

[Peer Assessments \(https://class.coursera.org/smac-001/human_grading/\)](https://class.coursera.org/smac-001/human_grading/)

/ Homework session 9: Simulated Annealing for sphere packings and the travelling salesman problem

[Help \(https://class.coursera.org/smac-001/help/peergrading?url=https%3A%2F%2Fclass.coursera.org%2Fsmac-001%2Fhuman_grading%2Fview%2Fcourses%2F971628%2Fassessments%2F13%2Fsubmissions\)](https://class.coursera.org/smac-001/help/peergrading?url=https%3A%2F%2Fclass.coursera.org%2Fsmac-001%2Fhuman_grading%2Fview%2Fcourses%2F971628%2Fassessments%2F13%2Fsubmissions)

due in 1wk 1d

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)

☐ In accordance with the Honor Code, I certify that my answers here are my own work, and that I have appropriately acknowledged all external sources (if any) that were used in this work.

[Save draft](#)

[Submit for grading](#)

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

$$T \rightarrow T * (1 - \text{gamma}) \quad (\text{Simulated annealing step})$$

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 $\text{gamma} = 0.5, 0.25, 0.125 \dots 0.002$ (approximately).

Convince yourself that

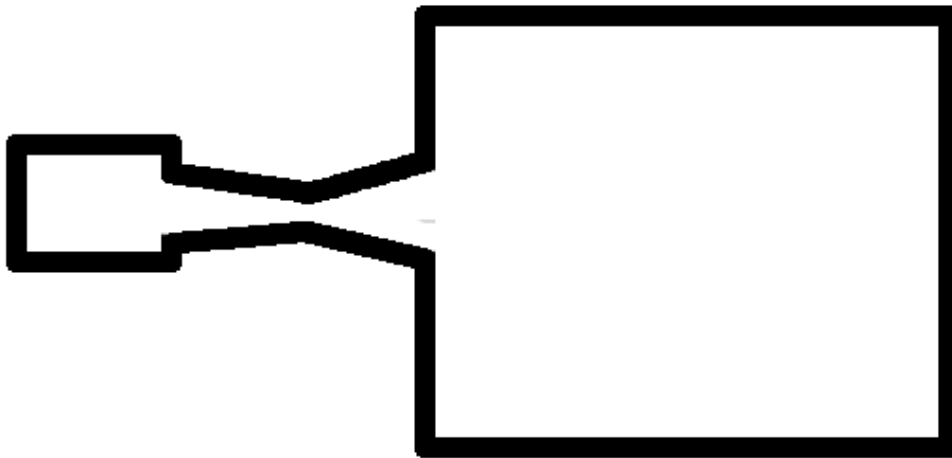
1. for finite annealing rate gamma , the correct solution is not necessarily found in the limit $t \rightarrow \infty$.
2. the correct solution is found with probability 1 for $t \rightarrow \infty$ in the limit of vanishing annealing rate ($\text{gamma} \rightarrow 0$).

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.

[Attach a file](#) (supports: txt, png, jpg, gif, pdf, py)

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 $(x,y) \rightarrow (x + \text{delx}, y + \text{dely})$, 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

$\text{sigma} \rightarrow \text{sigma} * (1 + \text{gamma})$ with $\text{gamma} > 0$ (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 $\text{gamma} \rightarrow 0$ in the limit $t \rightarrow \text{infinity}$. **Write down a short explanation** of why this is so.

B	<i>I</i>			 Link	<code><code></code>	Math		Edit: Rich ▼	Preview

[Attach a file](#) (supports: *txt, png, jpg, gif, pdf, py*)

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 γ , and the control parameter σ , 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 γ 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 γ 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.

B	<i>I</i>			Link	<code>	Math	Edit: Rich ▼	Preview
----------	----------	--	--	------	--------	------	--------------	---------

[Attach a file](#) (supports: txt, png, jpg, gif, pdf, py)

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.**

B	<i>I</i>			Link	<code>	Math		Edit: Rich ▼	Preview

[Attach a file](#) (supports: txt, png, jpg, gif, pdf, py)

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.

B	<i>I</i>			Link	<code>	Math		Edit: Rich ▼	Preview

[Attach a file](#) (supports: txt, png, jpg, gif, pdf, py)

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).**

B

I

≡

≡

Link

<code>

Math

Edit: Rich ▼

Preview

[Attach a file](#) (supports: *txt, png, jpg, gif, pdf, py*)

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], cities[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[1:]
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.

B	<i>I</i>			Link	<code>	Math		Edit: Rich ▼	Preview

[Attach a file](#) (supports: txt, png, jpg, gif, pdf, py)

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!

B	<i>I</i>			Link	<code>	Math		Edit: Rich ▼	Preview

[Attach a file](#) (supports: txt, png, jpg, gif, pdf, py)

☐ In accordance with the Honor Code, I certify that my answers here are my own work, and that I have appropriately acknowledged all external sources (if any) that were used in this work.

[Save draft](#)

[Submit for grading](#)

