Swarm Intelligence

Swarm intelligence (SI) is in the field of artificial intelligence (AI) and is based on the collective behaviour of elements in decentralized and self-organized systems. **Swarm Intelligence (S.I.)** was introduced by **Gerardo Beni and Jing Wang in the year 1989**.

S.I. simply means using the knowledge of collective objects (people, insects, etc.) together and then reaching the optimized solution for a given problem. ***"Swarm" means a group of objects (people, insects***, ***etc.)***.

In other words, let's say we give a problem statement to a single person and tell him or her to go through this problem and then give the solution, then this means that we will consider the solution of that particular person only, but the problem is that the solution given by that person may not be the best solution or maybe, that solution is not good for others. So to avoid that, what we do is we give that problem to a certain amount of people together (swarm) and ask them to reach the best solution possible for that problem, and then computing all the responses together to reach the best solution possible, so here we are using the knowledge of the group as a whole to reach to the best solution or optimized solution for that problem and that solution will be good for all of them individually too, so that is the idea behind swarm intelligence.

Note*: In swarm intelligence, each individual (object) in the group is independent of others, each individual is responsible for their own contribution to solve that problem regardless of what others are doing.*

Ant Colony optimization

Ant Colony Optimization (ACO) are examples of swarm intelligence and metaheuristics. The goal of swarm intelligence is to design intelligent multi-agent systems by taking inspiration from the collective behaviour of social insects such as ants, termites, bees, wasps, and other animal societies such as flocks of birds or schools of fish.

Background:

Ant Colony Optimization technique is purely inspired from the **foraging** behaviour of ant colonies, first introduced by **Marco Dorigo** in the 1990s. Ants are eusocial insects that prefer community survival and sustaining rather than as individual species. They communicate with each other using sound, touch and pheromone. **Pheromones** are organic chemical compounds secreted by the ants that trigger a social response in members of same species. These are chemicals capable of acting like hormones outside the body of the secreting individual, to impact the behaviour of the receiving individuals. Since most ants live on the ground, they use the soil surface to leave pheromone trails that may be followed (smelled) by other ants.

Ants live in community nests and the underlying principle of ACO is to observe the movement of the ants from their nests in order to search for food in the shortest possible path. Initially, ants start to move randomly in search of food around their nests. This randomized search opens multiple routes from the nest to the food source. Now, based on the quality and quantity of the food, ants carry a portion of the food back with necessary pheromone concentration on its return path. Depending on these pheromone trials, the probability of selection of a specific path by the following ants would be a guiding factor to the food source. Evidently, this probability is based on the concentration as well as the rate of evaporation of pheromone. It can also be observed that since the evaporation rate of pheromone is also a deciding factor, the length of each path can easily be accounted for.

In the above figure, for simplicity, only two possible paths have been considered between the food source and the ant nest. The stages can be analysed as follows:

1. **Stage 1:** All ants are in their nest. There is no pheromone content in the environment. (For algorithmic design, residual pheromone amount can be considered without interfering with the probability)

2. **Stage 2:** Ants begin their search with equal (0.5 each) probability along each path. Clearly, the curved path is the longer and hence the time taken by ants to reach food source is greater than the other.

3. **Stage 3:** The ants through the shorter path reaches food source earlier. Now, evidently, they face with a similar selection dilemma, but this time due to pheromone trail along the shorter path already available, probability of selection is higher.

4. **Stage 4:** More ants return via the shorter path and subsequently the pheromone concentrations also increase. Moreover, due to evaporation, the pheromone concentration in the longer path reduces, decreasing the probability of selection of this path in further stages. Therefore, the whole colony gradually uses the shorter path in higher probabilities. So, path optimization is attained.

Algorithmic Design:

Pertaining to the above behaviour of the ants, an algorithmic design can now be developed. For simplicity, a single food source and single ant colony have been considered with just two paths of possible traversal. The whole scenario can be realized through weighted graphs where the ant colony and the food source act as vertices (or nodes); the paths serve as the edges and the pheromone values are the weights associated with the edges. Let the graph be **G = (V, E)** where V, E are the edges and the vertices of the graph. The vertices according to our consideration are **Vs** (Source vertex – ant colony) and **Vd** (Destination vertex – Food source), The two edges are **E1** and **E2** with lengths **L1** and **L2** assigned to each. Now, the associated pheromone values (indicative of their strength) can be assumed to be **R1** and **R2** for vertices E1 and E2 respectively. Thus, for each ant, the starting probability of selection of path (between E1 and E2) can be expressed as follows:

Evidently, if R1>R2, the probability of choosing E1 is higher and vice-versa. Now, while returning through this shortest path say Ei, the pheromone value is updated for the corresponding path. The updation is done based on the length of the paths as well as the evaporation rate of pheromone. So, the update can be step-wise realized as follows:

1. **In accordance to path length** –

$$Ri \leftarrow Ri + \frac{K}{Li}$$

   In the above updation, i = 1, 2 and ‘K’ serves as a parameter of the model. Moreover, the update is dependent on the length of the path. Shorter the path, higher the pheromone added.

2. **In accordance to evaporation rate of pheromone** –

$$Ri \leftarrow (1 - v) * Ri$$

   The parameter ‘v’ belongs to interval (0, 1] that regulates the pheromone evaporation. Further, i = 1, 2.

At each iteration, all ants are placed at source vertex Vs (ant colony). Subsequently, ants move from Vs to Vd (food source) following step 1. Next, all ants conduct their return trip and reinforce their chosen path based on step 2.

Pseudocode:

Procedure Ant Colony Optimization: Initialize necessary parameters and pheromone trials; **while** not termination **do**: Generate ant population; Calculate fitness values associated with each ant; Find best solution through selection methods; Update pheromone trial; **end** while **end** procedure

The pheromone update and the fitness calculations in the above pseudocode can be found through the step-wise implementations mentioned above. Thus, the introduction of the ACO optimization technique has been established. The application of the ACO can be extended to various problems such as the famous **TSP (Travelling Salesman Problem)**.

Swarm optimization in Bees

Artificial Bee Colony Algorithm (ABC) is an optimization algorithm. This algorithm is inspired by the foraging behaviour of honey bees when seeking a quality food source. In the ABC algorithm, there is a population of food positions and the artificial bees modify these food positions over time. The algorithm uses a set of computational agents called honeybees to find the optimal solution. The honeybees in ABC can be categorized into three groups:

employed bees, onlooker bees, and scout bees. The employed bees exploit the food positions, while the onlooker bees are waiting for information from the employed bees about nectar amount of the food positions. The onlooker bees select food positions using the employed bee information and they exploit the selected food positions. Finally, the scout bees find new random food positions. Each solution in the search space consists of a set of optimization parameters which represent a food source position. The number of employed bees is equal to the number of food sources. The quality of a food source is called its “fitness value” and it is associated with its position.

In the algorithm, the employed bees will be responsible for investigating their food sources (using fitness values) and sharing the information to recruit the onlooker bees. The number of the employed bees or the onlooker bees is equal to the number of solutions in the population (*SN*). Each solution (food source) xi(i=1,2,…,SN) is a *D*-dimensional vector. The onlooker bees will make a decision to choose a food source based on this information. A food source with a higher quality will have a larger probability of being selected by onlooker bees. This process of a bee swarm seeking, advertising, and eventually selecting the best known food source is the process used to find the optimal solution. An onlooker bee chooses a food source depending on the probability value associated with that food source **pi** calculated by the following expression:

$$\mathbf{pi=fiti\sum q=SNfitq,}$$

where fiti is the fitness value of the solution *i* evaluated by its employed bee, which is proportional to the nectar amount of the food source in position *i*, and *SN* is the number of food sources which is equal to the number of employed bees (BN). In this way, the employed bees exchange their information with the onlookers. In order to produce a candidate food position from the old one, the ABC uses the following expression:

$$\mathbf{xij_{*}=xij+\phi ij(xij-xlj),}$$

where $*x_{ij}*$ is the new feasible food source, which is selected by comparing the previous food source x_{ij} and the randomly selected food source, $l \in \{1,2,\ldots,SN\}$ and $j \in \{1,2,\ldots,D\}$ are randomly chosen indices, and ϕ_{ij} is a random number between $[-1,1]$ which is used to adjust the old food source to become the new food source in the next iteration.

Cuckoo Search Algorithm

Cuckoo Search (CS) is a new heuristic algorithm inspired from the obligate brood parasitic behaviour of some cuckoo species as they lay their eggs in the nests of host birds.

Some cuckoos have a specialty of imitating colours and patterns of eggs of a few chosen host species. This reduces the probability of eggs being abandoned. If host bird discovers foreign eggs, they either abandon the eggs or throw them away.

Parasitic cuckoos choose a nest where the host bird just lays its eggs. Eggs of cuckoo hatch earlier than their host eggs and when it hatches, it propels the host eggs out of the nests. Hence cuckoo chicks get a good share of food and sometimes they even imitate the voice of host chicks to get more food (Payne, 2005). Mostly cuckoos search food by a simple random walk, Using Lévy flights instead of simple random walks improve the search capabilities. Lévy flight is a random walk-in step-lengths following a heavy-tailed probability distribution (Yang & Deb, 2009). Each cuckoo acts as a potential solution to the problem under consideration. The main aim is to generate a new and potentially better solution (cuckoo) to be replaced with a not so good solution. Each nest has one egg but as the problem complexity increases, multiple eggs can be used to represent a set of solutions.

There are three basic idealized rules of CS. First rule says that each cuckoo lays one egg and dumps it in a random nest. The second rule is that the nest with the highest fitness will carry over to next generations whereas the final rule defines that the number of available host nests is kept fixed, and the egg laid by cuckoo is discovered by host bird with a probability p [0, 1]. And depending on p, the host bird either throws the egg away or abandons the nest. It is assumed, that only a fraction p of nests is replaced by new nests. Based on the three rules, the cuckoo search has been implemented. To generate a new solution xt+1 i for ith cuckoo, Lévy flight is performed. This step is called global random walk and is given by xt+1 i = xt i + α Levy(λ) xbest − xt i (1) The local random walk is given by: xt+1 i = xt i + α H(p −) xt j − xt k (2) where xt i is the previous solution, α > 0 is the step size related to problem scales and is entry wise multiplication. Here xt j and xt k are randomly selected solutions and xbest is the current best solution. In present work, the random step length via Lévy flight is considered due to more efficiency of Lévy flights in exploring the search space and is drawn from a Lévy distribution having infinite variance and mean. Levy ~ λ(λ) sin (πλ/2) π 1 s1+λ, (s s0 > 0) (3) As some fractions of new solutions are generated by Lévy flights, so the local search speeds up. Here some of the solutions should be generated by far field randomization which will keep the system away from getting trapped in local optimum, (λ) is the gamma function, p is the switch probability, ε is a random number and (1 < λ ≤ 3). The step length in cuckoo search is heavy-tailed and any large step is possible due to large scale randomization

FireFly Algorithm

Firefly algorithm is classified as swarm intelligent, metaheuristic and nature-inspired, and it is developed by Yang in 2008 by animating the characteristic behaviours of fireflies. In fact, the population of fireflies show characteristic luminary flashing activities to function as attracting the partners, communication, and risk warning for predators. As inspiring from those activities, Yang formulated this method under the assumptions of all fireflies are unisexual such that all fireflies has attracting potential for each other and the attractiveness is directly proportionate to the brightness level of individuals.
 Hence, the brighter fireflies attract to the less brighter ones to move toward to them, besides that in the case of no fireflies brighter than a certain firefly then it moves randomly.
 (FA) was first developed by Yang in 2007 (Yang, 2008, 2009) which was based on the flashing patterns and behaviour of fireflies. In essence, FA uses the following three idealized rules:

- Fireflies are unisexual so that one firefly will be attracted to other fireflies regardless of their sex.

- The attractiveness is proportional to the brightness and they both decrease as their distance increases. Thus, for any two flashing fireflies, the less brighter one will move toward the more brighter one. If there is no brighter one than a particular firefly, it will move randomly.

- The brightness of a firefly is determined by the landscape of the objective function.

The movement of a firefly *i* is attracted to another, more attractive (brighter) firefly *j* is determined by

$$\boldsymbol{x_i^{t+1} = x_i^t + \beta_0 \mathrm{e}^{-\gamma r_{ij}^2}\left(x_j^t - x_i^t\right) + \alpha\varepsilon_i^t}$$

where β0 is the attractiveness at the distance r=0, and the second term is due to the attraction. The third term is randomization with α being the randomization parameter, and $\boldsymbol{\alpha\varepsilon_i^t}$I s a vector of random numbers drawn from a Gaussian distribution or uniform distribution at time *t*. If β0=0, it becomes a simple random walk. Furthermore, the randomization $\boldsymbol{\alpha\varepsilon_i^t}$ can easily be extended to other distributions such as Lévy flights.
The Lévy flight essentially provides a random walk whose random step length is drawn from a Lévy distribution

$$\boldsymbol{L(s,\lambda) = s^{-(1+\lambda)}, (0 < \lambda \leq 2)}$$

which has an infinite variance with an infinite mean. Here the steps essentially form a random walk process with a power-law step-length distribution with a heavy tail. Some of the new solutions should be generated by Lévy walk around the best solution obtained so far; this will often speed up the local search (Pavlyukevich, 2007).
A demo version of firefly algorithm implementation, without Lévy flights, can be found at Mathworks file exchange web site.[1] Firefly algorithm has attracted much attention (Apostolopoulos and Vlachos, 2011; Gandomi et al., 2011b; Sayadi et al., 2010). A discrete version of FA can efficiently solve NP-hard scheduling problems (Sayadi et al., 2010), while a detailed analysis has demonstrated the efficiency of FA over a wide range of test problems, including multiobjective load dispatch problems (Apostolopoulos and Vlachos, 2011). A chaos-enhanced firefly algorithm with a basic method for automatic parameter tuning is also developed (Yang, 2011b).

Crow Search Algorithm

Crows are said to be the most intelligent of all birds. They have shown self-awareness in mirror tests and the capacity to make tools. Crows can recognise one other's faces and warn one another when a hostile one approaches. Furthermore, they can utilise tools, communicate in sophisticated ways, and recall the location of their meal for many months afterwards.

Based on the Crow a nature-based algorithm is developed which is named as Crow Search Algorithm. This algorithm shares these four properties with the crow's behaviour.

- Crows live in groups.
- Crows memorize the location of their food stashes.
- Crows follow each other to steal
- Crows guard their stores against theft

Why is the Crow Search Algorithm used?

The property of independently gathering responses from distinct things and then computing all replies as a whole to find the optimal solution to the given challenge. As a result of this method, a better optimal solution for a specific issue is found, which is why CSA is employed. Simply said, the notion of collective behaviour of decentralized, self-organization is used to solve a particular issue. For example, Forecasting an optimal policy which is best suited to launch a new product in the market.

How does it work?

The goal of this meta-heuristic is for a given crow to be able to follow another crow in order to find its hidden food location. The crow's position should be updated gradually throughout this procedure. Furthermore, when food is stolen, the crow must alter its location.

It is considered that there is a d-dimensional environment with many crows. A vector specifies the number of crows, which is the flock size, as well as the location of the crow at each iteration in the search space. Each crow has a memory that stores the location of its hiding spot. The location of the crow's hiding spot is indicated at each iteration. This is the finest position that a crow has ever had. Indeed, each crow remembers the location of its finest encounter. Crows travel across the area looking for better food sources (hiding places).

Assume that in the next iteration, another crow wants to go to its hiding location, which the previous crow designated. In this iteration, the first crow decides to follow another crow to another crow's hiding location. In this situation, two outcomes are possible.

1. The other crow does not know that the first crow is following. As a result, the first crow will approach the hiding place of the other crow. In this case, the new position of the first crow is obtained with the help of a random number with a uniform distribution between 0 and 1 and the flight length of the first crow at iteration.

2. The other crow knows that the first crow is following. As a result, in order to protect its cache from being stolen, the other crow will fool the first crow by going to another position in the search space.

Whale Optimization Algorithm

Whale optimization algorithm (WOA): A nature inspired meta-heuristic optimization algorithm which mimics the hunting behaviour of humpback whales. The algorithm is inspired by the bubble-net hunting strategy

Foraging behavior of Humpback whales is called bubble-net feeding method. Humpback whales prefer to hunt school of krill or small fishes close to the surface. It has been observed that this foraging is done by creating distinctive bubbles along a circle or '9'-shaped path as shown in Fig. 1.

Two maneuvers associated with bubble net feeding are 'upward-spirals' and 'double-loops'.

- In '**upward-spirals**' maneuver humpback whales dive around 12 m down and then start to create bubble in a spiral shape around the prey and swim up toward the surface.
- '**double-loops**' maneuver includes three different stages: coral loop, lobtail, and capture loop

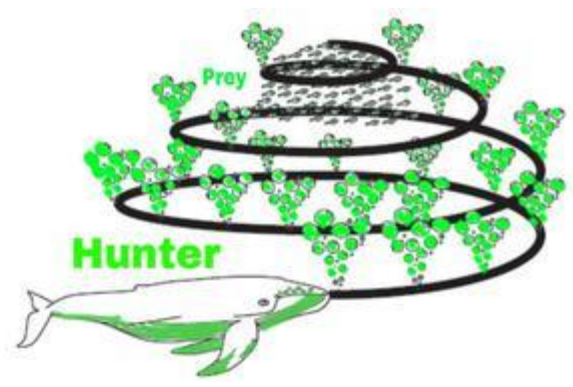


Figure 1: Bubble-net feeding behaviour of humpback whales.

The WOA algorithm imitates the social behavior and hunting method of humpback whales in oceans. So, let’s go over the fundamentals of whale hunting:

1. Exploration phase: search for the prey
2. Encircling the prey
3. Exploitation phase: attacking the prey using a bubble-net method

Let’s take a look at the mathematical model behind each phase of the whale hunting process.

. Exploration Phase: Searching Model

The search agent (humpback whale) looks for the best solution (the prey) randomly based on the position of each agent. We update the position of a search agent during this phase by using a randomly selected search agent rather than the best search agent. Following that, if $|A| > 1$

, as defined in Equation 3, then force the search agent to move away from a reference whale. Here’s the mathematical model behind this phase:

$$\vec{D} = |\vec{C} * \vec{X_{rand}} - \vec{X}|$$
$$\vec{X}(t+1) = \vec{X_{rand}} - \vec{A} * \vec{D}$$

Where   is a random position vector selected from the current population, and [,   ] are coefficient vectors. Besides, here are the equations of [and   ] that can be used to find the best search agents:

$$(\vec{A} = 2 * \vec{a} * \vec{r} - \vec{a}$$

$$\vec{C} = 2 * \vec{r}$$

Where $\vec{X_{rand}}$ is a random vector in range [0, 1] and   decreases linearly from 2 to 0 during the iterations.

4.2. Encircling Model

Humpback whales encircle the prey during hunting. Then, they consider the current best candidate solution as the best solution and near the optimal one. In short, here’s the model of encircling behavior that is used to update the position of the other whales towards the best search agent:

$$D = |\vec{C} * \vec{X}'(t) - \vec{X}(t)| \quad (5)$$

$$\vec{X}(t+1) = \vec{X}'(t) - \vec{A} * \vec{D} \quad (6)$$

Where t is the current iteration,   is the position of the best solution,   refers to the position vector of a solution, and [,   ] are coefficient vectors as shown in Equations 3 and

Exploitation Phase: Bubble-Net Attacking Model

This phase focuses on using a bubble-net to attack the prey. This bubble-net strategy combines two approaches. Let's look at the mathematical model of each approach in order to better understand this step:

- **Shrinking encircling mechanism**: The value of in this mechanism is a random value within the [-a, a] interval, and the value of decreases from 2 to 0 over the course of iterations, as shown in Equation 3. Let's now define random values for in [-1, 1]. We can also define the new position of a search agent anywhere between the current best agent's position and the agent's original position. So, let's look at the graph that displays the possible positions from to :

- **Spiral updating position mechanism**: This method starts with the calculation of the distance between the whale and the prey . Here's the spiral equation behind the helix-shaped movement of humpback whales to define the position between whale and prey:

Grasshopper Algorithm

The GOA algorithm mimics the social behaviour and hunting method of grasshoppers in nature. It's a population-based algorithm, with each grasshopper representing a solution in the population. **So, how do we determine the position of each solution in the swarm?**

It's simple, we only need to calculate three forces: the social interaction between the solution and the other grasshoppers, the wind advection, and the gravity force on the solution. Let's now have a look at the mathematical model used to calculate the position X_i of each solution:

$$X_i = S_i + G_i + A_i$$

where S_i represents the social interaction between the solution and the other grasshoppers, G_i represents the gravity force on the solution, and A_i represents the wind advection. The equation below represents the position of each solution after adding random behavior to it:

$$X_i = r_1 S_i + r_2 G_i + r_3 A_i$$

where r_1, r_2, and r_3 are random numbers in [0, 1]. Let's now have a look at the model of each force used in Equation 1.

Force of Social Interaction

Let's firstly start with the force of social interaction S_i. The equation below represents the social interaction between the solution and the other grasshoppers:

$$S_i = \sum_{j=1}^{N} s(d_{ij})\hat{d_{ij}} \text{ , where } i \neq j$$

$$s = fe^{\frac{-r}{l}} - e^{-r}$$

where $d_{ij} = |x_j - x_i|$ represents the distance between i-th grasshopper and the j-th grasshopper, $\hat{d_{ij}} = \frac{|x_j - x_i|}{d_{ij}}$ represents the unit vector. In addition, s represents the strength of two social forces (repulsion and attraction between grasshoppers), where l is the attractive length scale and f is the intensity of attraction. In fact, these cofficients l and f have a great impact in the social behavior of grasshoppers.

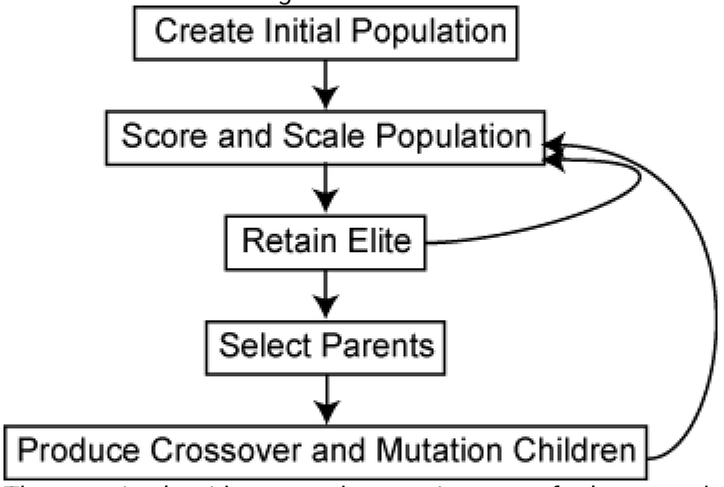
This work proposed an optimisation algorithm called the Grasshopper Optimisation Algorithm. The proposed algorithm mathematically modelled and mimicked the swarming behaviour of grasshoppers in nature for solving optimisation problems. A mathematical model was proposed to simulate repulsion and attraction forces between the grasshoppers. Repulsion forces allow grasshoppers to explore the search space, whereas attraction forces encouraged them to exploit promising regions. To balance between

What Is the Genetic Algorithm?

Genetic Algorithms(GAs) are adaptive heuristic search algorithms that belong to the larger part of evolutionary algorithms. Genetic algorithms are based on the ideas of natural selection and genetics. These are intelligent exploitation of random search provided with historical data to direct the search into the region of better performance in solution space. They are commonly used to generate high-quality solutions for optimization problems and search problems.

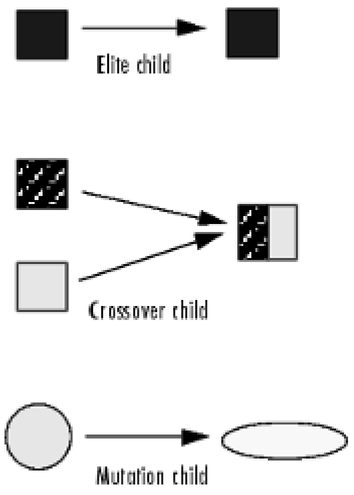
Genetic algorithms simulate the process of natural selection which means those species who can adapt to changes in their environment are able to survive and reproduce and go to next generation. In simple words, they simulate "survival of the fittest" among individual of consecutive generation for solving a problem. **Each generation consist of a population of individuals** and each individual represents a point in search space and possible solution. Each individual is represented as a string of character/integer/float/bits. This string is analogous to the Chromosome.

The genetic algorithm can address problems of *mixed integer programming*, where some components are restricted to be integer-valued.


Create Initial Population
Score and Scale Population
Retain Elite
Select Parents
Produce Crossover and Mutation Children

The genetic algorithm uses three main types of rules at each step to create the next generation from the current population:

- *Selection rules* select the individuals, called *parents*, that contribute to the population at the next generation. The selection is generally stochastic, and can depend on the individuals' scores.
- *Crossover rules* combine two parents to form children for the next generation.
- *Mutation rules* apply random changes to individual parents to form children.



The genetic algorithm differs from a classical, derivative-based, optimization algorithm in two main ways, as summarized in the following table:

Classical Algorithm	Genetic Algorithm
Generates a single point at each iteration. The sequence of points approaches an optimal solution.	Generates a population of points at each iteration. The best point in the population approaches an optimal solution.
Selects the next point in the sequence by a deterministic computation.	Selects the next population by computation which uses random number generators.
Typically converges quickly to a local solution.	Typically takes many function evaluations to converge. May or may not converge to a local or global minimum.

Operators of Genetic Algorithms

Once the initial generation is created, the algorithm evolves the generation using following operators –

1) Selection Operator: The idea is to give preference to the individuals with good fitness scores and allow them to pass their genes to successive generations.

2) Crossover Operator: This represents mating between individuals. Two individuals are selected using selection operator and crossover sites are chosen randomly. Then the genes at these crossover sites are exchanged thus creating a completely new individual (offspring).

3) Mutation Operator: The key idea is to insert random genes in offspring to maintain the diversity in the population to avoid premature convergence. For example –

The whole algorithm can be summarized as –

1) Randomly initialize populations p

2) Determine fitness of population

3) Until convergence repeat:

a) Select parents from population b) Crossover and generate new population c) Perform mutation on new population d) Calculate fitness for new population

Example problem and solution using Genetic Algorithms

Given a target string, the goal is to produce target string starting from a random string of the same length. In the following implementation, following analogies are made –

- Characters A-Z, a-z, 0-9, and other special symbols are considered as genes
- A string generated by these characters is considered as chromosome/solution/Individual

Fitness score is the number of characters which differ from characters in target string at a particular index. So individual having lower fitness value is given more preference.

Why use Genetic Algorithms

- They are Robust
- Provide optimisation over large space state.
- Unlike traditional AI, they do not break on slight change in input or presence of noise

Applications of Genetic Algorithm

1. Acoustics- GA is used to distinguish between sonar reflections and different types of objects. Apart from that it is also used to design Active noise control systems which cancel out undesired sound by producing sound waves that destructively interfere the unwanted noise.

2. Aerospace Engineering- To design wing shape Super Sonic aircraft minimizing aerodynamic drag at supersonic cruising speeds, minimizing drag at subsonic speeds, and minimizing aerodynamic load.

3. Financial Markets – to predict the future performance of publicly traded stocks.

4. Geophysics-locate earthquake hypocentres based on seismological data.

5. Materials Engineering-To design electrically conductive carbon-based polymers known as polyanilines. To design exposure pattern for an electron lithography beam.

6. Routing and Scheduling- To find optimal routing paths in telecommunication networks which are used to relay data from sender to recipients .

7. Systems Engineering- To perform multi–objective task of designing wind turbines used to generate electric power.

8. Data mining- GA can also be used in data mining and pattern recognition. Given a large set of data, GA can perform a search for the optimum set of data which satisfies the conditions. Initial population in this case may be single objective conditions and at the end of the GA, we get a combined complex condition which when applied on the large data set can lead to finding the required set of data

Working of Genetic algorithm

Initialization

The evolutionary process begins with initialization, wherein an initial population of candidate solutions is generated. There are many different methods of initializing populations, but with Genetic Algorithms the most popular method of initialization is simply to create a population of randomly initialized binary strings. Once the initial population has been created, the evolutionary generational cycle begins.

Selection

At each generational step, a pool of parents is chosen from the parent population based on the fitness values of each individual using a selection mechanism, such that the fittest individuals will have a greater probability of passing on genetic material to subsequent generations. This selected population (known as the “parent” population) then forms the basis for all subsequent optimizations during the generational step.

Variation

Once the parent population is fully populated via the selection process, a child population is created which will form the basis of the next generation. This child population is generated by variation operators, which are performed on individuals from the parent population. The most prominent such methods are crossover and mutation.

Crossover

Crossover takes pairs of parents from the parent population (usually random selection with replacement, such that the same parent can be selected multiple times to produce children). These parents then have some of their genetic information swapped between them. Crossover of two parent chromosomes is usually done in a single-point manner, where a single crossover point on both chromosomes is randomly selected and then all subsequent genetic information is swapped between individuals.

Crossover is typically performed on randomly selected pairs of parents from the parent population until the new “child” population reaches the same size as the original parent population.

Mutation

While crossover simply swaps pre-existing information between pairs of candidate solutions, mutation in Evolutionary Algorithms is typically the standard method of introducing “new” genetic material into the population. Once the child population has been created via crossover, mutation in canonical Genetic Algorithms then occurs on all children in a “bit-flip” fashion by randomly changing codons on the chromosome between 0 and 1.

Evaluation

Once the child population has been created, all children then need to be evaluated in order to assign a fitness value by which they can be judged against their peers. This fitness is used to sort/rank a population and to impose probabilities for both selection and replacement phases of the search process, such that the fittest members of the population have a higher probability of passing on genetic material.

Replacement

The final act of the generational step is to replace the old “N” population with the new “N+1” child population. While many methods exist, the most typical replacement method is

Generational Replacement, wherein the entire previous generation is replaced with the newly generated child population.

Termination

A Genetic Algorithm will typically terminate after a predefined number of generations, or if some stopping criterion has been met (e.g. fitness is above some threshold, error rate is below some threshold, etc). The fittest solution in the population is then returned as the overall best solution.

Dimensionality Reduction

In machine learning classification problems, there are often too many factors based on which the final classification is done. These factors are basically variables called features. The higher the number of features, the harder it gets to visualize the training set and then work on it. Sometimes, most of these features are correlated, and hence redundant. This is where dimensionality reduction algorithms come into play. Dimensionality reduction is the process of reducing the number of random variables under consideration, by obtaining a set of principal variables. It can be divided into feature selection and feature extraction.

Why is Dimensionality Reduction important in Machine Learning and Predictive Modelling?

An intuitive example of dimensionality reduction can be discussed through a simple e-mail classification problem, where we need to classify whether the e-mail is spam or not. This can involve a large number of features, such as whether or not the e-mail has a generic title, the content of the e-mail, whether the e-mail uses a template, etc. However, some of these features may overlap. In another condition, a classification problem that relies on both humidity and rainfall can be collapsed into just one underlying feature, since both of the aforementioned are correlated to a high degree. Hence, we can reduce the number of features in such problems. A 3-D classification problem can be hard to visualize, whereas a 2-D one can be mapped to a simple 2 dimensional space, and a 1-D problem to a simple line. The below figure illustrates this concept, where a 3-D feature space is split into two 2-D feature spaces, and later, if found to be correlated, the number of features can be reduced even further.

Components of Dimensionality Reduction

There are two components of dimensionality reduction:

- **Feature selection:** In this, we try to find a subset of the original set of variables, or features, to get a smaller subset which can be used to model the problem. It usually involves three ways:
  1. Filter
  2. Wrapper
  3. Embedded
- **Feature extraction:** This reduces the data in a high dimensional space to a lower dimension space, i.e. a space with lesser no. of dimensions.

Methods of Dimensionality Reduction

The various methods used for dimensionality reduction include:

- Principal Component Analysis (PCA)
- Linear Discriminant Analysis (LDA)
- Generalized Discriminant Analysis (GDA)

Dimensionality reduction may be both linear or non-linear, depending upon the method used. The prime linear method, called Principal Component Analysis, or PCA, is discussed below.

Principal Component Analysis

This method was introduced by Karl Pearson. It works on a condition that while the data in a higher dimensional space is mapped to data in a lower dimension space, the variance of the data in the lower dimensional space should be maximum. It involves the following steps:

- Construct the covariance matrix of the data.
- Compute the eigenvectors of this matrix.
- Eigenvectors corresponding to the largest eigenvalues are used to reconstruct a large fraction of variance of the original data.

Hence, we are left with a lesser number of eigenvectors, and there might have been some data loss in the process. But, the most important variances should be retained by the remaining eigenvectors.

Advantages of Dimensionality Reduction

- It helps in data compression, and hence reduced storage space.
- It reduces computation time.
- It also helps remove redundant features, if any.

Disadvantages of Dimensionality Reduction

- It may lead to some amount of data loss.
- PCA tends to find linear correlations between variables, which is sometimes undesirable.
- PCA fails in cases where mean and covariance are not enough to define datasets.
- We may not know how many principal components to keep- in practice, some thumb rules are applied.

What Is Principal Component Analysis?

Principal component analysis, or PCA, is a dimensionality-reduction method that is often used to reduce the dimensionality of large data sets, by transforming a large set of variables into a smaller one that still contains most of the information in the large set.

Reducing the number of variables of a data set naturally comes at the expense of accuracy, but the trick in dimensionality reduction is to trade a little accuracy for simplicity. Because smaller data sets are easier to explore and visualize and make analyzing data much easier and faster for machine learning algorithms without extraneous variables to process.

So, to sum up, the idea of PCA is simple — **reduce the number of variables of a data set, while preserving as much information as possible.**

Step-by-Step Explanation of PCA

STEP 1: STANDARDIZATION

The aim of this step is to standardize the range of the continuous initial variables so that each one of them contributes equally to the analysis.

More specifically, the reason why it is critical to perform standardization prior to PCA, is that the latter is quite sensitive regarding the variances of the initial variables. That is, if there are large differences between the ranges of initial variables, those variables with larger ranges will dominate over those with small ranges (for example, a variable that ranges between 0 and 100 will dominate over a variable that ranges between 0 and 1), which will lead to biased results. So, transforming the data to comparable scales can prevent this problem.

Mathematically, this can be done by subtracting the mean and dividing by the standard deviation for each value of each variable.

$$z = \frac{value - mean}{standard\ deviation}$$

Once the standardization is done, all the variables will be transformed to the same scale.

STEP 2: COVARIANCE MATRIX COMPUTATION

The aim of this step is to understand how the variables of the input data set are varying from the mean with respect to each other, or in other words, to see if there is any relationship between them. Because sometimes, variables are highly correlated in such a way that they contain redundant information. So, in order to identify these correlations, we compute the covariance matrix.

The covariance matrix is a $p \times p$ symmetric matrix (where p is the number of dimensions) that has as entries the covariances associated with all possible pairs of the initial variables. For example, for a 3-dimensional data set with 3 variables *x*, *y*, and *z*, the covariance matrix is a 3×3 matrix of this from:

$$\begin{bmatrix} Cov(x,x) & Cov(x,y) & Cov(x,z) \\ Cov(y,x) & Cov(y,y) & Cov(y,z) \\ Cov(z,x) & Cov(z,y) & Cov(z,z) \end{bmatrix}$$

Covariance Matrix for 3-Dimensional Data

Since the covariance of a variable with itself is its variance (Cov(a,a)=Var(a)), in the main diagonal (Top left to bottom right) we actually have the variances of each initial variable. And since the covariance is commutative (Cov(a,b)=Cov(b,a)), the entries of the covariance matrix are symmetric with respect to the main diagonal, which means that the upper and the lower triangular portions are equal.

What do the covariances that we have as entries of the matrix tell us about the correlations between the variables?

It's actually the sign of the covariance that matters:

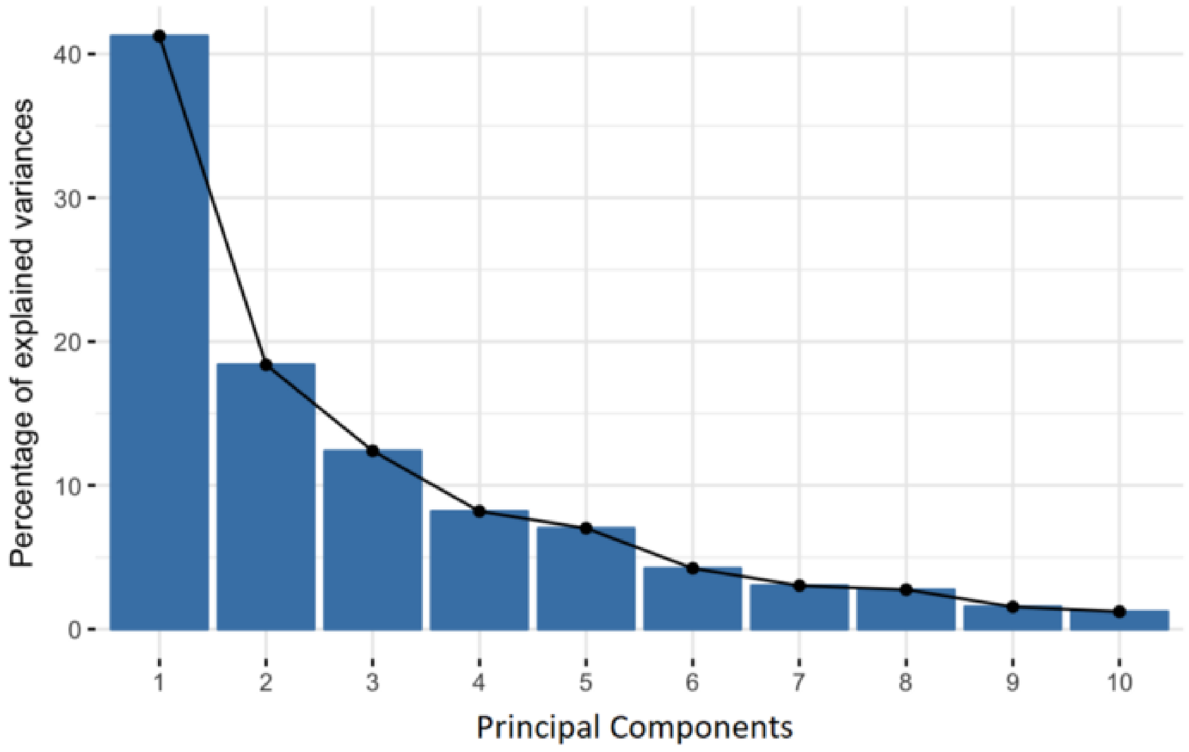
- If positive then: the two variables increase or decrease together (correlated)
- If negative then: one increases when the other decreases (Inversely correlated)

Now that we know that the covariance matrix is not more than a table that summarizes the correlations between all the possible pairs of variables, let's move to the next step.

STEP 3: COMPUTE THE EIGENVECTORS AND EIGENVALUES OF THE COVARIANCE MATRIX TO IDENTIFY THE PRINCIPAL COMPONENTS

Eigenvectors and eigenvalues are the linear algebra concepts that we need to compute from the covariance matrix in order to determine the ***principal components*** of the data. Before getting to the explanation of these concepts, let's first understand what do we mean by principal components.

Principal components are new variables that are constructed as linear combinations or mixtures of the initial variables. These combinations are done in such a way that the new variables (i.e., principal components) are uncorrelated and most of the information within the initial variables is squeezed or compressed into the first components. So, the idea is 10-dimensional data gives you 10 principal components, but PCA tries to put maximum possible information in the first component, then maximum remaining information in the second and so on, until having something like shown in the scree plot below.



Percentage of Variance (Information) for each by PC

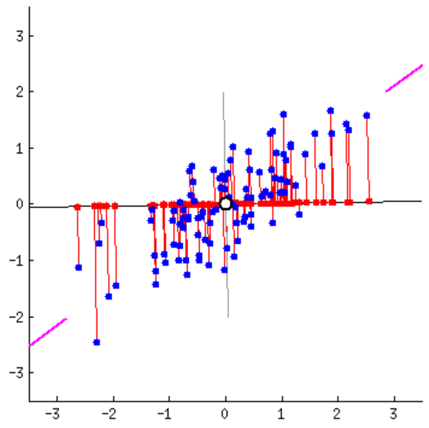
Organizing information in principal components this way, will allow you to reduce dimensionality without losing much information, and this by discarding the components with low information and considering the remaining components as your new variables.

An important thing to realize here is that the principal components are less interpretable and don't have any real meaning since they are constructed as linear combinations of the initial variables.

Geometrically speaking, principal components represent the directions of the data that explain a **maximal amount of variance**, that is to say, the lines that capture most information of the data. The relationship between variance and information here, is that, the larger the variance carried by a line, the larger the dispersion of the data points along it, and the larger the dispersion along a line, the more information it has. To put all this simply, just think of principal components as new axes that provide the best angle to see and evaluate the data, so that the differences between the observations are better visible.

How PCA Constructs the Principal Components

As there are as many principal components as there are variables in the data, principal components are constructed in such a manner that the first principal component accounts for the **largest possible variance** in the data set. For example, let's assume that the scatter plot of our data set is as shown below, can we guess the first principal component ? Yes, it's approximately the line that matches the purple marks because it goes through the origin and it's the line in which the projection of the points (red dots) is the most spread out. Or mathematically speaking, it's the line that maximizes the variance (the average of the squared distances from the projected points (red dots) to the origin).



The second principal component is calculated in the same way, with the condition that it is uncorrelated with (i.e., perpendicular to) the first principal component and that it accounts for the next highest variance.

This continues until a total of p principal components have been calculated, equal to the original number of variables.

Now that we understand what we mean by principal components, let's go back to eigenvectors and eigenvalues. What you first need to know about them is that they always come in pairs, so that every eigenvector has an eigenvalue. And their number is equal to the number of dimensions of the data. For example, for a 3-dimensional data set, there are 3 variables, therefore there are 3 eigenvectors with 3 corresponding eigenvalues.

Without further ado, it is eigenvectors and eigenvalues who are behind all the magic explained above, because the eigenvectors of the Covariance matrix are actually *the directions of the axes where there is the most variance*(most information) and that we call Principal Components. And eigenvalues are simply the coefficients attached to eigenvectors, which give the *amount of variance carried in each Principal Component*.

By ranking your eigenvectors in order of their eigenvalues, highest to lowest, you get the principal components in order of significance.

Example:

Let's suppose that our data set is 2-dimensional with 2 variables ***x,y*** and that the eigenvectors and eigenvalues of the covariance matrix are as follows:

$$v1 = \begin{bmatrix} 0.6778736 \\ 0.7351785 \end{bmatrix} \qquad \lambda_1 = 1.284028$$

$$v2 = \begin{bmatrix} -0.7351785 \\ 0.6778736 \end{bmatrix} \qquad \lambda_2 = 0.04908323$$

If we rank the eigenvalues in descending order, we get λ1>λ2, which means that the eigenvector that corresponds to the first principal component (PC1) is *v1* and the one that corresponds to the second component (PC2) is*v2*.

After having the principal components, to compute the percentage of variance (information) accounted for by each component, we divide the eigenvalue of each component by the sum of eigenvalues. If we apply this on the example above, we find that PC1 and PC2 carry respectively 96% and 4% of the variance of the data.

STEP 4: FEATURE VECTOR

As we saw in the previous step, computing the eigenvectors and ordering them by their eigenvalues in descending order, allow us to find the principal components in order of significance. In this step, what we do is, to choose whether to keep all these components or discard those of lesser significance (of low eigenvalues), and form with the remaining ones a matrix of vectors that we call *Feature vector*.

So, the feature vector is simply a matrix that has as columns the eigenvectors of the components that we decide to keep. This makes it the first step towards dimensionality reduction, because if we choose to keep only ***p*** eigenvectors (components) out of ***n***, the final data set will have only ***p*** dimensions.

Example:

Continuing with the example from the previous step, we can either form a feature vector with both of the eigenvectors *v*1 and *v*2:

$$\begin{bmatrix} 0.6778736 & -0.7351785 \\ 0.7351785 & 0.6778736 \end{bmatrix}$$

Or discard the eigenvector *v*2, which is the one of lesser significance, and form a feature vector with *v*1 only:

$$\begin{bmatrix} 0.6778736 \\ 0.7351785 \end{bmatrix}$$

Discarding the eigenvector *v2* will reduce dimensionality by 1, and will consequently cause a loss of information in the final data set. But given that *v*2 was carrying only 4% of the information, the loss will be therefore not important and we will still have 96% of the information that is carried by *v*1.

So, as we saw in the example, it’s up to you to choose whether to keep all the components or discard the ones of lesser significance, depending on what you are looking for. Because if you just want to describe your data in terms of new variables (principal components) that are uncorrelated without seeking to reduce dimensionality, leaving out lesser significant components is not needed.

LAST STEP: RECAST THE DATA ALONG THE PRINCIPAL COMPONENTS AXES

In the previous steps, apart from standardization, you do not make any changes on the data, you just select the principal components and form the feature vector, but the input data set remains always in terms of the original axes (i.e, in terms of the initial variables).

In this step, which is the last one, the aim is to use the feature vector formed using the eigenvectors of the covariance matrix, to reorient the data from the original axes to the ones represented by the principal components (hence the name Principal Components Analysis). This can be done by multiplying the transpose of the original data set by the transpose of the feature vector.

$$FinalDataSet = FeatureVector^T * StandardizedOriginalDataSet^T$$

Linear Discriminant Analysis

Linear Discriminant Analysis **or** Normal Discriminant Analysis **or** Discriminant Function Analysis is a dimensionality reduction technique that is commonly used for supervised classification problems. It is used for modelling differences in groups i.e. separating two or more classes. It is used to project the features in higher dimension space into a lower dimension space. For example, we have two classes, and we need to separate them efficiently. Classes can have multiple features. Using only a single feature to classify them may result in some overlapping as shown in the below figure. So, we will keep on increasing the number of features for proper classification.

Example: Suppose we have two sets of data points belonging to two different classes that we want to classify. As shown in the given 2D graph, when the data points are plotted on the 2D plane, there's no straight line that can separate the two classes of the data points completely. Hence, in this case, LDA (Linear Discriminant Analysis) is used which reduces the 2D graph into a 1D graph in order to maximize the separability between the two classes.

Here, Linear Discriminant Analysis uses both the axes (X and Y) to create a new axis and projects data onto a new axis in a way to maximize the separation of the two categories and hence, reducing the 2D graph into a 1D graph.

Two criteria are used by LDA to create a new axis:

1. Maximize the distance between means of the two classes.
2. Minimize the variation within each class

KERNEL PCA:

Kernel PCA is an extension of PCA that allows for the separability of nonlinear data by making use of kernels. The basic idea behind it is to project the linearly inseparable data onto a higher dimensional space where it becomes linearly separable.

PCA is a linear method. That is it can only be applied to datasets which are linearly separable. It does an excellent job for datasets, which are linearly separable. But, if we use it to non-linear datasets, we might get a result which may not be the optimal dimensionality reduction. Kernel PCA uses a kernel function to project dataset into a higher dimensional feature space, where it is linearly separable. It is similar to the idea of Support Vector Machines.

There are various kernel methods like linear, polynomial, and gaussian.

Fuzzification

Fuzzification: It is the method of transforming a crisp quantity(set) into a fuzzy quantity(set). This can be achieved by identifying the various known crisp and deterministic quantities as completely nondeterministic and quite uncertain in nature. This uncertainty may have emerged because of vagueness and imprecision which then lead the variables to be represented by a membership function as they can be fuzzy in nature. For example, when I say the temperature is 45° Celsius the viewer converts the crisp input value into a linguistic variable like favourable temperature for the human body, hot or cold. Defuzzification: It is the inverse of fuzzification. The former one was used to convert the crisp results into fuzzy results but here the mapping is done to convert the fuzzy results into crisp results. This process can generate a non-fuzzy control action which illustrates the possibility distribution of an inferred fuzzy control action. Defuzzification process can also be treated as the rounding off process, where fuzzy set having a group of membership values on the unit interval reduced to a single scalar quantity.
‘

Difference Between Defuzzification and Fuzzification

S.No.	Comparison	Fuzzification	Defuzzification
1.	Basic	Precise data is converted into imprecise data.	Imprecise data is converted into precise data.
2.	Definition	Fuzzification is the method of converting a crisp quantity into a fuzzy quantity.	Defuzzification is the inverse process of fuzzification where the mapping is done to convert the fuzzy results into crisp results.
3.	Example	Like, Voltmeter	Like, Stepper motor and D/A converter
4.	Methods	Intuition, inference, rank ordering, angular fuzzy sets, neural network, etcetera.	Maximum membership principle, centroid method, weighted average method, center of sums, etcetera.
5.	Complexity	It is quite simple.	It is quite complicated.
6.	Use	It can use IF-THEN rules for fuzzifying the crisp value.	It uses the center of gravity methods to find the centroid of the sets.

Fuzzy interference System

What is Fuzzy Inference System and How it works?

What is Fuzzy Inference System (FIS)? Fuzzy inference system is key component of any fuzzy logic system. It uses fuzzy set theory, IF-THEN rules and fuzzy reasoning process to find the output corresponding to crisp inputs. Predicates in IF-THEN rules are connected using *and* or *or* logical connectives.

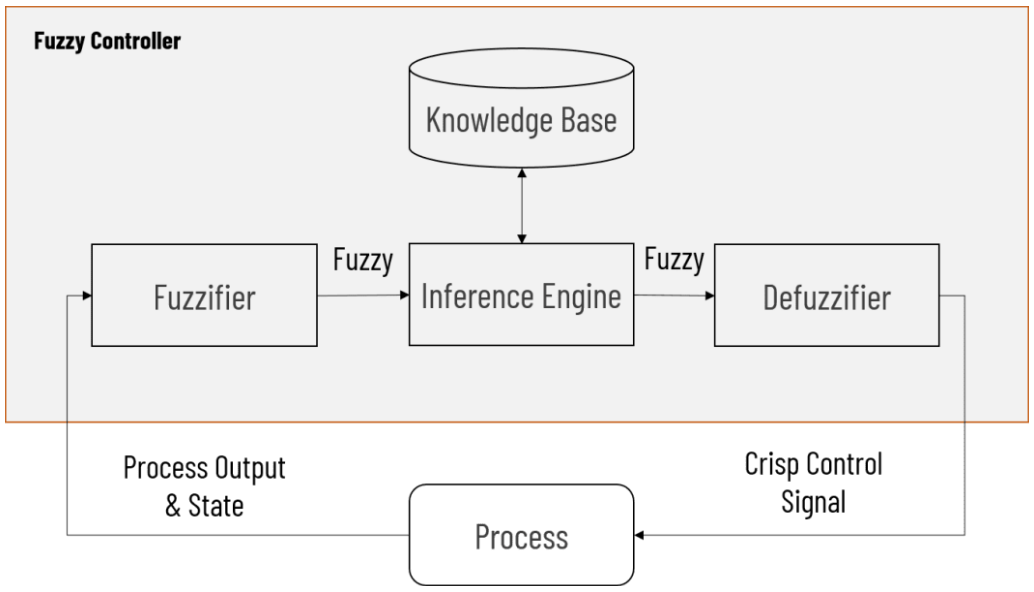
Characteristics of FIS:

- Read crisp value from the process
- Maps the crisp value into fuzzy value using fuzzy membership function
- Apply IF-THEN rules from fuzzy rule base and compute fuzzy output
- Convert fuzzy output into crisp by applying some defuzzification methods.

How fuzzy inference system works?

Crisp input of any process (measuring temperature of air conditioner, measuring altitude, attitude, height, angle of direction for airplane etc.) is given to the fuzzifier, which applies fuzzy membership function and maps the actual readings into fuzzy value (i.e. the value between 0 to 1).

Inference engine applies fuzzy rules from knowledge base and produce the fuzzy output, which is again between 0 and 1. This output can not be used directly into any process or system. It needs to be mapped into original domain. Defuzzifier is the inverse process of fuzzification, it converts the fuzzy output into crisp output, which can be fed to the process. Crisp sets are internally converted to fuzzy sets.



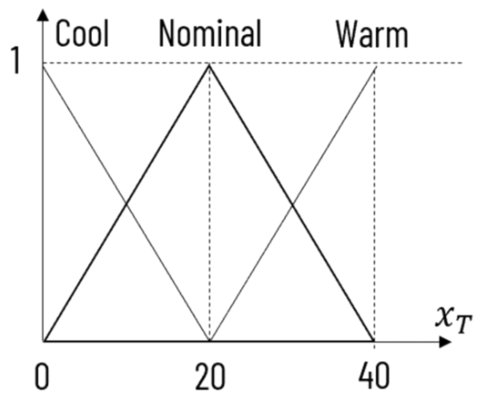
Fuzzy inference system

Lets take the scenario of air conditioner. For simplicity we will consider only one parameter which determines the temperature of air conditioner to be set. Let us consider that the temperature of air conditioner depends on the parameter called room temperature.

Let us divide the range of room temperature to cool, nominal and warm. Any temperature value may belong to multiple sets with different membership value. The value t = 20 has membership value 1 in nominal set and 0 in other two. If we move on right, the membership value in nominal set decreases and in warm set it increases. And as we move left from t = 20, the membership value in nominal set decreases and in cool set it increases

Any room temperature is converted to fuzzy value using such fuzzification methods (here we have used triangular membership function, more are discussed later in this article). And that input goes to inference engine, which will determine what should be the temperature for air conditioner. Assume that the internal representation of that temperature belongs to increase and decrease set.

So the generated output might have some membership for both the increase and decrease sets. Assume that its 0.7 for increase class and 0.15 for decrease class. Although output is computed, it can not be given directly to the controller of AC, because controller does not understand what is meaning of membership value of temperature 0.7 in increase class and 0.15 in decrease class.



Linguistic representation of temperature

Number of defuzzification methods are there which can be used to convert this fuzzy output into crisp one. For particular instance, the crisp output may be -4, which says reduce the temperature on air conditioner by 4 degree.

Approaches for Fuzzy Inference System

Following are the popular approaches to fuzzy inference system. Antecedent part of all rules remain same, they differ only in consequent part.

- Mamdani fuzzy inference system
- Takagi-Sugeno fuzzy inference system
- Tsukamoto fuzzy inference system

Defuzzification

Defuzzification is the process of obtaining a single number from the output of the aggregated fuzzy set. It is used to transfer fuzzy inference results into a crisp output. In other words, defuzzification is realized by a decision-making algorithm that selects the best crisp value based on a fuzzy set. There are several forms of defuzzification including center of gravity (COG), mean of maximum (MOM), and center average methods. The COG method returns the value of the center of area under the curve and the MOM approach can be regarded as the point where balance is obtained on a curve.

The most commonly used defuzzification method is the center of area method (COA), also commonly referred to as the *centroid* method. This method determines the center of area of fuzzy set and returns the corresponding crisp value. The center of sums (COS) method and the mean of maximum method are two alternative methods in defuzzification.