Local Search and Optimization

Aditya Challa

BITS Pilani
Pilani Campus

August 31, 2024

Course: Artificial and Computational Intelligence
Lecture No. 5 & 6 & 7

RECAP - Introduction

- Four things they mean when people say AI
  - Acting Humanly - Turing Test - Can a machine exhibit intelligent behavior indistinguishable from a human?
  - Thinking Humanly - Cognitive Science - How do humans think and can machines think like humans?
  - Thinking Rationally - Logic - Can machines think logically?
  - Acting Rationally - Rational Agents - Can machines act rationally?
- There are issues with all of these ideas!
- We focus on the Acting Rationally part - Rational Agents

RECAP - Introduction

- Other important points:
  - Philosophy, Mathematics, Economics, Neuroscience, Psychology, Control Theory, Computer Science, Linguistics - All contribute to AI. So, its a very interdisciplinary field.
  - The history of AI is a series of overexpectations and underdeliveries. But the field has matured a lot in the last 10 years to a *point of no return*!.

RECAP - Introduction

- The broad lessons from history:
  - Economics trumps always. Anything which is not feasible and does not reduce costs might not last. See `https://www.goldmansachs.com/intelligence/pages/gs-research/gen-ai-too-much-spend-too-little-benefit/report.pdf`
  - Humans have an amazing ability to adapt and accept - ChapGPT was a phenomenon in 2021, but after 3 years there are open source models Llama3.1 which can do the same thing.
  - Efficiency is important - Bayesian methods can theoretically solve every problem out there, but it's highly inefficient.
  - What you measure is what you get - Evaluation metrics and designing them are probably the most important value addition humans still make.

RECAP - Introduction

- In this course we are interested in "acting rationally". So, we try to model rational agents. Before that we need to setup the *environment* -

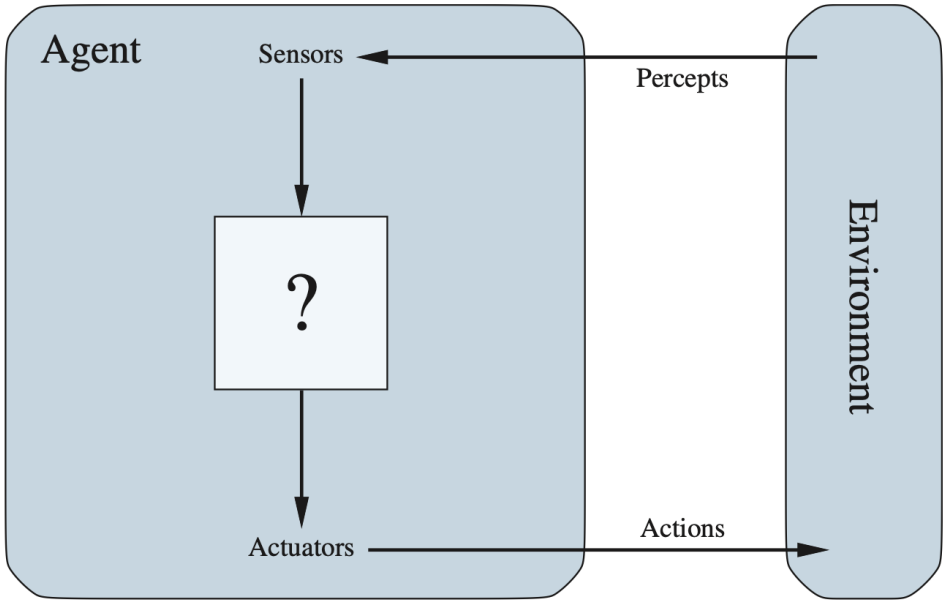

Figure 2.1 Agents interact with environments through sensors and actuators.

RECAP - Introduction

- So our model for rational agent
  - Takes as input a sequence of sensory reading which we will call *percept sequence*
  - Returns as output some actuator movements, which the environment uses.

    For each possible percept sequence, a rational agent should select an action that is expected to maximize its performance measure, given the evidence provided by the percept sequence and whatever built-in knowledge the agent has.

RECAP - Introduction

- The broad aim is going to be to model these rational agents.
  - Question: Do you think it is possible to have a rational agent irrespective of the environment?
  - So, we will try and have some restrictions on the environment as well.
  - We looked a lot of examples – the running example is Autonomous driving.

RECAP - Introduction

• We saw some common architectures (rather design patterns) to model these rational agents:

- A simple table driven action - The agent looks-up the sequence from a long table and returns the action.
- Simple reflex agents - Use markov assumption (essentially forgets percept history)
- Model reflex agents - Builds an internal model of the world
- Goal based agents - Uses goals to dictate the actions, they too build an internal model of the world.
- Utility based agents - Uses "utility" functions to dictate the actions. They try to buils an internal model of performance.
- Learning Agents - They keep on improving the internal models.

RECAP - Introduction

- We also discussed the concept of representations.
  - Atomic Representations are just vectors without any meaning within it.
  - Factored Representations are representations which you can think of as a python dictionary – indicators on a dashboard.
  - Structured Representations are essentially a fixed format where each element has a fixed meaning. We shall look at these in first order logic.
- Local vs Distributed Representations
  - Local Representations separate out the concepts – Each concept is at a memory location.
  - Distributed Representations are where concepts are interleaved with memory locations. Example:Word2Vec.

RECAP - Solving Problems by Searching

- Our first model of AI agents
  - Consider an environment which is episodic, single agent, fully observable, deterministic, static, discrete, and known.
  - The agent is modelled to do the following:
    - Formulate the goal
    - Formulate the Problem
    - Search for a solution
    - Then execute

    The key thing to note, we are not considering changing the actions once the execution starts!

RECAP - Solving Problems by Searching

- We formulated the problem as a graph -
  - State Space of vertices which are possibly infinite (or very large)
  - Initial State
  - Goal State
  - Actions from each state
  - Transition Model
  - Cost of each action
- Two kinds of solutions based on information we have -
  - Uninformed Search
    - Breadth First Search
    - Dijkstra Algorithm
    - Depth First Search
  - Informed Search - Use a heuristic function $h(n)$ to encode the additional information.

RECAP - Solving Problems by Searching

- Informed Search.
  - Take $f(n) = g(n) + h(n)$,
  - A^* search
  - Admissible Heuristic – $h(n)$ does not over-estimate the cost to the goal
  - Consistent Heuristic – $h(n) \leq c(n, a, n') + h(n')$
  - Intuition on why using $h(n)$ helps with search – Search Contours

RECAP - Solving Problems by Searching

- Some tips to design good heuristics
  - Relaxed problems – Allow more edges/vertices and solve the search problem there
  - Combining admissible heuristics – $\max\{h_i(n)\}$
  - Pattern Databases – Store solution costs for some possible patterns
  - Landmarks – Select a subset of vertices and store solution costs between these!

Local Search and Optimization

- Let's make things slightly more complex
  - Recall as assumed in the previous lecture that we know the goal state and we were trying to find an optimal path to it.
  - Instead, here we ask – Suppose each state has an objective value attached to it, and we want to find the state which has the maximum objective value – How to go about this?
- Examples:
  - Any Optimization problem
  - IC design
  - Automatic Programming etc.

Local Search and Optimization

- State Space Landscape – One dimensional caricature

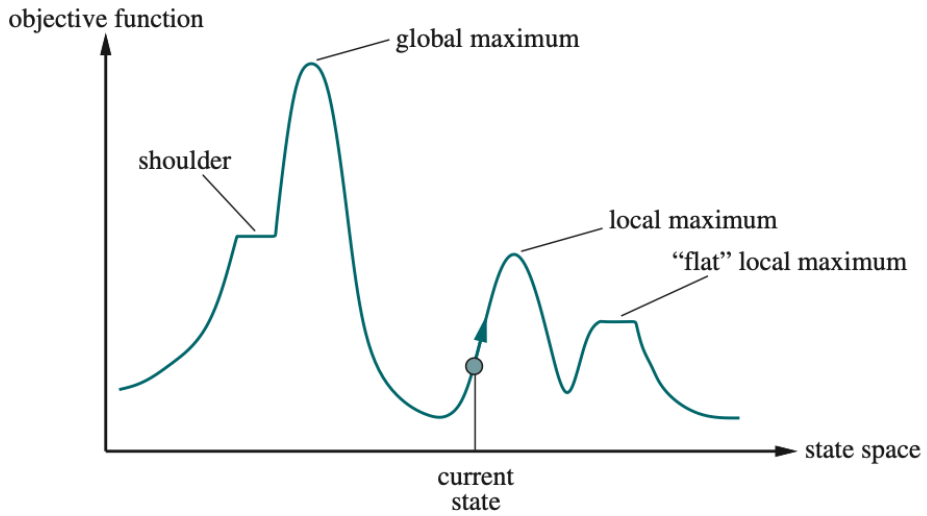

Figure 4.1 A one-dimensional state-space landscape in which elevation corresponds to the objective function. The aim is to find the global maximum.

Local Search and Optimization

- Hill-Climbing Search
  - Let's start with the most naive algorithm we can think of - Go to the state which has the "best" objective value!

```
function HILL-CLIMBING(problem) returns a state that is a local maximum
  current ← problem.INITIAL
  while true do
    neighbor ← a highest-valued successor state of current
    if VALUE(neighbor) ≤ VALUE(current) then return current
    current ← neighbor
```

Figure 4.2 The hill-climbing search algorithm, which is the most basic local search technique. At each step the current node is replaced by the best neighbor.

Local Search and Optimization

- Hill Climbing Search – 8 queens problem

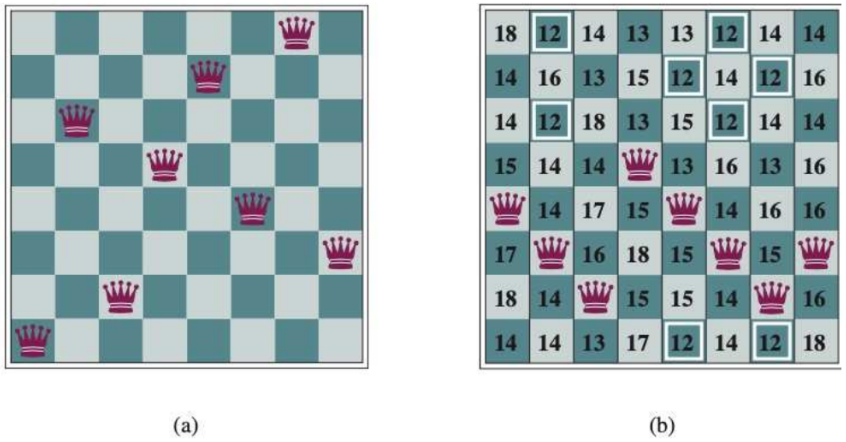

Figure 4.3 (a) The 8-queens problem: place 8 queens on a chess board so that no queen attacks another. (A queen attacks any piece in the same row, column, or diagonal.) This position is almost a solution, except for the two queens in the fourth and seventh columns that attack each other along the diagonal. (b) An 8-queens state with heuristic cost estimate $h=17$. The board shows the value of h for each possible successor obtained by moving a queen within its column. There are 8 moves that are tied for best, with $h=12$. The hill-climbing algorithm will pick one of these.

- The aim is to have one queen per column so that no two queens attack eachother.
  - There are 56 pairs of queens and hence 56 possible conflicts.
  - We can take cost to be *negative* of number of conflicts!

Local Search and Optimization

- Hill Climbing Search – Advantages:
  - Fast – Does not consider sequence of actions! Trying to find a sequence of actions leads to very large search space at each step.
  - Memory efficient – Observe that the agent only has to remember it's current state and possible actions.

Local Search and Optimization

- Hill Climbing Search – Issues
  - Local Maximum – We can only ensure that it reaches local maximum
  - Saddle Points (a.k.a Ridges) – Places where you have to compromise in order to get better!
  - Plateaus – Places where the objective values does not change at all. (Question: Should all Plateaus be minima/maxima??)

Local Search and Optimization

- Hill Climbing Search – How to improve?
  - Repeat the algorithm from different starting points – This increases the chance of getting an global optimum. (Intuition??)
  - Stop being greedy – Select one of the better solutions instead of best solution!
  - Anything else??

Local Search and Optimization

- Simulated Annealing
  - Derives it's name from *annealing* – where we quickly heat the materials and gradually cool down.

function SIMULATED-ANNEALING(*problem*, *schedule*) **returns** a solution state
  current ← *problem*.INITIAL
  for $t = 1$ **to** ∞ **do**
    $T \leftarrow schedule(t)$
    if $T = 0$ **then return** *current*
    next ← a randomly selected successor of *current*
    $\Delta E \leftarrow$ VALUE(*current*) – VALUE(*next*)
    if $\Delta E > 0$ **then** *current* ← *next*
    else *current* ← *next* only with probability $e^{\Delta E/T}$

Figure 4.5 The simulated annealing algorithm, a version of stochastic hill climbing where some downhill moves are allowed. The *schedule* input determines the value of the "temperature" T as a function of time.

- Remark: the above algorithm considers *minimization* problem instead of maximization!

Local Search and Optimization

• Intuition behind Simulated Annealing (also applies to cyclic learning rate schedules) – (Continuous case)

- Consider a loss-landscape where the maxima are same but some minima are deeper than others.
- Apriori, we do not know where the "good" states are to exploit.
- By increasing the temperature we will get to one of the maxima randomly and then exploit it by reducing the temperature.

Local Search and Optimization

- Local Beam Search
  - We already saw this idea when discussing the case where we remember *top-k* elements of the frontier!
  - In this situation,
    - Start with k random states,
    - Process the successors of all the k random states
    - Keep the "best" k from the successors
    - Repeat

Genetic Algorithm

- Understanding Evolution using abstraction
  - Abstract each organism as a vector (forget species etc.)
  - The following steps repeat infinitely in Evolution:
    - Selection of the individuals to form next species
    - Cross-Over (a.k.a reproduction) of the selected species
    - mutations which embeds random noise
    - Survival of the fittest (a.k.a Elitism)

Genetic Algorithm

- Example

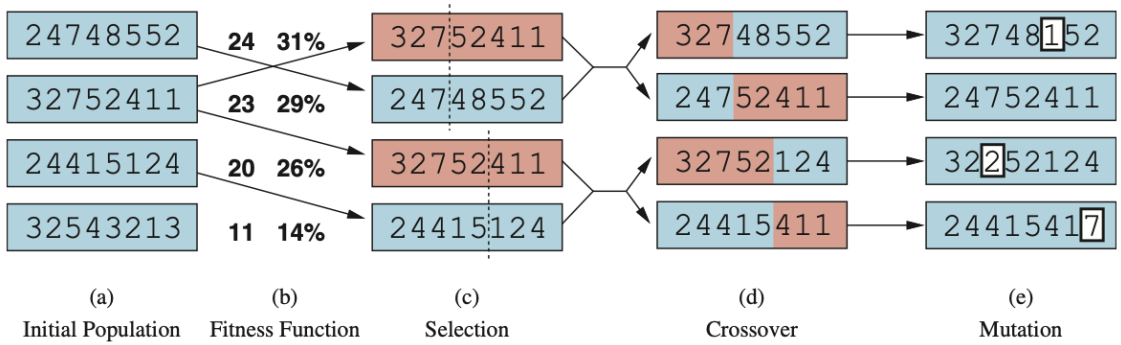

Figure 4.6 A genetic algorithm, illustrated for digit strings representing 8-queens states. The initial population in (a) is ranked by a fitness function in (b) resulting in pairs for mating in (c). They produce offspring in (d), which are subject to mutation in (e).

Genetic Algorithm

• Recall that we have a large state space and we want to search it for an optimal state.

- Assume each state is a vector of n real numbers - $(x_1, x_2, \cdots, x_n)$,
- We first **encode** the vector as a string of binary digits (other options are possible but outside the scope) These are called **chromosomes**
- Generate the **Population** by initializing N chromosomes randomly.
- Compute the objective value for each chromosome – called **fitness** value,
- Repeat the following (just like abstract evolution):
  - Select the most fit individuals **(Selection)**
  - Reproduction or Cross-Over
  - Mutate the individuals
  - Replace the individuals

Genetic Algorithm

- Example

1. Initialize the population by randomly generating N binary strings (chromosomes) $c_i, i = 1, \ldots, N$ of length $m = kn$, where k denotes the number of bits per variable.
2. Evaluate the individuals:
   - 2.1. Decode chromosome c_i to form the corresponding variables $x_{ij}, j = 1, \ldots, n$ (or, in vector form, $\mathbf{x}_i$).
   - 2.2. Evaluate the objective function f using the variable values obtained in the previous step, and assign a fitness value $F_i = f(\mathbf{x}_i)$.
   - 2.3. Repeat steps 2.1 and 2.2 until the entire population has been evaluated.
3. Form the next generation:
   - 3.1. Select two individuals i_1 and i_2 from the evaluated population, such that individuals with high fitness have a greater probability of being selected than individuals with low fitness.
   - 3.2. Generate two new chromosomes by crossing the two selected chromosomes c_{i_1} and c_{i_2}.
   - 3.3. Mutate the two chromosomes generated in the previous step.
   - 3.4. Repeat steps 3.1–3.3 until N new individuals have been generated. Then replace the N old individuals by the N newly generated individuals.
4. Return to step 2, unless the termination criterion has been reached.

Algorithm 3.1: *Basic genetic algorithm. See the main text for a complete description of the algorithm.*

Genetic Algorithm

- Visualizing the progress of Genetic Algorithm

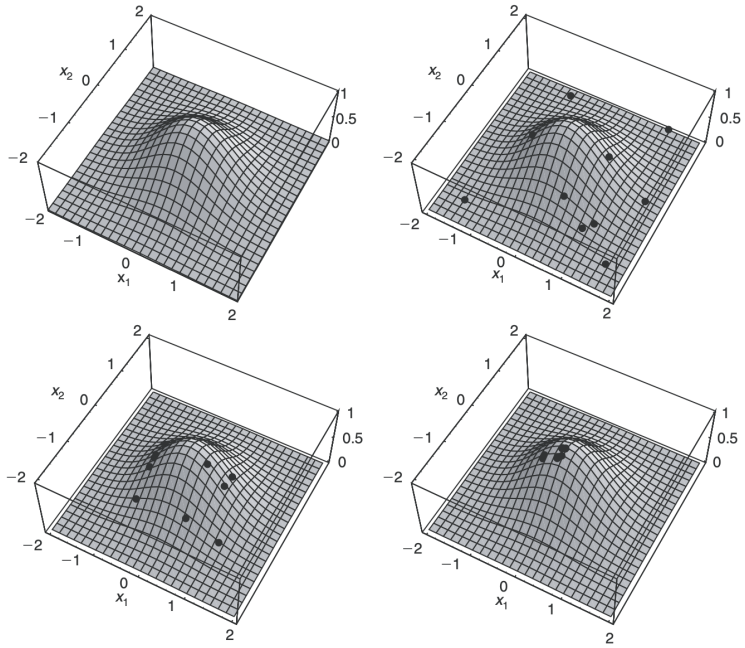

Figure 3.7: *The progress of a GA searching for the maximum of the function* $f(x_1, x_2) = e^{-x_1^2 - x_2^2}$. *The upper right panel shows the initial population, whereas the lower left and lower right panels show the population after 2 and 25 generations, respectively.*

- Let's now dive into few more details and standard genetic algorithm

Genetic Algorithm

- **Encoding Scheme:** There are several ways to encode the state information as a chromosome. The most used one is binary encoding
  - IF x_i is the real number, and $(g_1, g_2, \cdots, g_k)$ is it's binary representation

$$x_i = -d + \frac{2d}{1 - 2^{-k}} \left(2^{-1} g_{1,i} + \cdots + 2^{-k} g_{k,i} \right) \tag{1}$$

  - To get the chromosome for the state, simply concatenate all the binary representations.

Genetic Algorithm

- **Selection:** Selecting the individuals to form the next generation
  - **Roulette-Wheel:** Select each individual with the probability proportional to it's fitness value.
  - **Tournament Selection:** Select two individuals at random (uniformly), and select the best among these with probability p_{tour} and worst with probability $1 - p_{tour}$
  -

Genetic Algorithm

- Cross-Over

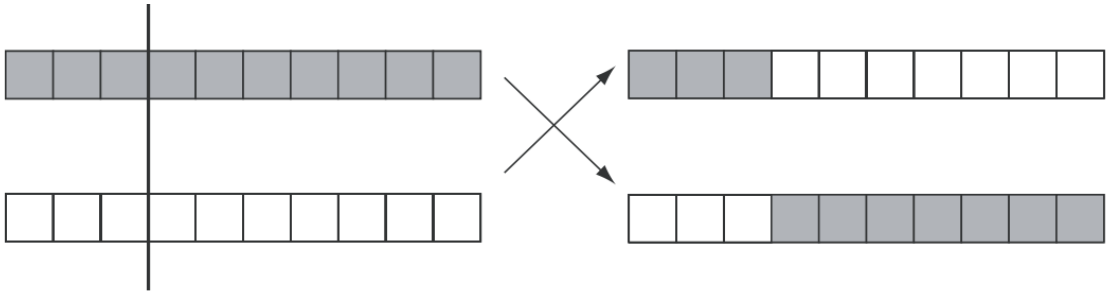

Figure 3.5: *The standard crossover procedure used in GAs. Each square corresponds to a gene. The crossover point, which is indicated by a thick line, is chosen randomly.*

- Take a random point in the chromose, and perform cut and paste to get new individuals. This is called single point cross-over.

Genetic Algorithm

- Mutation

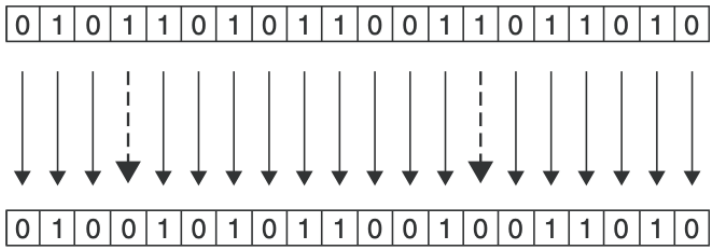

Figure 3.6: *The mutation procedure. Each gene is mutated with a small probability* p_{mut}. *In this case two mutations, marked with dashed arrows, occurred.*

- In mutation, we flip the symbols $0 \leftrightarrow 1$ with some probability p_{mut}.

Genetic Algorithm

- Replacement
  - We repeat the above steps N times to generate N individuals and replace the population.
  - **Elitism:** Since, you might run the risk of loosing the "best" individual, we make $2-4$ copies of the best individual and add this to the next generation.

Genetic Algorithm

- Typical Genetic Algorithm

1. Initialize the population by randomly generating N binary strings (chromosomes) c_i, $i = 1, \ldots, N$ of length $m = kn$, where k denotes the number of bits per variable.
2. Evaluate the individuals:
   - 2.1. Decode chromosome c_i to form the corresponding variables x_{ij}, $j = 1, \ldots, n$.
   - 2.2. Evaluate the objective function f using the variable values obtained in the previous step, and assign a fitness value $F_i = f(\mathbf{x}_i)$.
   - 2.3. Repeat steps 2.1–2.2 until the entire population has been evaluated.
   - 2.4. Set $i_{\text{best}} \leftarrow 1, F_{\text{best}} \leftarrow F_1$. Then loop through all individuals; if $F_i > F_{\text{best}}$, then $F_{\text{best}} \leftarrow F_i$ and $i_{\text{best}} \leftarrow i$.
3. Form the next generation:
   - 3.1. Make an exact copy of the best chromosome $c_{i_{\text{best}}}$.
   - 3.2. Select two individuals i_1 and i_2 from the evaluated population, using a suitable selection operator.
   - 3.3. Generate two offspring chromosomes by crossing, with probability p_c, the two chromosomes c_{i_1} and c_{i_2} of the two parents. With probability $1 - p_c$, copy the parent chromosomes without modification.
   - 3.4. Mutate the two offspring chromosomes.
   - 3.5. Repeat steps 3.2–3.4 until $N - 1$ additional individuals have been generated. Then replace the N old individuals by the N newly generated individuals.
4. Return to step 2, unless the termination criterion has been reached.

Algorithm 3.2: *A standard genetic algorithm applied to the case of function maximization. See the main text for a detailed description.*

Ant Colony Optimization

• Another example of using nature-inspired algorithms is based on the behavior of ants.

- Generally, if these is a “central command” the problems are easy.
- However, if you want a truly distributed parallel algorithm, you should reduce the dependence of these “master nodes”.
- Ant Colony Optimization is an amazing example where you only make local decisions, but it results in global optimum!

Ant Colony Optimization

- Some biology –
  - Ants secrete **pheromones** which other ants can perceive.
  - The other ants follow the trail of pheromones, and further secrete their own.
  - However, these pheromones slowly erode with time.
  - The beautiful part is – This is sufficient to identify shortest paths – WHY??

Ant Colony Optimization

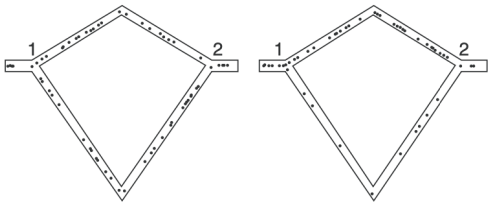

Figure 4.3: *A simplified setup, used in numerical experiments of ant navigation. The ants move from the nest at the left edge, following either of two possible paths to the food source at the right edge, before returning to the nest, again selecting either of two possible paths. The left panel shows a snapshot from the early stages of an experiment, where the probability of an ant choosing either path is roughly equal. Later on, however (right panel), the simulated ants show a clear preference for the shorter path.*

- Can you intuitively justify why the above simple laws are sufficient to obtain the shortest path??

Ant Colony Optimization

- So, how to make this into an algorithm??
  - We assume that the problem is formulated as a *shortest path problem*.
  - Let $G = (V, E)$ denote the graph.
  - We simulate *artificial ants* as follows:
    - Each ant at every node with pick an edge e_{ij} based on the pheromone level τ_{ij} corresponding to that edge.

Ant Colony Optimization

- Encoding the problem as *shortest path problem*
  - Problems like travelling-salesman can easily be converted to a shortest path.
  - In fact most of the problems of rational agents we discussed was also on a graph, albeit where state space might be infinite.
  - How to encode an usual optimization problem as shortest-path problem??

Ant Colony Optimization

- More formally, let
  - $\eta_{ij} = 1/d_{ij}$ for each edge, and τ_{ij} denote the pheromone level on that edge. Also let ϕ_S denote the set of nodes already visited by the ant The edge an "ant" takes is sampled from

$$P(e_{ij}) = \frac{\tau_{ij}^{\alpha}\eta_{ij}^{\beta}}{\sum_{\nu_l \notin \phi_S} \tau_{lj}^{\alpha}\eta_{lj}^{\beta}}$$

  - However, whenever an ant takes a path two things happen - (i) Some pheromone evaporated and (ii) More pheromone gets added. This is captured by the equation

$$\tau_{ij} \leftarrow (1-\rho)\tau_{ij} + \Delta\tau_{ij}$$

where, if D_k is the total length of the path

$$\Delta\tau_{ij} = \begin{cases} 1/D_k & \text{if } e_{ij} \text{ is traversed} \\ 0 & \text{otherwise} \end{cases}$$

Ant Colony Optimization

1. Initialize pheromone levels:

   $\tau_{ij} = \tau_0, \quad \forall i, j \in [1, n].$

2. For each ant k, select a random starting node, and add it to the (initially empty) tabu list L_T. Next, build the tour S. In each step of the tour, select the move from node j to node i with probability $p(e_{ij}|S)$, given by:

   $$p(e_{ij}|S) = \frac{\tau_{ij}^{\alpha}\eta_{ij}^{\beta}}{\sum_{v_l \notin L_T(S)} \tau_{lj}^{\alpha}\eta_{lj}^{\beta}}.$$

   In the final step, return to the node of origin, i.e. the first element in L_T. Finally, compute and store the length D_k of the tour.
3. Update the pheromone levels:
   3.1. For each ant k, determine $\Delta\tau_{ij}^{[k]}$ as:

   $$\Delta\tau_{ij}^{[k]} = \begin{cases} \frac{1}{D_k} & \text{if ant } k \text{ traversed the edge } e_{ij}, \\ 0 & \text{otherwise.} \end{cases}$$

   3.2. Sum the $\Delta\tau_{ij}^{[k]}$ to generate $\Delta\tau_{ij}$:

   $$\Delta\tau_{ij} = \sum_{k=1}^{N} \Delta\tau_{ij}^{[k]}.$$

   3.3. Modify τ_{ij}:

   $$\tau_{ij} \leftarrow (1 - \rho)\tau_{ij} + \Delta\tau_{ij}.$$

4. Repeat steps 2 and 3 until a satisfactory solution has been found.

Algorithm 4.1: *Ant system (AS), applied to the case of TSP. See the main text for a complete description of the algorithm.*

Ant Colony Optimization

- Important Note:
  - Like any other bio-inspired algorithms, there are a lot of variations possible - Can you think of a few?

Ant Colony Optimization

- Some variations:
  - Allow only best ants to deposit pheromones
  - Using the ideas from relaxed problems – We can first solve the problem on the larger graph and slowly remove the nodes/edges!
  - etc.

Particle Swarm Optimization

- Yet another natural phenomenon which when translated into an algorithm gives good results!
  - Forming a swarm/herd of species is one such evolutionary development seen across various species.
  - The question is – Can we use this idea to make search better??

Particle Swarm Optimization

- A model of swarming:
  - Let $S = \{p_i, i = 1, 2, \cdots, N\}$ denote the particles/animal, and let x_i, v_i, a_i denote the position, velocity and acceleration of each particle.
  - Each animal does not have global picture but only sees it's surroundings.
  - Each animal is driven by 3 things:
    - **cohesion:** Wanting to stay near the center of the group

$$\rho_i = \frac{1}{k_i} \sum_{p_j \in V_i} x_j \quad c_i = \frac{1}{T^2}(\rho_i - x_i)$$

    - **alignement:** The speed of p_i should match the group.

$$l_i = \frac{1}{Tk_i} \sum_{p_j \in V_i} v_j$$

    - **separation:** Since the animals also want to avoid collision:

$$s_i = \frac{1}{T^2} \sum_{p_j \in V_i} (x_i - x_j)$$

Particle Swarm Optimization

- Combining the three aspects gives the direction of acceleration:

$$a_i = C_c c_i + C_l l_i + C_s s_i$$

- So, the particle will move as follows:

$$v_i \leftarrow v_i + a_i \Delta t$$
$$x_i \leftarrow x_i + v_i \Delta t$$

Particle Swarm Optimization

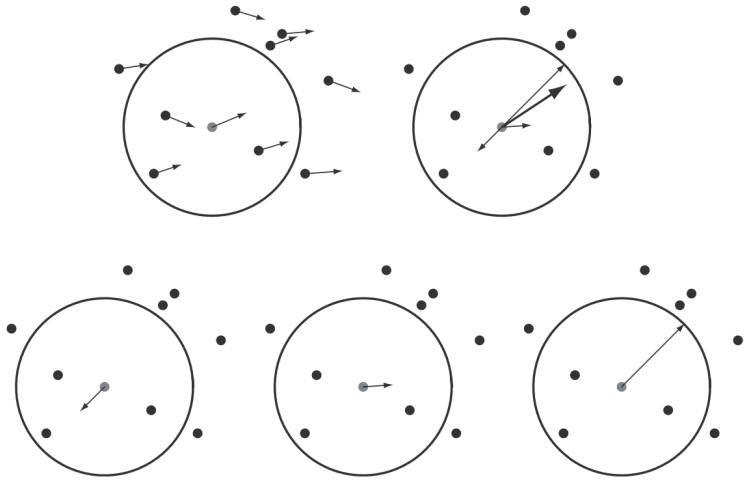

Figure 5.1: *A two-dimensional example of steering vectors: The upper left panel shows the positions and velocities of a swarm of boids, and the upper right panel shows the resulting acceleration vector (thick arrow) for the boid in the centre, as well as its components* **c**, **l** *and* **s**. *The lower panel shows, from left to right, the individual components* **c**, **l** *and* **s**, *respectively. In this example, the adjustable parameters were set to* $T = C_c = C_l = C_s = 1$. *The circles indicate the size of the visibility sphere.*

Particle Swarm Optimization

- So, the question is – How do I apply this for optimizing a function? To adapt the swarm model for optimization –
  - We want the particles to be close to the best possible answers I got till now instead of the center
  - But at the same time not *collide* in the sense that there should be enough random for evolution to play a role.
- This is distilled to the following algorithm:

Particle Swarm Optimization

1. Initialize positions and velocities of the particles p_i:

   1.1 $x_{ij} = x_{\min} + r(x_{\max} - x_{\min})$, $i = 1, \ldots, N$, $j = 1, \ldots, n$

   1.2 $v_{ij} = \frac{\alpha}{\Delta t}\left(-\frac{x_{\max} - x_{\min}}{2} + r(x_{\max} - x_{\min})\right)$, $i = 1, \ldots, N$, $j = 1, \ldots, n$

2. Evaluate each particle in the swarm, i.e. compute $f(\mathbf{x}_i)$, $i = 1, \ldots, N$.
3. Update the best position of each particle, and the global best position. Thus, for all particles p_i, $i = 1, \ldots, N$:

   3.1 if $f(\mathbf{x}_i) < f(\mathbf{x}_i^{\text{pb}})$ then $\mathbf{x}_i^{\text{pb}} \leftarrow \mathbf{x}_i$.

   3.2 if $f(\mathbf{x}_i) < f(\mathbf{x}^{\text{sb}})$ then $\mathbf{x}^{\text{sb}} \leftarrow \mathbf{x}_i$.

4. Update particle velocities and positions:

   4.1 $v_{ij} \leftarrow v_{ij} + c_1 q\left(\frac{x_{ij}^{\text{pb}} - x_{ij}}{\Delta t}\right) + c_2 r\left(\frac{x_j^{\text{sb}} - x_{ij}}{\Delta t}\right)$, $i = 1, \ldots, N$, $j = 1, \ldots, n$

   4.2 Restrict velocities, such that $|v_{ij}| < v_{\max}$.

   4.3 $x_{ij} \leftarrow x_{ij} + v_{ij}\Delta t$, $i = 1, \ldots, N$, $j = 1, \ldots, n$.

5. Return to step 2, unless the termination criterion has been reached.

Algorithm 5.1: *Basic particle swarm optimization algorithm. N denotes the number of particles in the swarm, and n denotes the number of variables in the problem under study. It has been assumed that the goal is to minimize the objective function $f(\mathbf{x})$. See the main text for a complete description of the algorithm.*

Particle Swarm Optimization

- Important Points
  - You want to initialize the positions/velocities so that there is enough diversity in the population.
  - Then you want to move in the direction of global best based on evaluation of all particles.
  - But at the same time, you want to preserve the diversity – modelled by making sure that you are close to your own best solution as well!
  - These two aspects - Staying close to global best and also close to your own best helps balance the exploit/explore dilemma better.

Particle Swarm Optimization

• Application: Learning Neural Networks – PSO has been decently successful in learning neural networks.

- Backpropagation cannot deal with activations which are not differentiable, but PSO can.
- There is a striking similarity between SGD+momentum methods and particle swarm optimization!

Particle Swarm Optimization

- When to use evolutionary algorithms?
  - While it is the case that evolution is a strong force of nature, naively adapting them for optimization is meaningless.
  - Most of the effort goes into understanding/designing the context and improving the efficiency.
  - For instance, if you use genetic algorithm for training neural networks – It would fail, but particle swarm optimization might work better.