CSL020U4E: Artificial Intelligence
Lecture 04 *(Stochastic Local Search)*

Sumit Kumar Pandey

August, 2025

Introduction

- We introduce *stochastic moves* to add an element of randomness to search.
- *Exploiting* the gradient deterministically has its drawbacks when the heuristic functions are imperfect, as they often are.
- The steepest gradient can lead to the nearest optimum and end there.
- We add a tendency of *exploration*, which could drag search away from the path to local optima.

Random Walk

Random Walk: Pure Exploration

- Exploitation entails following the indicated path.
- Exploration is the exact opposite.
- Pure exploration is to move in a random direction at every step.
- Mathematicians have termed such sequences of moves a random walk.
- A random walk has no sense of direction. Of the available choices, we select one at random.
- In a 1-dimensional random walk on integers, one starts with a given value, and either adds one or subtracts one at each step.

Random Walk

```
RandomWalk()
  node ← random candidate solution or start
  bestNode ← node
  for i ← 1 to n
    node ← RandomChoose(MoveGen(node))
    if node is better than bestNode
      then bestNode ← node
  return bestNode
```

Stochastic Hill Climbing

Stochastic Hill Climbing

- In the algorithm RandomWalk, all moves are equal.
- But that means the algorithm is completely oblivious of the gradient. It has no sense of direction whatsoever, and no heuristic function to draw it towards the goal node.
- We do want our search to be attracted towards the goal.
- That is what HillClimbing did single-mindedly, doing only exploitation.
- Now we inject an element of randomness into an algorithm that does prefer the gradient but is not bound by it.
- We call this algorithm StochasticHillClimbing.

Stochastic Hill Climbing

- Let C be the current node, and let N be the random neighbour being considered.
- Let $\textsc{Eval}(N)$ be the objective function for our optimization problem, the value that we want to optimize.
- This could well be the heuristic value $h(N)$ if we were doing heuristic guided search, as in $\textsc{HillClimbing}$.
- We define the gradient ΔE as the difference between the values of N and C.

$$\Delta E = \textsc{Eval}(N) - \textsc{Eval}(C)$$

- We move from C to N with some probability depending upon ΔE.
- For all neighbours N, this probability is non-zero, even if the neighbour is worse than the current node.

Stochastic Hill Climbing

- For a maximization problem, the larger the ΔE, the better is node N, and the higher should be the probability of moving from node C to node N.
- A function that serves our purpose is the *sigmoid* function.
- The probability P of making the move is given by

$$P = 1/(1 + e^{-\Delta E/T})$$

  where T is the parameter that determines the shape of the curve.
- Observe that the sigmoid function always evaluates to a value between 0 and 1.

ΔE *and* T *in the Sigmoid Function*

- As ΔE tends to $-\infty$, the probability tends to 0, This means that for a very bad neighbour, the probability will be very low.
- Conversely, as ΔE tends to $+\infty$ the probability tends to 1. Thus, StochasticHillClimbing moves to better neighbours with a higher probability than to worse ones.

ΔE *and* T *in the Sigmoid Function*

- When $T \to \infty$, then the probability P is 0.5 for all values of ΔE. This means that the probability does not depend on ΔE and is therefore like a random walk. Purely explorative.
- On the other hand, as T $\to$ 0, the sigmoid function approaches a step function, with the probability being 0 for all ΔE less than 0, and 1 for all values greater than 0. Thus it is totally deterministic, like HillClimbing.
- If the neighbour is better, it always moves to it, otherwise never. Purely exploitative.
- For values of T between 0 and 1 the behaviour of stochastic hill climbing is a blend of exploitation and exploration.
- The higher the value of T, the more random the movement, and the lower the value, the more deterministic it is.

Stochastic Hill Climbing

StochasticHillClimbing(T, start)
  node $\leftarrow$ *start*
  bestNode $\leftarrow$ *node*
  while some termination criteria
    neighbour $\leftarrow$ RandomNeighbour(*node*)
    $\Delta E \leftarrow$ Eval(*neighbour*) - Eval(*node*)
    if Random$(0, 1) < 1/(1 + e^{-\Delta E/T})$
      node $\leftarrow$ *neighbour*
      if Eval(*node*) $>$ Eval(*bestnode*)
        bestNode $\leftarrow$ *node*
  return *bestNode*

Simulated Annealing

```
SimulatedAnnealing(start, numberOfEpochs)
  node ← start
  bestNode ← node
  T ← some large value
  for time ← 1 to numberOfEpochs
    while some termination criterion do
      neighbour ← RandomNeighbour(node)
      ΔE ← Eval(neighbour) - Eval(node)
      if Random(0, 1) < 1/(1 + e^{−ΔE/T})
        then node ← neighbour
          if Eval(node) > Eval(bestNode)
            then bestNode ← node
    T ← CoolingFunction(T, time)
  return bestNode
```

Genetic Algorithms

Genetic Algorithms

- In the solution space search algorithms we have seen so far, new candidates are generated from old ones by perturbation of a parent candidate.
- The neighbourhood function makes a small change in a candidate, to produce a variant in the neighbourhood.
- This is possible when candidates and solutions are made up of components, which can be replaced by other components.
- The candidate can be thought of as a chromosome, and the components themselves as genes.
- GeneticAlgorithms take a cue from nature and produce a new candidate from two parents. This is done by a process called crossover, which does this mixing up of genes.

Genetic Algorithms

A GeneticAlgorithm is a process of starting with a population of N chromosomes, or candidates, and producing new candidates by trying different combinations of the genes, or components, inherited from two parents at a time. The following three steps are repeated until some termination criterion is reached.

1. **Reproduction**. Produce a new population in N cycles. In each cycle select one member with probability proportional to its fitness and add it to the new population.
2. **Crossover**. Randomly pair the resulting population, and for each pair do a random mixing up of genes, or components.
3. **Mutation**. Once in a while randomly replace a gene in some candidate with another one.

Example-1

- Consider a 7-variable SAT problem. Given a candidate 1111111 and a perturbation function $N1$ which flips one bit, SimulatedAnnealing will generate a neighbour which is one of 0111111, 1011111, 1101111, 1110111, 1111011, 1111101, and 1111110.
- The GeneticAlgorithm generates a child from two parents. Let the two parents be 1111111 and 0000000.
- Then the single point crossover would choose a random point in the two chromosomes and create a child with the left part of one and the right part of the second.
- The first child would be one of 1000000, 1100000, 1110000, 1111000, 1111100, and 1111110.

Example-2

Consider the problem of finding a 5-bit string over the alphabet $\{0, 1\}$ where X is the number represented by the string and the fitness function is the value X^2. Let us say we start with four strings:

$$M_1 = 01101 \text{ with } f(M_1) = 169$$
$$M_2 = 11000 \text{ with } f(M_2) = 576$$
$$M_3 = 01000 \text{ with } f(M_3) = 64$$
$$M_4 = 10011 \text{ with } f(M_4) = 361$$

The total fitness of the population is 1170 and the average fitness is 293. The probabilities of the four candidates being reproduced are as follows:

$$\text{Prob}(M_1) = 169/1170 = 0.14$$
$$\text{Prob}(M_2) = 576/1170 = 0.49$$
$$\text{Prob}(M_3) = 64/1170 = 0.06$$
$$\text{Prob}(M_4) = 361/1170 = 0.31$$

Example-2 (continued...)

As one can see, M_2 has the highest probability of being reproduced, followed by M_4, M_1 and M_3. We spin the roulette wheel four times and let us say that we get two copies of M_2, one of M_4, and one of M_1. The set of parents then is

$$P_1 = 01101$$
$$P_2 = 11000$$
$$P_3 = 11000$$
$$P_4 = 10011$$

Let us say that we (randomly) mate P_1 with P_2, and P_3 with P_4. We apply a single point crossover (randomly) after four bits for the first pair, and after two bits for the second.

Example-2 (continued...)

The set of children we now get along with their fitness values are

$$C_1 = 01100 \text{ with } f(M_1) = 144$$
$$C_2 = 11001 \text{ with } f(M_2) = 625$$
$$C_3 = 11011 \text{ with } f(M_3) = 729$$
$$C_4 = 10000 \text{ with } f(M_4) = 256$$

The total fitness of the population is 1754 and the average fitness is 439. The new population is a fitter population, with C_2 and C_3 being much fitter than the other two.
It is quite likely that in the next cycle both C_2 and C_3 will get two copies each. But if that happens, the third gene, or bit, would have disappeared from the population, and however much we churn the genes after that, we can never generate the candidate 11111 which has the highest possible fitness.

Example-2 (continued...)

To keep the possibility of breaking free from the confines of a restricted gene pool, genetic algorithms take another leaf out of nature's book. Every once in a while some gene is perturbed randomly. This, the third phase, is called **Mutation**. Now since a random move is more likely to be detrimental to the fitness of the candidate, this should be done very rarely. It does not promise a better candidate but keeps the possibility alive.

Genetic-Algorithm()

A high level description of a genetic algorithm. Problem specific decisions to be made are - defining the fitness function, the population size, the crossover operation, the mutation, and the number of members to be carried forward.

Genetic-Algorithm()
  $P \leftarrow$ **create** N **candidate solutions**
  repeat
    compute fitness value for each member of P
    $S \leftarrow$ **with probability proportional to fitness value, randomly selects** N **members from** P
    offspring $\leftarrow$ **partition** S **into two halves, and randomly mate and crossover members to generate** N **offsprings**
    with a low probability mutate some offsprings
    replace k **weakest members of** P **with** k **strongest offsprings**
  until some termination criteria

  return the best member of P

Ant Colony Optimization

Ant Colony Optimization

- Populations of chromosomes in genetic algorithms form a large pool of *candidate solutions* from which new combinations are churned out and presented to the selection phase.
- Ant colonies on the other hand are *populations of agents* going about their simple, often repetitive, business but which gives the colony a higher level of purposefulness and agency not witnessed in individual ants.

Ant Colony Optimization

- The secret behind the success of an ant colony is communication.
- The chemical used by ants is pheromone, which acts as a marker for other ants to follow.
- Wherever an ant goes, it leaves behind a trail of pheromone.
- Whenever any ant finds a lump of sugar, the ant follows its own pheromone trail back to its nest.
- While doing so, it deposits even more pheromone on the trail, out of sheer habit. This has the happy effect of strengthening the pheromone trail.
- Now ants are addicted to pheromone, and other ants that emerge from the nest instinctively follow the strongest trail, further strengthening it as they do so.
- Eventually this positive feedback cycle results in all ants marching along in a single column.

Ant Colony Optimization (ACO)

- Motivated by the behaviour of the ant colony, Marco Dorigo and his associates devised an optimization algorithm in which simple problem solving agents cooperate with each other, finding better and better solutions by following the cues given out by other agents (Colorni, Dorigo, and Maniezzo, 1991; Dorigo, 2004). The algorithm is called *ant colony optimization* (ACO).
- The main idea here is that an army of simple agents repeatedly solves a problem individually, and in each new cycle is influenced by the solutions synthesized by all agents in the previous cycle.

ACO

- In each cycle beginning at time t, the ants start constructing a tour, completing it at time $t + N$.
- At each choice point at City$_i$, an ant chooses a city to move to in a probabilistic manner.
- The probability of moving to City$_j$ is influenced by two factors. One, η_{ij}, called visibility, which is inversely proportional to the cost of the edge between City$_i$ and City$_j$.
- The other, $\tau_{ij}(t)$, is the total pheromone on edge$_{ij}$ at the beginning of the cycle at time t.
- The two factors are moderated by parameters α and β.

The probability of the kth ant moving from City$_i$ to City$_j$ in the cycle beginning at time t is given by

$$P_{ij}^{k}(t) = \begin{cases} \dfrac{(\tau_{ij}(t))^{\alpha}(\eta_{ij})^{\beta}}{\sum_{h \in \text{allowed}_k(t)} (\tau_{ih}(t))^{\alpha}(\eta_{ih})^{\beta}} & \text{if } j \in \text{allowed}_k(t) \text{ the cities ant}_k \text{ is allowed to move to} \\ 0 & \text{otherwise.} \end{cases}$$

ACO

- In the real world, an ant constantly deposits pheromone as it ambles along.
- If the path that it travels is a short one, it returns quickly with food and deposits more pheromone on the way back.
- In this way shorter paths have more pheromone, and an ant colony finds the shortest path to the lump of food.
- The amount of pheromone deposited is inversely proportional to the length L_k of the tour constructed by the kth ant.
- After constructing a tour in N time steps, each ant k deposits an amount of pheromone Q/L_k on all the edges it traversed, where Q is a parameter the user can control.

ACO

- Being a chemical substance, pheromone evaporates in real life.
- In ACO, this phenomenon is incorporated by a parameter ρ, which is the rate of evaporation.
- We assume that a fraction proportional to ρ evaporates in every cycle.
- The total pheromone on edge$_{ij}$ after the cycle is over is

$$\tau_{ij}(t+n) = (1-\rho)\tau_{ij}(t) + \Delta\tau_{ij}(t, t+n)$$

where $\Delta\tau_{ij}(t, t+n)$ is the pheromone deposited by all the ants in the cycle beginning at time t and ending at time $t+N$.

ACO

j \ i	1	2	3	4	5
1	–	10	3	6	18
2	10	–	2	20	9
3	3	2	–	12	7
4	6	20	12	–	15
5	18	9	7	15	–

Table d_{ij} for edge ij

j \ i	1	2	3	4	5
1	–	1	1	1	1
2	1	–	1	1	1
3	1	1	–	1	1
4	1	1	1	–	1
5	1	1	1	1	–

Table τ_{ij} for edge ij

j \ i	1	2	3	4	5
1	–	10/10	10/3	10/6	10/18
2	10/10	–	10/2	10/20	10/9
3	10/3	10/2	–	10/12	10/7
4	10/6	10/20	10/12	–	10/15
5	10/18	10/9	10/7	10/15	–

Table η_{ij} for edge ij

ACO

Suppose there are three ants at nodes 1, 2 and 4 respectively. Let $\alpha = 1$ and $\beta = 1$. Let's calculate the probability of the movement of ant 1.

1. $$S = (\tau_{12} \cdot \eta_{12}) + (\tau_{13} \cdot \eta_{13}) + (\tau_{14} \cdot \eta_{14}) + (\tau_{15} \cdot \eta_{15}) = 590/90.$$
$$P_{12} = (\tau_{12} \cdot \eta_{12})/S = 0.15, \quad P_{13} = (\tau_{13} \cdot \eta_{13})/S = 0.5,$$
$$P_{14} = (\tau_{14} \cdot \eta_{14})/S = 0.25, P_{15} = (\tau_{15} \cdot \eta_{15})/S = 0.1.$$

   Let the ant chooses the edge 3.

2. $$S = (\tau_{32} \cdot \eta_{32}) + (\tau_{34} \cdot \eta_{34}) + (\tau_{35} \cdot \eta_{35}) = 610/84.$$
$$P_{32} = (\tau_{32} \cdot \eta_{32})/S = 0.69, \quad P_{34} = (\tau_{34} \cdot \eta_{34})/S = 0.11,$$
$$P_{35} = (\tau_{35} \cdot \eta_{35})/S = 0.2$$

   Let the ant chooses the edge 5.

3. $$S = (\tau_{52} \cdot \eta_{52}) + (\tau_{54} \cdot \eta_{54}) = 80/45$$
$$P_{52} = (\tau_{52} \cdot \eta_{52})/S = 0.63, \quad P_{54} = (\tau_{54} \cdot \eta_{54})/S = 0.37$$

   Let the ant chooses the edge 4.

4. There is only one edge left to complete the tour. The ant chooses the edge 2.

The tour of ant 1 is

$$1 - 3 - 5 - 4 - 2 - 1$$

Suppose there are three ants, and their tours are

1. $1-3-5-4-2-1$
2. $2-1-4-3-5-2$
3. $4-5-1-3-2-4$

Let $Q = 100$. Note that

1. $L_1 = 3+7+15+20+10 = 55,\ Q/L_1 = 100/55 = 1.82,$
2. $L_2 = 10+6+12+7+9 = 44,\ Q/L_2 = 100/44 = 2.27,$
3. $L_3 = 15+18+3+2+20 = 58,\ Q/L_3 = 100/58 = 1.72.$

Hence,

1. $\Delta\tau_{13} = \Delta\tau_{13}^{(1)} + \Delta\tau_{13}^{(2)} + \Delta\tau_{13}^{(3)} = 1.82+0+1.72 = 3.54,$
2. $\Delta\tau_{35} = \Delta\tau_{35}^{(1)} + \Delta\tau_{35}^{(2)} + \Delta\tau_{35}^{(3)} = 1.82+2.27+0 = 4.09.$
3. $\Delta\tau_{45} = \Delta\tau_{45}^{(1)} + \Delta\tau_{45}^{(2)} + \Delta\tau_{45}^{(3)} = 1.82+0+1.72 = 3.54.$

ACO

1. $\Delta\tau_{12} = \Delta\tau_{12}^{(1)} + \Delta\tau_{12}^{(2)} + \Delta\tau_{12}^{(3)} = 1.82 + 2.27 + 0 = 4.09,$
2. $\Delta\tau_{13} = \Delta\tau_{13}^{(1)} + \Delta\tau_{13}^{(2)} + \Delta\tau_{13}^{(3)} = 1.82 + 0 + 1.72 = 3.54,$
3. $\Delta\tau_{14} = \Delta\tau_{14}^{(1)} + \Delta\tau_{14}^{(2)} + \Delta\tau_{14}^{(3)} = 0 + 2.27 + 0 = 2.27,$
4. $\Delta\tau_{15} = \Delta\tau_{15}^{(1)} + \Delta\tau_{15}^{(2)} + \Delta\tau_{15}^{(3)} = 0 + 0 + 1.72 = 1.72,$
5. $\Delta\tau_{23} = \Delta\tau_{23}^{(1)} + \Delta\tau_{23}^{(2)} + \Delta\tau_{23}^{(3)} = 0 + 0 + 1.72 = 1.72,$
6. $\Delta\tau_{24} = \Delta\tau_{24}^{(1)} + \Delta\tau_{24}^{(2)} + \Delta\tau_{24}^{(3)} = 1.82 + 0 + 1.72 = 3.54,$
7. $\Delta\tau_{25} = \Delta\tau_{25}^{(1)} + \Delta\tau_{25}^{(2)} + \Delta\tau_{25}^{(3)} = 0 + 2.27 + 0 = 2.27,$
8. $\Delta\tau_{34} = \Delta\tau_{34}^{(1)} + \Delta\tau_{34}^{(2)} + \Delta\tau_{34}^{(3)} = 0 + 2.27 + 0 = 2.27,$
9. $\Delta\tau_{35} = \Delta\tau_{35}^{(1)} + \Delta\tau_{35}^{(2)} + \Delta\tau_{35}^{(3)} = 1.82 + 2.27 + 0 = 4.09.$
10. $\Delta\tau_{45} = \Delta\tau_{45}^{(1)} + \Delta\tau_{45}^{(2)} + \Delta\tau_{45}^{(3)} = 1.82 + 0 + 1.72 = 3.54.$

Let $\rho = 0.8$

1. $\tau_{12} = (1 - 0.8) \cdot 1 + 4.09 = 0.2 + 4.09 = 4.29.$
2. $\tau_{13} = (1 - 0.8) \cdot 1 + 3.54 = 0.2 + 3.54 = 3.74.$
3. $\tau_{14} = (1 - 0.8) \cdot 1 + 2.27 = 0.2 + 2.27 = 2.47.$
4. $\tau_{15} = (1 - 0.8) \cdot 1 + 1.72 = 0.2 + 1.72 = 1.92.$
5. $\tau_{23} = (1 - 0.8) \cdot 1 + 1.72 = 0.2 + 1.72 = 1.92.$
6. $\tau_{24} = (1 - 0.8) \cdot 1 + 3.54 = 0.2 + 3.54 = 3.74.$
7. $\tau_{25} = (1 - 0.8) \cdot 1 + 2.27 = 0.2 + 2.27 = 2.47.$
8. $\tau_{34} = (1 - 0.8) \cdot 1 + 2.27 = 0.2 + 2.27 = 2.47.$
9. $\tau_{35} = (1 - 0.8) \cdot 1 + 4.09 = 0.2 + 4.09 = 4.29.$
10. $\tau_{45} = (1 - 0.8) \cdot 1 + 3.54 = 0.2 + 3.54 = 3.74.$

ACO

j \ i	1	2	3	4	5
1	–	10	3	6	18
2	10	–	2	20	9
3	3	2	–	12	7
4	6	20	12	–	15
5	18	9	7	15	–

Table d_{ij} for edge ij

j \ i	1	2	3	4	5
1	–	4.29	3.74	2.47	1.92
2	4.29	–	1.92	3.74	2.47
3	3.74	1.92	–	2.47	4.29
4	2.47	3.74	2.47	–	3.74
5	1.92	2.47	4.29	3.74	–

Table τ_{ij} for edge ij

j \ i	1	2	3	4	5
1	–	10/10	10/3	10/6	10/18
2	10/10	–	10/2	10/20	10/9
3	10/3	10/2	–	10/12	10/7
4	10/6	10/20	10/12	–	10/15
5	10/18	10/9	10/7	10/15	–

Table η_{ij} for edge ij

TSP-ACO

TSP-ACO()

bestTour $\leftarrow$ **nil**

repeat

randomly place M **ants on** N **cities**

for each ant a /* Construct Tour */

for $n \leftarrow 1$ **to** N

ant a **selects and edge from the distribution** P_N^a

update bestTour

for each ant a

for each edge (u, v) **in the ant's tour**

deposit pheromone $\propto 1/$**(tour-length) on edge** (u, v)

until some termination criteria

return bestTour

Thank You