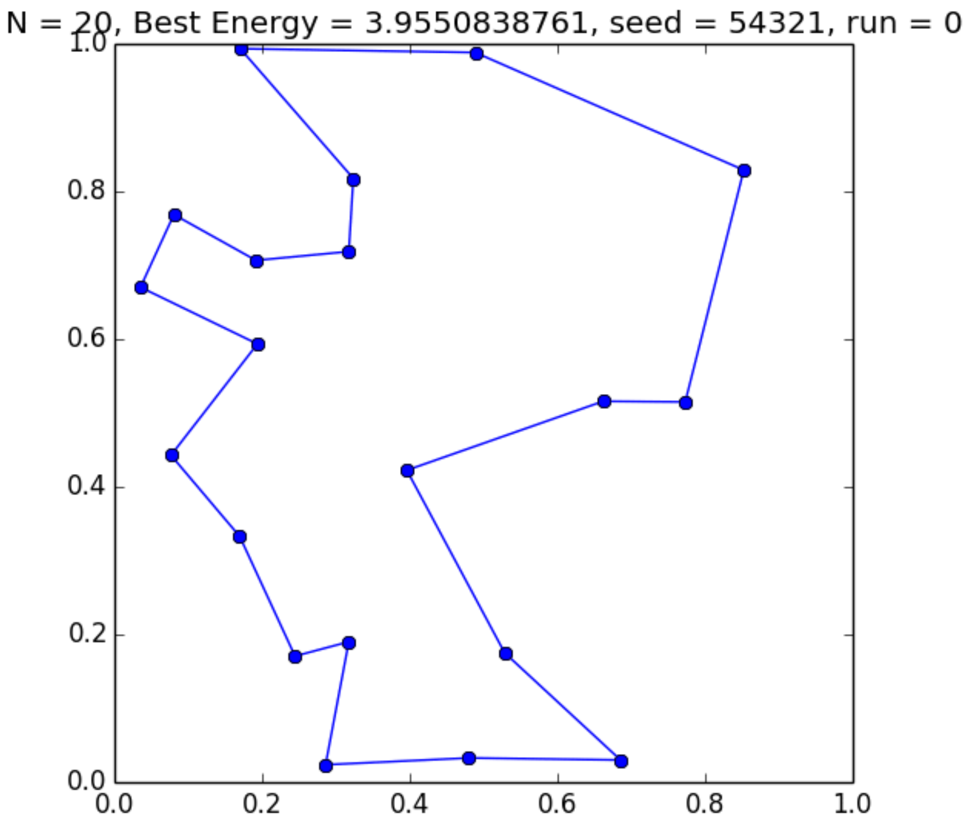

N = 10, Best Energy = 2.77714009948, seed = 54321, run = 0
0.0
0.2
0.4
0.6
0.8
1.0

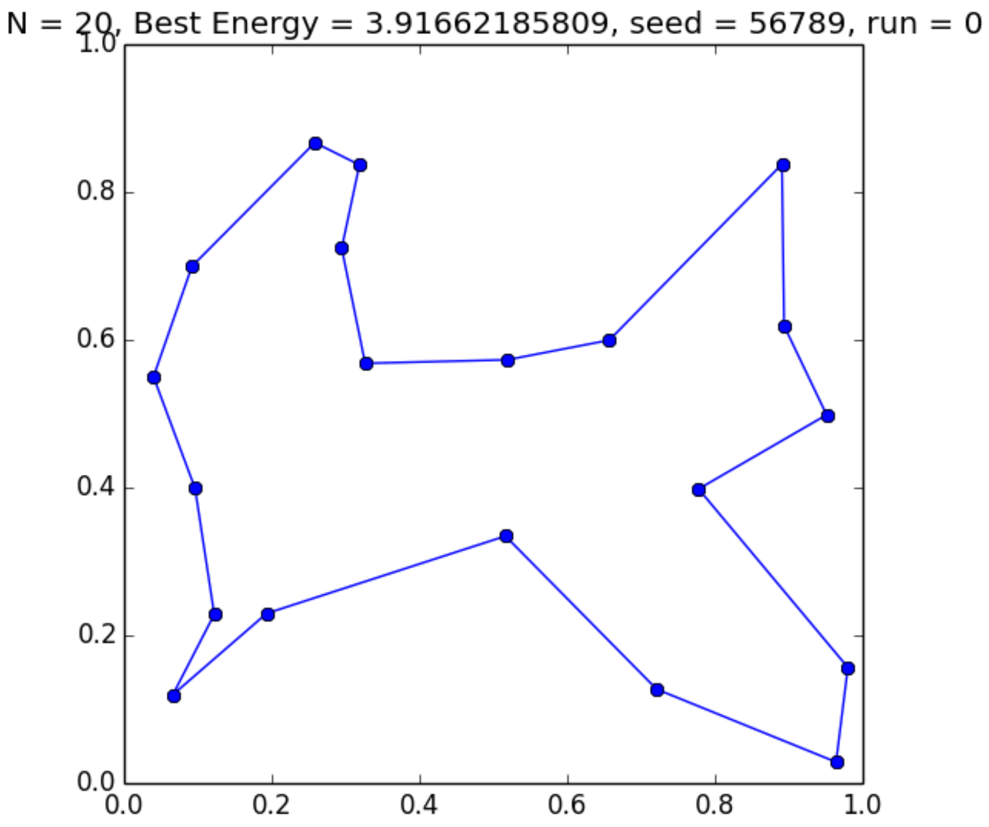


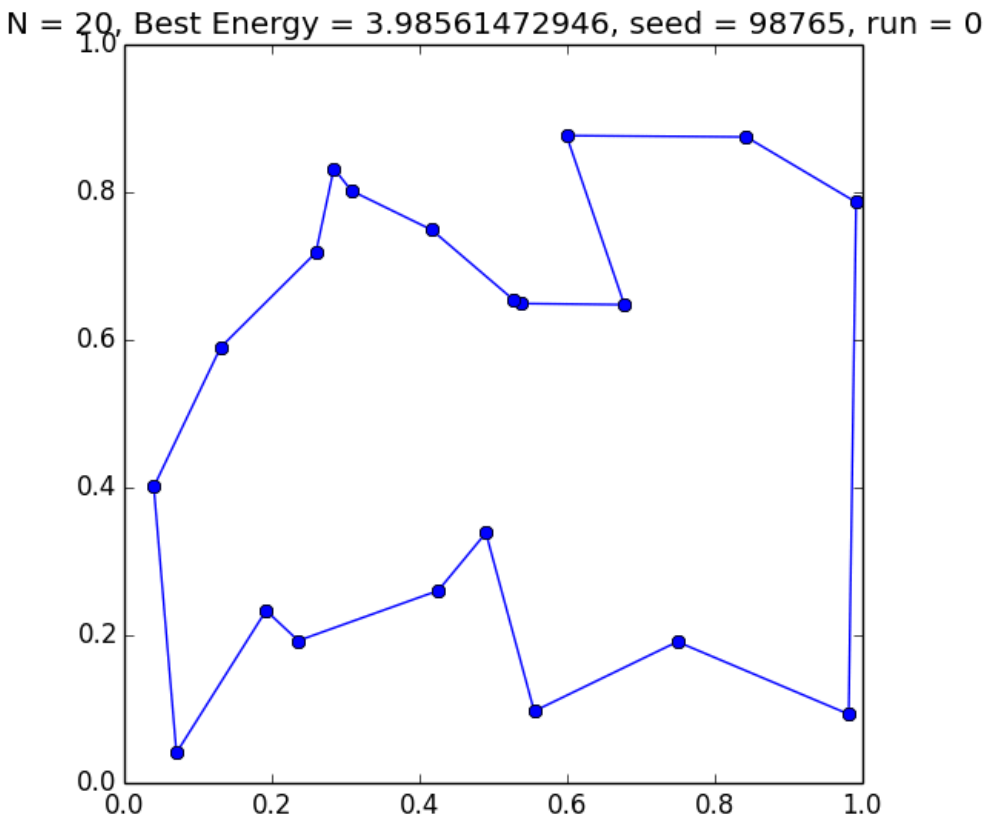


N=20 case for four seeds:

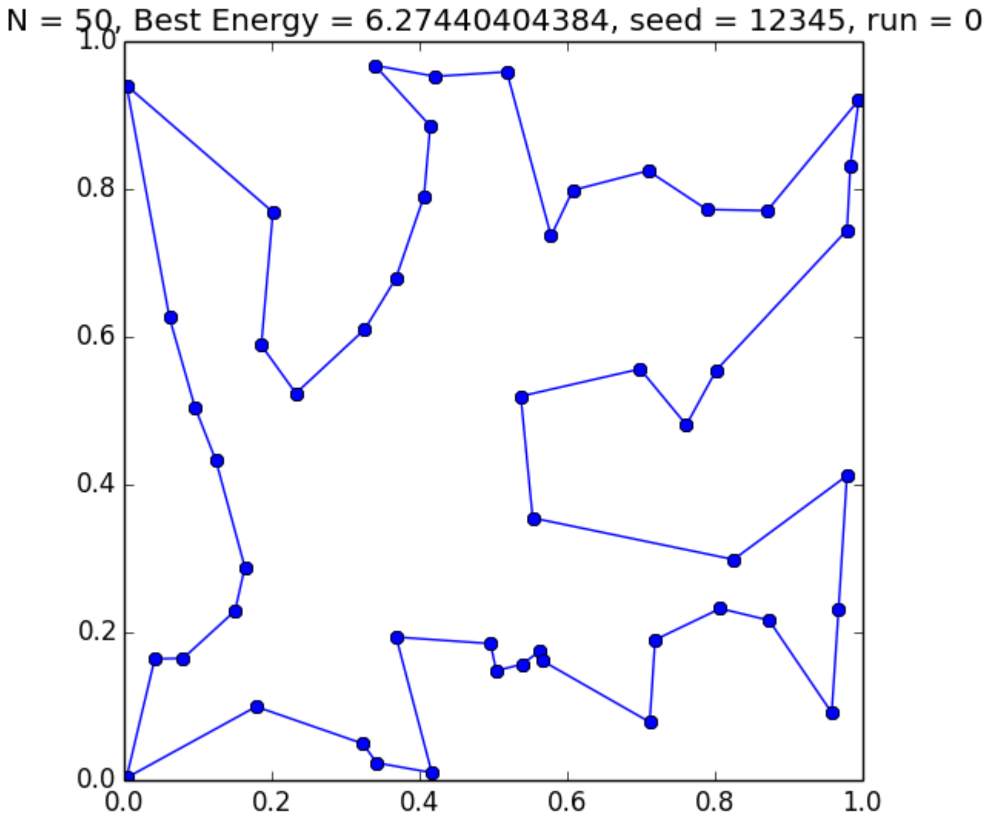

N = 20, Best Energy = 4.13803677936, seed = 12345, run = 0
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

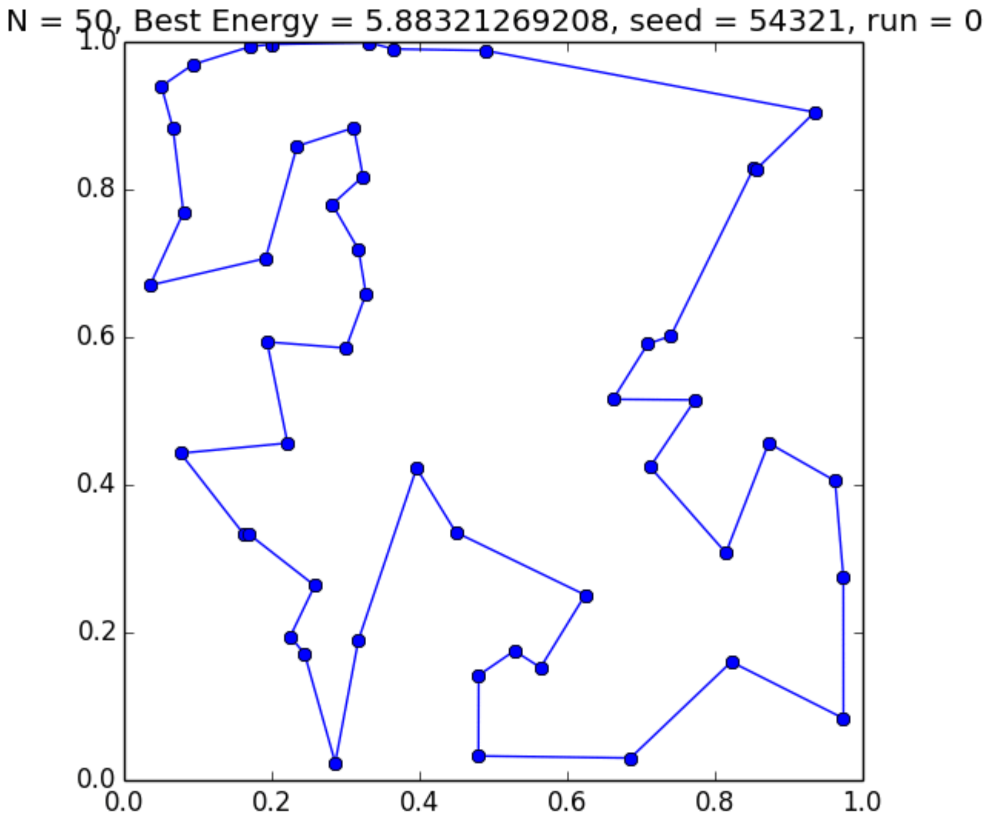

N = 20, Best Energy = 3.9550838761, seed = 54321, run = 0
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

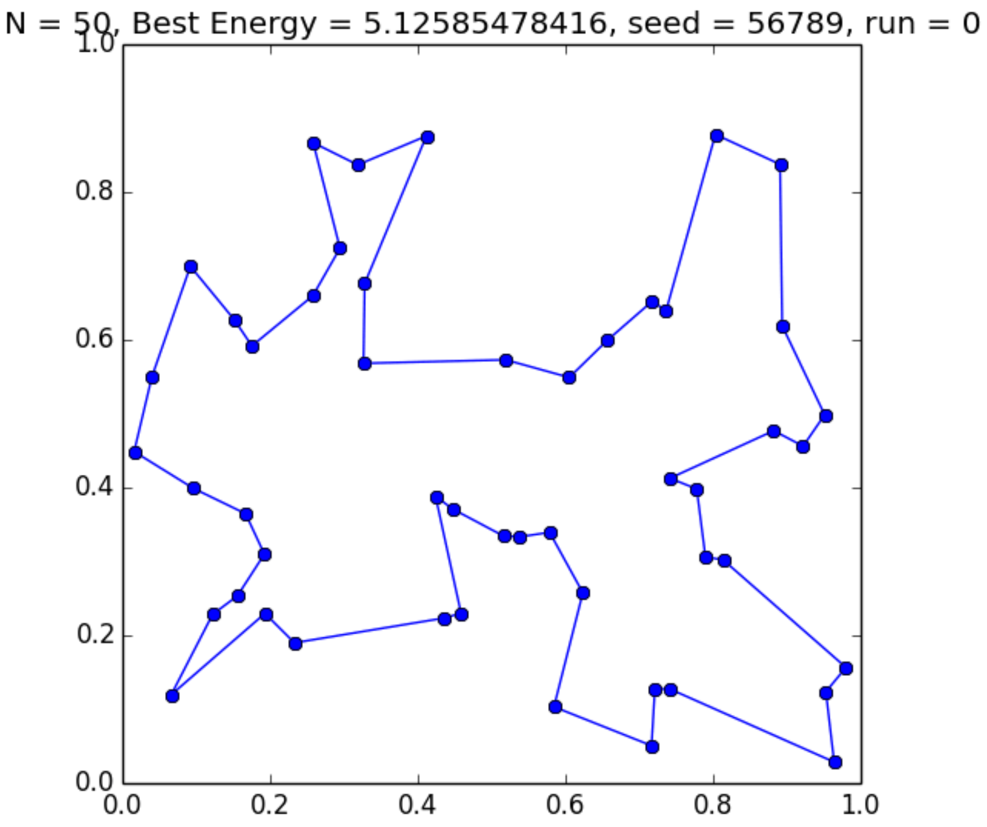


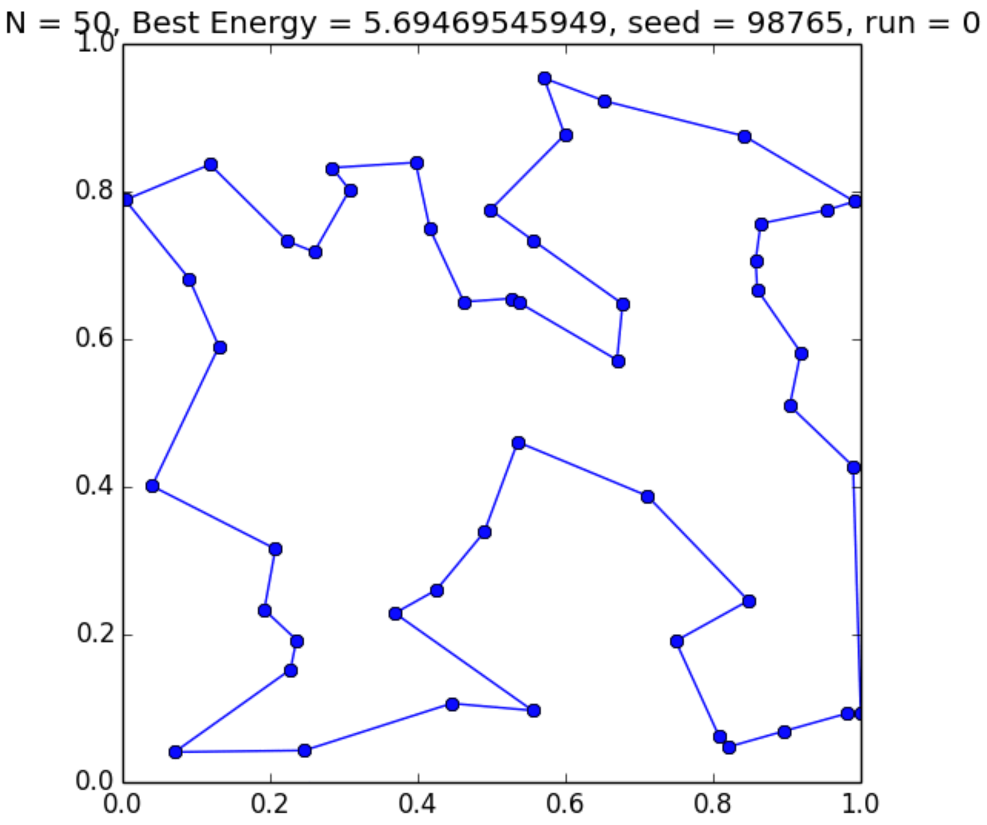


N=50 case for four seeds:


N = 50, Best Energy = 6.27440404384, seed = 12345, run = 0
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0


N = 50, Best Energy = 5.88321269208, seed = 54321, run = 0
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0





Evaluation/feedback on the above work

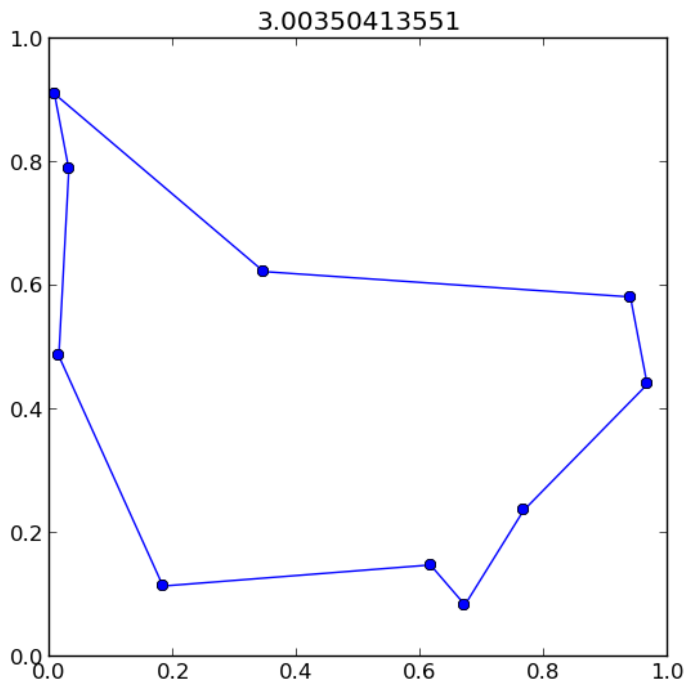
Note: this section can only be filled out during the evaluation phase.

Here you check whether your fellow student is able to to run a simulated annealing program. There are three system sizes:

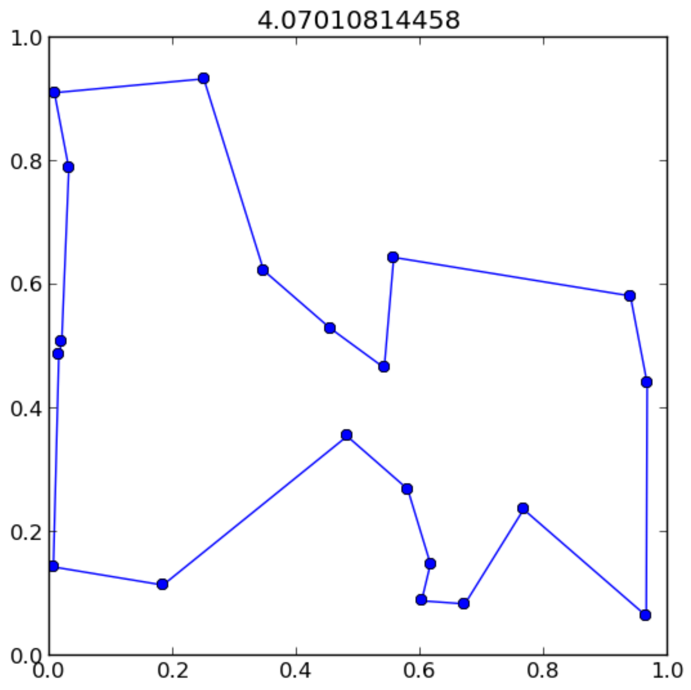
1. Small size: Annealing equivalent to direct sampling
2. Intermediate: Annealing clearly better and capable of obtaining the exact solution
3. Large size: Annealing quite amazing: able to find the optimal solution (but for even larger sizes, this will no longer be possible).

The following answer would have obtained full score:

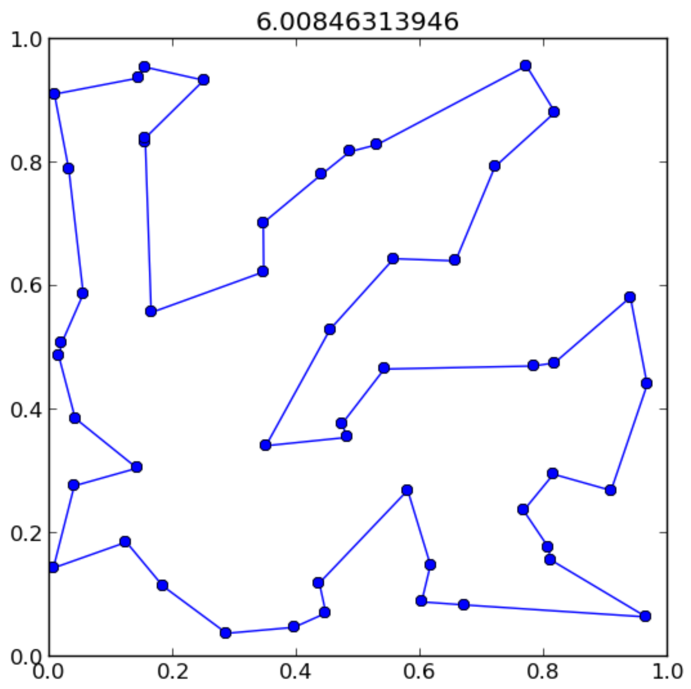
Here, the best solution found by simulated annealing for N=10. I used the beta *= 1.005, as in the program that was provided.



And here, the best solution found by simulated annealing for N=20, for the same parameters. It is clearly better than the direct-sampling solution



Here is the best solution found for N=50, for the same parameters. This solution looks very good, and it might well be the optimal solution (it is also possible that a clearly insufficient solution was found).



POINTS - DESCRIPTION

0 points if nothing was done
1 point if the simulated annealing was tried for N=10 (file uploaded)
2 point if the simulated annealing was tried for N=20 (file uploaded)
3 point if the simulated annealing was tried for N=50 (file uploaded). Note that the conclusions may be different from the ones we arrived at here...

NB:There is a **final-week homework bonus**: If some work was done (for N=10 and N=20 and N=50), your fellow student should get full points.

Score from your peers: **3**

peer 1 → *[This area was left blank by the evaluator.]*

peer 2 → Good job! :)

C3 At the end of this series of exercises, let us thank you all for your interest in working out the solutions. You have done a great job!

The course was challenging at times. However, if one stuck around and persevered, the results were truly rewarding. The "Discussion Forums" truly helped to go over the barriers in the understanding and the spirit of cooperation was great.

Thanks to all the peer-reviewers for their nice feedback. Much appreciated :)