Subject Name: Soft Computing **Subject Code: IT 701**

Syllabus: Unit V

Genetic Algorithm: Introduction to GA, Simple Genetic Algorithm, terminology and operators of GA (individual, gene, fitness, population, data structure, encoding, selection, crossover, mutation, convergence criteria). Reasons for working of GA and Schema theorem, GA optimization problems like TSP (Travelling salesman problem), Network design routing. Introduction to Ant Colony optimization (ACO) and Particle swarm optimization (PSO).

Course Objectives:

1) The objective of this course is to understand genetic algorithm and its operators.
2) To understand genetic algorithm optimization problems

Introduction to Genetic Algorithm

Genetic Algorithm (GA) is a search-based optimization technique based on the principles of Genetics and Natural Selection. It is frequently used to find optimal or near-optimal solutions to difficult problems which otherwise would take a lifetime to solve. It is frequently used to solve optimization problems, in research, and in machine learning.

What are Genetic Algorithms?

Nature has always been a great source of inspiration to all mankind. Genetic Algorithms (GAs) are search based algorithms based on the concepts of natural selection and genetics. GAs are a subset of a much larger branch of computation known as Evolutionary Computation.

In GAs, we have a pool or a population of possible solutions to the given problem. These solutions then undergo recombination and mutation (like in natural genetics), producing new children, and the process is repeated over various generations. Each individual (or candidate solution) is assigned a fitness value (based on its objective function value) and the fitter individuals are given a higher chance to mate and yield more "fitter" individuals. This is in line with the Darwinian Theory of "Survival of the Fittest".

Genetic Algorithms are sufficiently randomized in nature, but they perform much better than random local search (in which we just try various random solutions, keeping track of the best so far), as they exploit historical information as well.

Terminology & Operators of GA

Individual: An individual is a single solution. An individual groups together two forms of solutions as given below:
1)The chromosome which is the raw "genetic" information (genotype) that the GA deals.
2)The phenotype which is the expressive of the chromosome in the terms of the model.

Genes: Genes are the basic "instructions" for building a GA. A chromosome is a sequence of genes. Genes may describe a possible solution to a problem, without actually being the solution. A gene is a bit string of arbitrary lengths. The bit string is a binary representation of number of intervals from a lower bound. A gene is the GA's representation of a single factor value for a control factor, where control factor must have an upper bound and a lower bound.

Fitness: The fitness of an individual in a GA is the value of an objective function for its phenotype. For calculating fitness, the chromosome has to be first decoded and the objective function has to be evaluated. The fitness not only indicates how good the solution is, but also corresponds to how close the chromosome is to the optimal one.
In case of multi criterion optimization, the fitness function is definitely more difficult to determine.

Population: A population is a collection of individuals. A population consist of a number of individuals being tested, the phenotype parameters defining the individuals and some information about the search space. The two important aspects of population used in GA are:

1. The initial population generation
2. The population size.

   For each and every problem, the population size will depend on the complexity of the problem. It is often a random initialization of population.

Data Structure: The basic data structure of a GA is as follows −
We start with an initial population (which may be generated at random or seeded by other heuristics), select parents from this population for mating. Apply crossover and mutation operators on the parents to generate new off-springs. And finally these off-springs replace the existing individuals in the population and the process repeats. In this way genetic algorithms actually try to mimic the human evolution to some extent.
A generalized pseudo-code for a GA is explained in the following program –

```
GA()
   initialize population
   find fitness of population
   while (termination criteria is reached) do
      parent selection
      crossover with probability pc
      mutation with probability pm
      decode and fitness calculation
      survivor selection
      find best
 return best
```

Structure of basic genetic algorithm:

1) Start: generate a random population initial. This population can be seen like a collection of chromosomes, as the individuals are reduced to the representation of his notable characteristics for the problem (chromosome); Structure and Operation of a Basic Genetic Algorithm.
2) Fitness: Evaluate each chromosome, by the function of fitness;
3) New population: produce a new population (or generate, or descendants), by execution of the following steps, so many times, those that the new individuals pretended;
a. Selection: select two chromosomes for crossing, respecting that while more fitness greater his probability of selection.
b. Crossing: The chromosomes of the parents have to cross somehow.
c. Mutation: consider, with low probability, the application of mutation in some position of the descendant's chromosomes.
d. To accept: to accept the descendant and place it in the new population.
4) Replacement: To substitute the old population by the new population, generated in the step 3.
5) Test: a. To test the condition of the algorithm; b. If satisfied, finish, producing how «better solution» (not confusing like optimum solution) the common population; of the contrary, continue.
6) Goto 2

Encoding: Encoding is a process of representing individual genes. The process can be performed using bits, numbers, trees, arrays, lists or any other objects. The encoding depends mainly on solving the problems.
Types of encoding

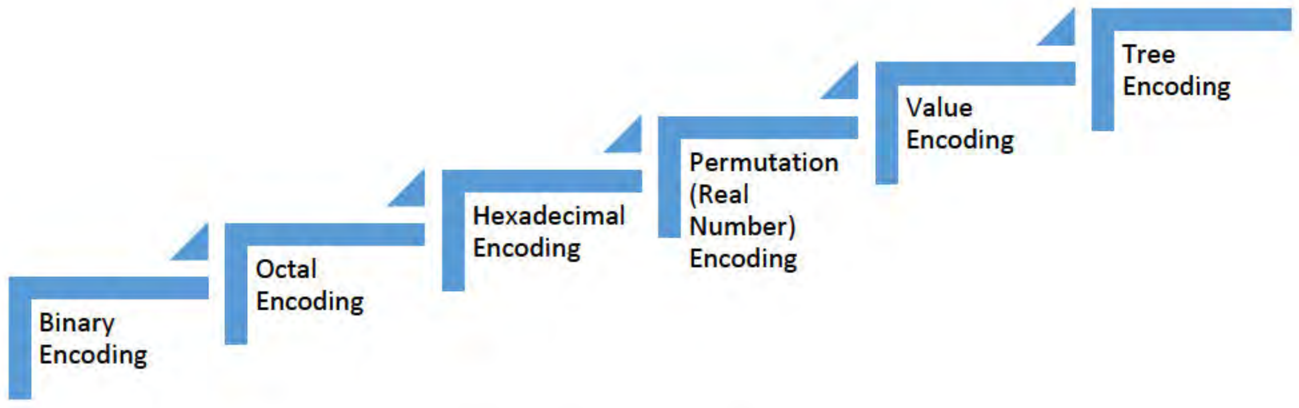


Figure 5.1: Types of Encoding

Selection: Selection is the process of choosing two parents from the population for crossing. After deciding on an encoding, the next step is to decide how to perform selection, i.e., how to choose individuals in the population that will create offspring for the next generation and how many offspring each will create.

Various Selection methods:

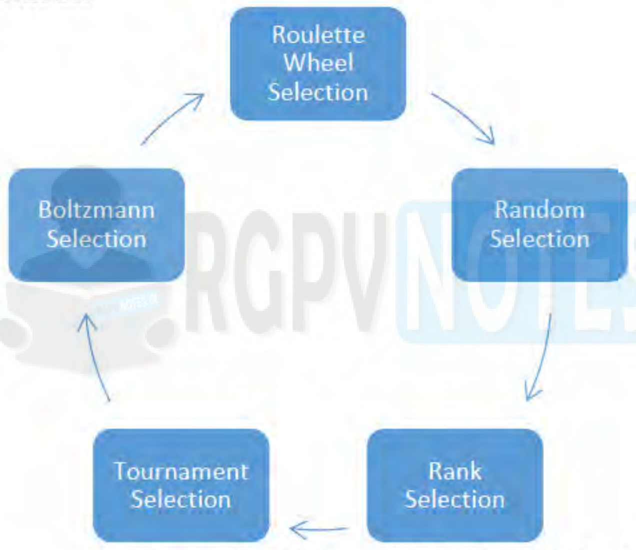


Figure 5.2: Types of Selection Methods

The purpose of selection is to emphasize fitter individuals in the population in hopes that their offspring have higher fitness.

Crossover (Recombination): Crossover is the process of taking two parent solutions and producing from them a child. After the selection (reproduction) process, the population is enriched with better individuals. Reproduction makes clones of good strings but does not create new ones. Crossover operator is applied to the mating pool with the hope that is creates a better offspring.

Crossover is a recombination operator that proceeds in three steps:

a. The reproduction operator selects at random a pair of two individual strings for the mating.
b. A cross site is selected at random along the string length.
c. Finally, the position values are swapped between the two strings following the cross site.

Various Crossover Techniques:

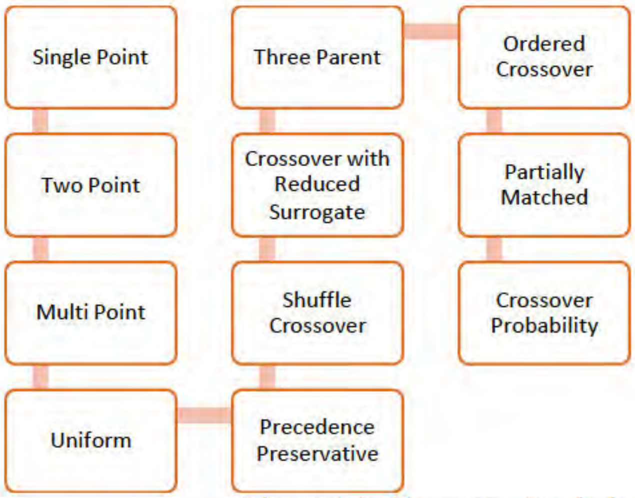


Figure 5.3: Crossover Techniques

Mutation: After crossover operation strings are subjected to mutation. Mutation prevents the algorithm to be trapped in a local minimum. Mutation plays the role of recovering the lost genetic materials as well as for randomly distributing genetic information. It is an insurance policy against the irreversible loss of genetic material. Mutation has been traditionally considered as a simple search operator. If crossover is supposed to exploit the current solution to find better ones, mutation is supposed to help for the exploration of the whole search space. Mutation is viewed as a background operator to maintain genetic diversity in the population.

Convergence criteria:

Convergence is a phenomenon in evolutionary computation. It causes evolution to halt because precisely every individual in the population is identical. Full convergence might be seen in genetic algorithms (a type of evolutionary computation) using only crossover (a way of combining individuals to make new offspring). Premature convergence is when a population has converged to a single solution, but that solution is not as high of quality as expected, i.e. the population has gotten 'stuck'. However, convergence is not necessarily a negative thing, because populations often stabilize after a time, in the sense that the best programs all have a common ancestor and their behavior is very similar (or identical) both to each other and to that of high fitness programs from the previous generations. Often the term convergence is loosely used. Convergence can be avoided with a variety of diversity-generating techniques.

The ideal convergence criterion for a genetic algorithm would be one that guaranteed that each and all of the parameters converge independently Beasley et al. (1993a); Goldberg (1989). However, this may be too demanding or may result in too many iterations, so more relaxed convergence criteria are usually employed. Here I used four convergence criteria:

1.The fitness function value must be below a given threshold value.
2.The difference between the best and the average fitness is less than a given fraction of the fitness of the average individual.
3.The difference between the best individual of the current population and the best individual so far must be very small (even zero). This means that the most-fit individual has converged even if the population itself has not.
4. The number of iterations (generations of the sample population) exceeds a given limit. This prevents the algorithm from spending too much time refining an existing solution.

The combination of these criteria is intended to guarantee that the solution is not due to a lucky guess of the random generator but to a comprehensive search of the model space.

Simple Genetic Algorithm

GA handles a population of possible solutions. Each solution is represented through s chromosome, which is just an abstract representation. Coding all the possible solutions into a chromosome is the first part, but certainly not the most straightforward one of GA. A set of reproduction operators has to be determined too. Reproduction operators are applied directly on the chromosome, and are used to perform mutations and recombination over solutions of the problem. The simple form of GA is given as:

1. Start with a randomly generated population.
2. Calculate the fitness of each chromosome in the population.
3. Repeat the following steps until n offsprings have been created:
a. Select a pair of parent chromosome from the current population
b. With probability P_c crossover the pair at a randomly chosen point to from two offspring
c. Mutate the two offspring at each locus with probability P_m
4. Replace the current population with the new population.
5. Go to step 2

General Genetic Algorithm

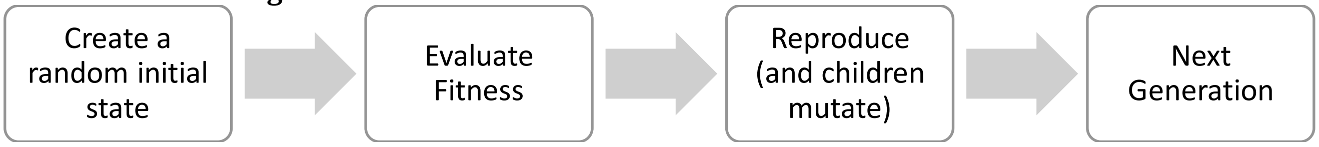

Create a random initial state
Evaluate Fitness
Reproduce (and children mutate)
Next Generation

Figure 5.4: Flow of General Genetic Algorithm

Reasons for working of GA:

1. The capability of GA to be implemented as a 'universal optimizer' that could be used for optimizing any type of problem belonging to different fields.
2. Simplicity and ease of implementation.
3.Proper balance between exploration and exploitation could be achieved by setting parameters properly.
4. Logical reasoning behind the use of operators like selection, crossover and mutation.
5. Mathematical or theoretical analysis in terms of schema theory or Markov chain models for the success of GA.
6. One of the pioneer evolutionary algorithms.
7. Solving both constrained and unconstrained optimization problems that is based on natural selection.

The Schema Theorem

Holland's schema theorem, also called the fundamental theorem of genetic algorithms is an inequality that results from coarse-graining an equation for evolutionary dynamics. The Schema Theorem says that short, low-order schemata with above-average fitness increase exponentially in frequency in successive generations. The theorem was proposed by John Holland in the 1970s. It was initially widely taken to be the foundation for explanations of the power of genetic algorithms.
For example, consider binary strings of length 6. The schema 1*10*1 describes the set of all strings of length 6 with 1's at positions 1, 3 and 6 and a 0 at position 4. The * is a wildcard symbol, which means that positions 2 and 5 can have a value of either 1 or 0. The order of a schema o(H) is defined as the number of fixed positions in the template, while the defining length delta (H) is the distance between the first and last specific positions. The order of 1*10*1 is 4 and its defining length is 5. The fitness of a schema is the average fitness of all strings matching the schema. The fitness of a string is a measure of the value of the encoded problem solution, as computed by a problem-specific evaluation function. Using the established methods and genetic operators of genetic algorithms, the

schema theorem states that short, low-order schemata with above-average fitness increase exponentially in successive generations. Expressed as an equation:

$$\mathrm{E}(m(H,t+1)) \geq \frac{m(H,t)f(H)}{a_t}[1-p].$$

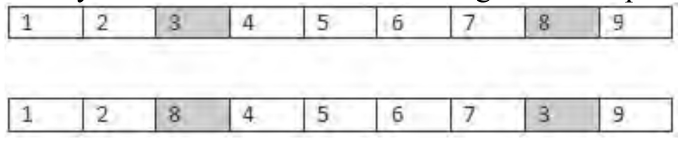
Here m(H,t) is the number os string belonging to schema Hat generation t, f(H) is the observed average fitness of schema H and a_t is the observed average fitness at generation t. The probability of disruption p is the probability that crossover or mutation will destroy the schema.

GA Optimization Problem using JSSP

Shop scheduling problems belong to the class of multi-stage scheduling problems, where each job consists of a set of operations. For describing these problems, we use the standard 3-parameter classification a | b | c. The parameter a indicates the machine environment, the parameter b describes job characteristics, and the parameter c givesthe optimization criterion.
In such a shop scheduling problem, a set of n jobs J_1, J_2, …..J_n has to be processed on a set of m machines M_1, M_2,…… M_m. The processing of a job J_i on a particular machine M_j is denoted as an operation and abbreviated by (I, j). Each job J_i consists of a number n_i of operations. For the deterministic scheduling problems, the processing time P_{ij} of each operation (i,j) is given in advance.
Among the shop scheduling problems, there are three basic types: a flow-shop, a job shop and an open-shop. In a flow shop problem (a = F), each job has exactly m operations, and the technological route (or machine order) in which the job passes through the machines is the same for any job. Without loss of generality, we assume that the technological route for any job is given by M_1->M_2->…..->M_m. In a job shop problem (a = J), a specific technological route M_{j1}→M_{j2}→….-->M_{jni} is given for each job Ji; 1<=i<=n. Note that the number of operations per job n_i is equal to m for the classical job shop problems, but this number may be also smaller than m or larger than m (recirculation; in this case we may use the notation (I, j, k) for the k-th processing of job J_i on machine M_j). In an open shop problem (a = O), no technological routes are imposed on the jobs. Usually, it is assumed that each job has to be processed on any machine. In addition to these three major types, there also exist generalizations such as a mixed shop or a general shop.

Travelling Salesman Problem using GA

Finding a solution to the travelling salesman problem requires we set up a genetic algorithm in a specialized way. For instance, a valid solution would need to represent a route where every location is included at least once and only once. If a route contain a single location more than once, or missed a location out completely it wouldn't be valid and we would be valuable computation time calculating its distance.
To ensure the genetic algorithm does indeed meet this requirement special types of mutation and crossover methods are needed.
Firstly, the mutation method should only be capable of shuffling the route, it shouldn't ever add or remove a location from the route, and otherwise it would risk creating an invalid solution. One type of mutation method we could use is swap mutation.
With swap mutation two location in the route are selected at random then their positions are simply swapped. For example, if we apply swap mutation to the following list, [1,2,3,4,5] we might end up with, [1,2,5,4,3]. Here, positions 3 and 5 were switched creating a new list with exactly the same values, just a different order. Because swap mutation is only swapping pre-existing values, it will never create a list which has missing or duplicate values when compared to the original, and that's exactly what we want for the traveling salesman problem.



1	2	3	4	5	6	7	8	9

1	2	8	4	5	6	7	3	9

Now we've dealt with the mutation method we need to pick a crossover method which can enforce the same constraint.
One crossover method that's able to produce a valid route is ordered crossover. In this crossover method we select a subset from the first parent, and then add that subset to the offspring. Any missing values are then adding to the offspring from the second parent in order that they are found. To make this explanation a little clearer consider the following example:
Parents

1	2	3	4	5	6	7	8	9

9	8	7	6	5	4	3	2	1

Offspring

					6	7	8	

9	5	4	3	2	6	7	8	1

Here a subset of the route is taken from the first parent (6,7,8) and added to the offspring's route. Next, the missing route locations are adding in order from the second parent. The first location in the second parent's route is 9 which isn't in the offspring's route so it's added in the first available position. The next position in the parent's route is 8 which is in the offspring's route so it's skipped. This process continues until the offspring has no remaining empty values. If implemented correctly the end result should be a route which contains all of the positions its parents did with no positions missing or duplicated.

A simple genetic algorithm can be defined in the following steps.
Step 1. Create an initial population of P chromosomes.
Step 2. Evaluate the fitness of each chromosome.
Step 3. Choose P/2 parents from the current population via proportional selection.
Step 4. Randomly select two parents to create offspring using crossover operator.
Step 5. Apply mutation operators for minor changes in the results.
Step 6. Repeat Steps 4 and 5 until all parents are selected and mated.
Step 7. Replace old population of chromosomes with new one.
Step 8. Evaluate the fitness of each chromosome in the new population.
Step 9. Terminate if the number of generations meets some upper bound; otherwise go to Step 3.

The TSP has several applications even in its purest formulation, such as planning, logistics, and the manufacture of microchips. Slightly modified, it appears as a sub-problem in many areas, such as DNA sequencing. In these applications, the concept city represents, for example, customers, soldering points, or DNA fragments, and the concept distance represents travelling times or cost, or a similarity measure between DNA fragments.

Network Design Routing using GA
The purpose of GA is to determine an ‘‘optimal’’ or ‘‘efficient’’ route set (comprising of a pre-specified (or fixed) number of routes) for a given road network and transit demand matrix.
The proposed algorithm, the conceptual overview which follows a three step iterative process. The important features of the proposed algorithm are:

- Various reasonable route sets for the given road network and demand matrix, aredetermined using a novel stochastic procedure, IRSG. It may be pointed out that this heuristic procedure, which onlyprovides a starting point for the proposed optimization procedure.
- After the initial one-time use of IRSG, the evaluation step, and the route modification step are executed repeatedly one after the other until a route set is obtained which satisfies the basic

properties of an efficient route set maximally; i.e. until an ''efficient'' or ''optimal'' route set obtained.

- The goodness of a route set as a whole is determined in the evaluation step using the evaluation scheme, EVAL.
- Given a group of route sets and their goodness, the route sets are modified using the proposed modification procedure, MODIFY, in order to obtain a better group of route sets. The modification is done using the evolutionary principles of genetic algorithms.

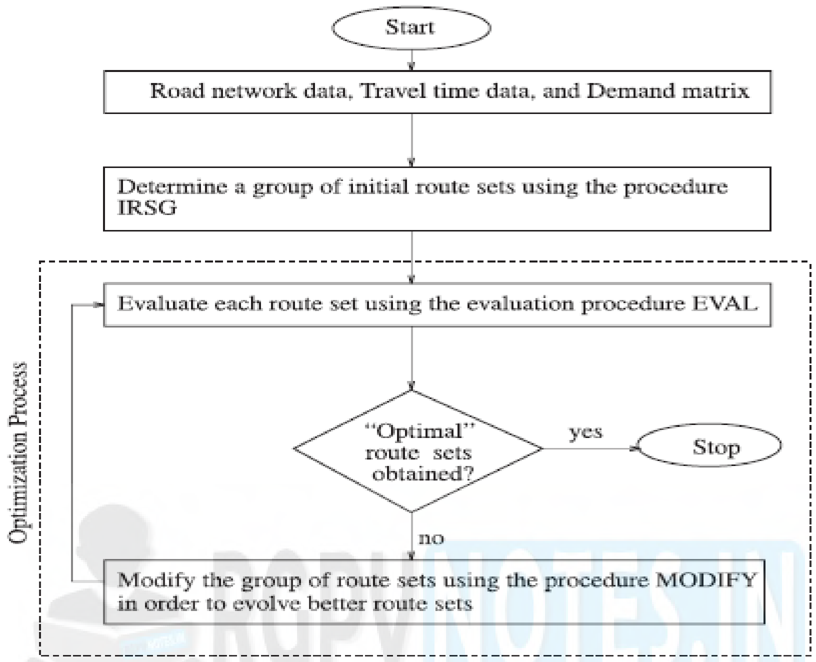

Start
Road network data, Travel time data, and Demand matrix
Determine a group of initial route sets using the procedure IRSG
Optimization Process
Evaluate each route set using the evaluation procedure EVAL
''Optimal'' route sets obtained?
yes
Stop
no
Modify the group of route sets using the procedure MODIFY in order to evolve better route sets

Figure 5.5: Block Diagram of Network Design Routing

Timetabling Problem Solution using GA

1. Generate a random population P
2. **while** Termination condition is not met **do**
3. **for each** individual i of P **do**
4. Generate a timetable using individual i
5. Evaluate the generated timetable using the fitness function
6. **end for**
7. Apply variation operators
8. Generate a new population P
9. End while

The main operators we use at the evolution stage are crossover and mutation. Extra operators can be easily added if necessary.
In order to calculate the fitness value of an individual, a timetable must be generated.
For this, the values in the individual (vector) are used to calculate all arrival and departure times. After the timetable is generated, it is evaluated using a slightly simplified version of the objective function.
The fitness function consists in minimizing the total weighted sum of constraints violations for the timetable.

GA implementation using MATLAB

MATLAB has a wide variety of functions useful to the genetic algorithm practitioner. Given the versatility of MATLAB's high-level language, problems can be coded in m-files in a fraction of the time that it would take to create C or FORTRAN programs for the same purpose. Couple this with MATLAB's advanced data analysis, visualization tools and special purpose application domain toolboxes and the user is presented with a uniform environment with which to explore the potential of genetic algorithms.

The Genetic Algorithm Toolbox uses MATLAB matrix functions to build a set of versatile tools for implementing a wide range of genetic algorithm methods. The Genetic Algorithm Toolbox is a collection of routines, written mostly in m-files, which implement the most important functions in genetic algorithms.

The main data structures used by MATLAB in the Genetic Algorithm toolbox are:

• Chromosomes
• Objective function values
• Fitness values

Introduction to Swarm Intelligence

Swarm Intelligence is a new subset of Artificial Intelligence (AI) designed to manage a group of connected machines. We are now entering the age of the Intelligent machines, also called the Internet of Things (IoT), where more and devices are being connected every day. Swarm intelligence is quickly emerging as a way this connectivity can be harnessed and put to good use.

Swarm intelligence (as the name suggests) comes from mimicking nature. Swarms of social insects, such as ants and bees, operate using a collective intelligence that is greater than any individual member of the swarm. Swarms are therefore highly effective problem-solving groups that can easily deal with the loss of individual members while still completing the task at hand-a capability that is very desirable for a huge number of applications. Today this concept is being applied in concert with machine learning and distributed computing systems. The result is a group of connected machines that can communicate, coordinate, learn and adapt to reach a specific goal. Check out the video below and its subsequent follow up videos to see how swarm intelligence is applied to a group of drones.

Swarm Intelligence Techniques: Ant Colony Optimization

Ant colony optimization (ACO), introduced by Dorigo in his doctoral dissertation, is a class of optimization algorithms modeled on the actions of an ant colony. ACO is a probabilistic technique useful in problems that deal with finding better paths through graphs. Artificial 'ants'—simulation agents—locate optimal solutions by moving through a parameter space representing all possible solutions. Natural ants lay down pheromones directing each other to resources while exploring their environment. The simulated 'ants' similarly record their positions and the quality of their solutions, so that in later simulation iterations more ants locate for better solutions.

The use of swarm intelligence in telecommunication networks has also been researched, in the form of ant-based routing. This was pioneered separately by Dorigo et al. and Hewlett Packard in the mid-1990s, with a number of variations since. Basically, this uses a probabilistic routing table rewarding/reinforcing the route successfully traversed by each "ant" (a small control packet) which flood the network. Reinforcement of the route in the forwards, reverse direction and both simultaneously has been researched: backwards reinforcement requires a symmetric network and couples the two directions together; forwards reinforcement rewards a route before the outcome is known (but then one would pay for the cinema before one knows how good the film is). As the system behaves stochastically and is therefore lacking repeatability, there are large hurdles to commercial deployment.

Mobile media and new technologies have the potential to change the threshold for collective action due to swarm intelligence

.

The location of transmission infrastructure for wireless communication networks is an important engineering problem involving competing objectives. A minimal selection of locations (or sites) is required subject to providing adequate area coverage for users. A very different-ant inspired swarm intelligence algorithm, stochastic diffusion search (SDS), has been successfully used to provide a general model for this problem, related to circle packing and set covering. It has been shown that the SDS can be applied to identify suitable solutions even for large problem instances.

Particle Swarm Optimization

Particle swarm optimization (PSO) is a global optimization algorithm for dealing with problems in which a best solution can be represented as a point or surface in an n-dimensional space. Hypotheses are plotted in this space and seeded with an initial velocity, as well as a communication channel between the particles. Particles then move through the solution space, and are evaluated according to some fitness criterion after each time step. Over time, particles are accelerated towards those particles within their communication grouping which have better fitness values. The main advantage of such an approach over other global minimization strategies such as simulated annealing is that the large numbers of members that make up the particle swarm make the technique impressively resilient to the problem of local minima.

Nanoparticles are bioengineered particles that can be injected into the body and operate as a system to do things drug treatments cannot. The primary problem with all of our current cancer treatments is most procedures target healthy cells in addition to tumors, causing a whole host of side effects. Nanoparticles by comparison, are custom designed to accumulate ONLY in tumors, while avoiding healthy tissue.

Nanoparticles can be designed to move, sense, and interact with their environment, just like robots. In medicine, we call this embodied intelligence. The challenge thus far has been figuring out how to properly "program" this embodied intelligence to ensure it produces the desired outcome.

Swarms are very effective when a group of individual elements (nanoparticles in this case) begin reacting as a group to local information. Swarm intelligence is emerging as the key to which will unlock the true potential of these tiny helpers. Researchers are now reaching out to the gaming community in an effort to crowd source the proper programming for swarm of nanoparticles.

Bee Colony Optimization

The Artificial Bee Colony (ABC) algorithm is a swarm based meta-heuristic algorithm that was introduced by Karaboga in 2005 (Karaboga, 2005) for optimizing numerical problems. It was inspired by the intelligent foraging behavior of honey bees. The algorithm is specifically based on the model proposed by Tereshko and Loengarov (2005) for the foraging behavior of honey bee colonies. The model consists of three essential components: employed and unemployed foraging bees, and food sources. The first two components, employed and unemployed foraging bees, search for rich food sources, which is the third component, close to their hive. The model also defines two leading modes of behavior which are necessary for self-organizing and collective intelligence: recruitment of foragers to rich food sources resulting in positive feedback and abandonment of poor sources by foragers causing negative feedback.

In ABC, a colony of artificial forager bees (agents) search for rich artificial food sources (good solutions for a given problem). To apply ABC, the considered optimization problem is first converted to the problem of finding the best parameter vector which minimizes an objective function. Then, the artificial bees randomly discover a population of initial solution vectors and then iteratively improve them by employing the strategies: moving towards better solutions by means of a neighbor search mechanism while abandoning poor solutions.
The general scheme of the ABC algorithm is as follows:

```
First Initialization Phase
REPEAT

  1. Employed Bees Phase
  2. Onlooker Bees Phase
  3. Scout Bees Phase
  4. Memorize the best solution achieved so far
UNTIL (Cycle=Maximum Cycle Number or a Maximum CPU time)
```