INTRODUCTION to ARTIFICIAL INTELLIGENCE
2015-2016 FALL SEMESTER
LABORATORY MANUAL

Experiment 6

Simulated Annealing Algorithm

Simulated annealing is a method for finding a good (not necessarily perfect) solution to an optimization problem. If you're in a situation where you want to maximize or minimize something, your problem can likely be tackled with simulated annealing.

Basic Algorithm

1. First, generate a random solution
2. Calculate its cost using some cost function you've defined
3. Generate a random neighboring solution
4. Calculate the new solution's cost
5. Compare them:
    a. If cnew < cold: move to the new solution
    b. If cnew > cold: maybe move to the new solution
6. Repeat steps 3-5 above until an acceptable solution is found or you reach some maximum number of iterations.

First, generate a random solution: You can do this however you want. The main point is that it's random - it doesn't need to be be your best guess at the optimal solution.

Calculate its cost using some cost function you've defined: Depending on your problem, it might be as simple as counting the total number of miles the traveling salesman's traveled. Or it might be an incredibly complex melding of multiple factors. Calculating the cost of each solution is often the most expensive part of the algorithm, so it pays to keep it simple.

Generate a random neighboring solution: "Neighboring" means there's only one thing that differs between the old solution and the new solution. Effectively, you switch two elements of your solution and re-calculate the cost. The main requirement is that it be done randomly.

Calculate the new solution's cost: Use the same cost function as above. You can see why it needs to perform well - it gets called with each iteration of the algorithm.

If cnew < cold: move to the new solution: If the new solution has a smaller cost than the old solution, the new one is better. This makes the algorithm happy - it's getting closer to an optimum. It will "move" to that new solution, saving it as the base for its next iteration.

If cnew > cold: maybe move to the new solution: This is where things get interesting. Most of the time, the algorithm will be moving to a worse solution. If it did that all of the time, though, it would get caught at local maxima. To avoid that problem, it sometimes selects to keep the worse solution. To decide, the algorithm calculates something called the 'acceptance probability' and then compares it to a random number.

NOTE: The explanation above leaves out an extremely important parameter called the "*temperature*". The temperature is a function of which iteration you're on; its name comes from the fact that this algorithm was inspired by a method of heating and cooling metals.

Usually, the temperature is started at 1.0 and is decreased at the end of each iteration by multiplying it by a constant called α. You get to decide what value to use for α; typical choices are between 0.8 and 0.99.

Furthermore, simulated annealing does better when the neighbor-cost-compare-move process is carried about many times (typically somewhere between 100 and 1,000) at each temperature.

Example Code

```python
from random import random

def anneal(sol):
    old_cost = cost(sol)
    T = 1.0
    T_min = 0.00001
    alpha = 0.9
    while T > T_min:
        i = 1
        while i <= 100:
            new_sol = neighbor(sol)
            new_cost = cost(new_sol)
            ap = acceptance_probability(old_cost, new_cost, T)
            if ap > random():
                sol = new_sol
                old_cost = new_cost
            i += 1
        T = T*alpha
    return sol, cost
```

This skeleton leaves a few gaps for you to fill in: **neighbor()**, in which you generate a random neighboring solution, **cost()**, in which you apply your cost function, and **acceptance_probability()**, which is basically defined for you.

The acceptance probability function: The acceptance probability function takes in the old cost, new cost, and current temperature and spits out a number between 0 and 1, which is a sort of recommendation on whether or not to jump to the new solution. For example:

- 1.0: definitely switch (the new solution is better)
- 0.0: definitely stay put (the new solution is infinitely worse)
- 0.5: the odds are 50-50

Once the acceptance probability is calculated, it's compared to a randomly-generated number between 0 and 1. If the acceptance probability is larger than the random number, you're switching!

Calculating the acceptance probability: The equation typically used for the acceptance probability is:

$$a = e^{\frac{c_{new} - c_{old}}{T}}$$

where a is the acceptance probability, (cnew−cold) is the difference between the new cost and the old one, T is the temperature, and e is 2.71828, that mathematical constant that pops up in all sorts of unexpected places.

This equation is the part of simulated annealing that was inspired by metalworking. Throw in a constant and it describes the embodied energy of metal particles as they are cooled slowly after being subjected to high heat. This process allows the particles to move from a random configuration to one with a very low embodied energy. Computer scientists borrow the annealing equation to help them move from a random solution to one with a very low cost.

This equation means that the acceptance probability:

- is always > 1 when the new solution is better than the old one. Since you can't have a probability greater than 100%, we use α=1 in this case..
- gets smaller as the new solution gets more worse than the old one.
- gets smaller as the temperature decreases (if the new solution is worse than the old one)

What this means is that the algorithm is more likely to accept sort-of-bad jumps than really-bad jumps, and is more likely to accept them early on, when the temperature is high.

Exercise:

	J1	**J2**	**J3**
J1	-	2	4
J2	5	-	6
J3	7	8	-

Tablo 1: Bir işten diğer işe geçerken hazırlanma süresi

Tablo 1'de bir işten diğer işe geçerken gereken hazırlanma süresini göstermektedir. Örneğin, 1. işten (J1) 2. işe (J2) geçerken gereken bekleme süresi 2 birimdir.

	J1	**J2**	**J3**
M1	4	5	3
M2	8	9	3

Tablo 2: Bir işin bir makinede tamamlanma zamanı

Tablo 2'de bir işin bir makinede tamamlanma zamanını göstermektedir. Örneğin, 1. işin (J1) 1. makinede (M1) tamamlanma zamanı 4 birimdir.

Tablo1 ve 2'deki değerleri de dikkate alarak tüm işlerin tüm makineler kullanılarak tamamlanma zamanlarını minimize edecek simulated annealing algoritmasını kodlayınız. Makinelerin aynı anda çalışmaya başlamalarını dikkate alınız.

Örnek çözüm: (En iyi çözüm olması garanti edilmemektedir.)

M1	J1	Hazırlanma Süresi (J1-J2)	J2
	4	2	5
M2	J3		
	3		

J1’in tamamlanma zamanı=4

J2’nin tamamlanma zamanı=4+2+5=11

J3’ün tamamlanma zamanı=3

Toplam tamamlanma zamanı (J1, J2, J3)=4+11+3=18