Pranshu Gupta
Indian Institute of Technology Kanpur

Algorithms Inspired by Nature: A Survey

Contents

1 Introduction

Nature is known to be the best optimizer. Natural processes most often than not reach an optimal equilibrium. Scientists have always strived to understand and model such processes. Thus, many algorithms exist today that are inspired by nature. Many of these algorithms and heuristics can be used to solve problems for which no polynomial time algorithms exist, such as Job Shop Scheduling and many other Combinatorial Optimization problems. We will discuss some of these algorithms and heuristics and how they help us solve complex problems of practical importance. We will focus mainly on Ant Colony Optimization, Particle Swarm Optimization, Simulated Annealing and Evolutionary Algorithms.

Remark 0.1. A heuristic can be thought of as a rule of thumb that will hopefully find a good solution for a given problem. These techniques are mainly applied to complex convex multivariate constrained combinatorial optimization problems which are usually NP complete or harder.

Remark 0.2. "A metaheuristic is a higher-level procedure or heuristic designed to find, generate, or select a heuristic that may provide a sufficiently good solution to an optimization problem, especially with incomplete or imperfect information or limited computation capacity." [Bianchi et al. (2009)]

Ant Colony Optimization utilizes a family of heuristics inspired by the foraging behavior of Ants. Ants are highly co-operative social insects, they search for food in huge groups and communicate indirectly using a chemical called pheromone. These heuristics model individual ants as independent solution finders roaming in the solution space of a problem and communicating via a shared variable(s) i.e. the pheromone.

When a liquid is cooled slowly, its particles arrange themselves in some configuration that allows minimum energy state - this process is called annealing. Simulated Annealing tries to model this phenomenon by drawing an analogy between between the behavior of multi-body systems (e.g. liquids) in thermal equilibrium at a finite temperature and the behaviour of the objective function of a combinatorial optimization problem at different values of the variables involved. [Kirkpatrick et al. (1983)]

Particle Swarm Optimization is a another optimization heuristic. It relies on a swarm of particles that are initialized with some configuration in the solution space of a given problem. These particles then explore the solution space individually while communicating with other particles around them to find the best solution. After some iterations these particles are expected to make swarms around optimal solution(s). PSO relies on randomization in the movement of particles to avoid getting stuck in local optima.

Evolutionary Algorithms are a family of algorithms inspired from Neo-Darwinism paradigm of biological evolution.

$$\text{Neo-Darwininsm} = \text{Darwinism} + \text{Selectionism} + \text{Mendelian Genetics}$$

These algorithms model solutions as individual species with their genetic code representing the solution itself. These individuals later reproduce to create new individuals with different genetic code. Reproduction involves choosing healthy parents and then crossing over their genes to create an offspring. The optimality of the solution instance represented by the genetic code of the offspring determines its fitness. Then the notion of "survival of the fittest" comes into play. The best individuals survive and then they reproduce for the next generational cycle. Thus, this model utilizes both exploration and exploitation of the solution space.

Genetic Algorithms, Evolutionary Strategies, Differential Evolution etc., are a few variants of EAs.

2   Ant Colony Optimization

Ant Algorithms are a family of algorithms inspired by the behavior of real ant colonies. Ants are social insects and they always work together to ensure the survival of the colony as a whole. When in search of food, ants deposit a chemical substance called *pheromone* on their path. If they succeed in finding a food source they again deposit pheromone on their path back to the colony. The amount of pheromone they deposit on the path often is determined by the quality and quantity of the food source that the path leads to. Also, if the multiple ants find the same food source, the one which followed the shorter path will deposit the pheromone first. When looking for food the ants also consider the pheromone deposited by other ants to aid their own search. Thus, the shorter paths and the paths which lead to better food sources will be followed by more and more ants and the pheromone on them will be further enriched (positive feedback in search). Also, because pheromone evaporates in atmosphere with the passage of time, paths which are no longer useful or followed will lose the pheromone deposited on them. However, ants are always free to choose their own paths ensuring the explore for new food sources as well instead of just clinging upon only one.

This behavior of ants can be utilized to design algorithms to solve search and optimization problems. Dorigo et al. (1999) were the first to model foraging behavior of ant colonies and design a metaheuristic that can be used to solve problems of practical importance.

2.1   Ant Colony Optimization Metaheuristic

The ACO metaheuristic is usually defined in the spirit of Combinatorial Optimization problems. It is indeed one of the family of approximate algorithms that are used to solve hard CO problems in reasonable amount of time. Let's define a CO problem.

Definition 0.1. A Combinatorial Optimization problem P is defined as $P = (S, \omega, f)$ where S is the solution space for the given problem, ω is the constraint set on the solution parameters and $f : S \to \mathbb{R}^+$ is the objective function to be minimized.

The solution set can be defined as a set of feasible solutions. A feasible solution is a set of N discrete variables $X_i \in D_i$ where $i \in 1, ...n$ which satisfy the solution constraints given by the problem instance.

The pheromone trail model is the is the most crucial component of any ACO framework. It is used to probabilistically build the solution components of the problem instance. A solution component c_{ij} is defined as the assignment of a variable X_i to a value $v_j \in D_i$. The pheromone trail corresponding to this solution component is denoted by T_{ij} with its value being τ_{ij}. The set of all solution components is denoted by C and the vector of pheromone trails is denoted by T.

For ACO, we define a complete graph which has a node corresponding to each possible solution component called the Construction Graph. In each iteration of ACO, each ant moves from one component to another effectively assigning values to the discrete variables (taking care of the constraints and considering deposited pheromone) and thus creating a feasible solution. At the end of every iteration, each ant would have constructed a feasible solution. Then, we compute the fitness of these solutions and deposit pheromone trail values on the edges that the ant which created the solution followed.

3

Algorithm 1: Generic ACO Algorithm [Dorigo and Blum (2005)]

---

1 Initialize Pheromone Trail values T
2 The best solution $S_{bs} \leftarrow$ NULL
3 **while** *termination conditions not met* **do**
4   Set of solutions found in this iteration $S_{iter} \leftarrow \phi$
5   **foreach** *ant in colony* **do**
6 $s \leftarrow$ ConstructSolution(T)
7   **if** *s is feasible* **then**
8 $s \leftarrow$ LocalSearch(s)
9   **if** $f(s) < f(s_{bs})$ *OR s_{bs} is NULL* **then**
10 $s_{bs} \leftarrow s$
11   **end**
12 $S_{iter} \leftarrow S_{iter} \cup \{s\}$
13   **end**
14   **end**
15   ApplyPheromoneUpdate(T, S_{iter}, s_{bs})
16 **end**
17 Return s_{bs}

The probability that an ant will choose a solution component c_{ij} as part of its solution conditioned on the set of solution components it has already chosen (the partial solution s^p) is given by:

$$p(c_{ij}|s^p) = \frac{\tau_{ij}^{\alpha} \cdot c_{ij}^{\beta}}{\sum_{c_{lk} \in N(s^p)} \tau_{lk}^{\alpha} \cdot c_{lk}^{\beta}}$$

Here, c_{ij} denotes the problem specific cost of choosing the assignment j for decision variable i. α and β are parameters than are used to tune the relative importance of the problem specific costs and pheromone trail value corresponding to solution component. $N(s^p)$ is the set of allowed solution components given that the partial solution s^p has already been constructed. $N(s^p)$ depends on the constraint set of the problem instance. If the constraint set does not allow the selection of a solution component given the partial solution, the corresponding transition probability is 0.

The pheromone update rule is given as follows

$$\tau_{ij} \leftarrow (1-\rho)\tau_{ij} + Q \sum_{k=1}^{m} \frac{1}{f(s_k)}$$

Here, ρ is the evaporation coefficient which models the atmospheric evaporation of the pheromone, Q is some constant, s_k is the solution constructed by the kth ant.

There are other variants of ACO which use different pheromone update rules. Consider the MIN-MAX ACO, the pheromone update rule for it is given by the following equation where s_{iterbs} is the best solution found in this iteration. [Dorigo et al. (2006)]

$$\tau_{ij} \leftarrow (1-\rho)\tau_{ij} + \frac{1}{f(s_{iterbs})}$$

In the Ant Colony System Framework, instantaneous updates are also made to the pheromone trails. This slightly decreases the pheromone value allowing other ants to possible find diverse solutions instead of premature convergence to a sub-optimal solution.

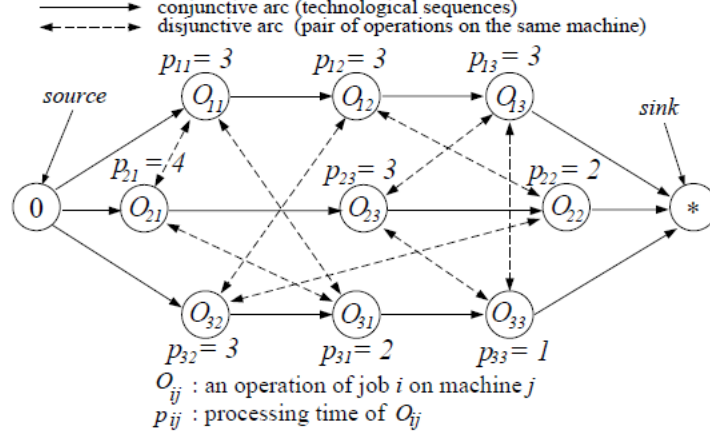$$\tau_{ij} \leftarrow (1-\phi)\tau_{ij} + \phi\tau_0$$

Figure 1: Disjunctive Graph for a JSSP with 3 machines and 3 Jobs each with 3 operations. O_{ij} denotes an operation belonging to job i which has to processed on machine j, p_{ij} denotes the processing time corresponding to the operation. Image taken from [Yamada and Nakano (1997)]

2.2 Applications of Ant Algorithms

2.2.1 Job Shop Scheduling

A Job Shop Scheduling problem consists of a set of n Jobs and a set of m machines. Each job consists of a sequence operations each to be processed on a specific machine. Now, because the resources have to be shared and each operation takes a given amount of time, we want to minimize the total time span required to finish all the jobs - the makespan. There are many techniques to solve this problem - Shifting Bottleneck, SA, ACO, PSO a well as EAs can be used to solve JSSP.

Every JSSP instance can be represented by a disjunctive graph [See Figure 1] in which we have one node for each operation with directed edges from o_i to o_j if the latter comes after the former in the constraint sequence given in the problem. These constraint sequences are generally given for operations that belong to the same job. We dont have a complete ordering across all the operations from different jobs. So, in the disjunctive graph we have un-directed edges between the operations belonging to different jobs but the same machine. If we define a machine specific ordering of operations on each machine, we have essentially solved the problem. We also have a start and finish node to denote the start and finish of the complete sequence of operations [Yamada and Nakano (1997)]

In ACO, we create the construction graph for ant traversal using the disjunctive graph of the problem instance. We add two directed edges in the construction graph for each undirected edge in the disjunctive graph. An ant will choose on of those edges and hence the machine specific ordering will be found. Now, an ant walk from start node to the finish node in the construction graph will essentially give a complete solution. Note that the ant visits all the nodes in the graph one by one. The pheromone update rule captures the makespan as the fitness function and the transition probabilities respect the feasibility of the solution by making the probability zero if the transition creates a cycle in the ant walk, as a cyclic order is non-realizable. [Blum and Sampels (2002)]
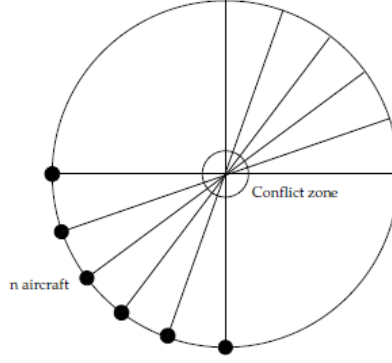
Figure 2: Aircraft Conflict Problem. Image taken from [Durand and Alliot (2009)]
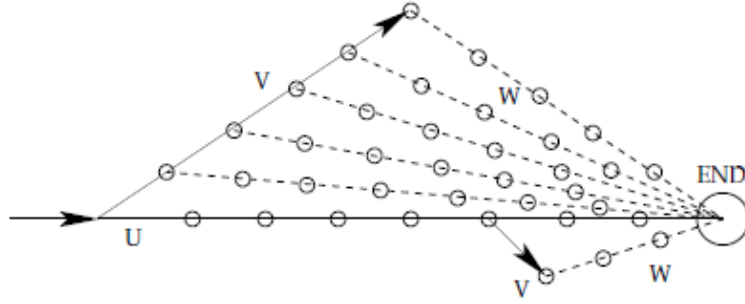

Figure 3: Construction Graph for Aircraft Conflict Problem. Image taken from [Durand and Alliot (2009)]

2.2.2 Aircraft Conflict Resolution

The Aircraft Conflict Resolution problem is of great practical importance. It is combinatorial optimization problem in which a set of n aircrafts approaching a single point have to deviate from their original course in order to avoid collision.

Durand and Alliot (2009) have solved a simplified version of this problem in which all the aircrafts are approaching with the same speed, time is discretized, the set of possible manoeuvres is finte (say $10^o, 20^o, 30^o$) and only two manoeuvres are to be applied to each aircraft. Then this becomes a combinatorial optimization problem wherein we have to choose the manoeuvres at 2 time steps. The construction graph they have designed is shown in Figure 3. Each node is present at a discrete time step and there are three kinds of nodes U, V and W. The only allowed transition between differently typed nodes are U to V and V to W.

Now, in each iteration of ACO, a set of n ants create a single solution for the problem. Each of these ants corresponds to an aircraft and the walk done by the the ant defines the manoeuvres for the aircraft. We take this set on n walks and see if they give a conflict free plan, if they do, the pheromone trail is updated according to the delay due to the course correction as the fitness function. They have demonstrated that this technique can resolve conflicts for upto 30 aircrafts in reasonable time.

2.2.3 Ant Based Clustering

Some species of ants are known to arrange their food supplies in clusters on the basis of different properties of the food items (size etc.). We can also model this behavior of ants to solve clustering problems as well.

Handl et al. (2006) have proposed a simple ant based clustering algorithm that gives natural clusters. Unlike K-Means and other traditional clustering techniques which often need the programmer to specify the number of cluster centers beforehand, this technique gives natural clusters all by itself. They have experimentally shown that it is very robust and gives clusters that are inherent from the dataset, even if the clusters overlap.

Algorithm 2: Ant Based Clustering [Handl et al. (2006)]

---

1 Randomly scatter all the data points in a toroidal grid
2 **foreach** *ant in colony* **do**
3   Ant picks a randomly chosen datapoint (i)
4   Ant places itself at a randomly selected location in the grid
5 **end**
6 **while** *termination conditions not met* **do**
7   Choose a random ant and displace it by a given stepsize
8   Drop the datapoint (i) at this location in the grid with probability given by

$$p = \left(\frac{f^*(i)}{0.3 + f(i)} \right)^2$$

   if *datapoint was dropped* **then**
9   **while** *another datapoint is not picked* **do**
10   Randomly choose a datapoint (i)
11   Pick it up with the probability given by

$$p = \left(\frac{0.1}{0.3 + f(i)} \right)^2$$

12   **end**
13   **end**
14 **end**

The function $f(i)$ is the Lumer and Faieta neighborhood function defined as:

$$f(i) = max\left(0, \frac{1}{\sigma^2} \sum_{j \in L} \left(1 - \frac{\delta(i,j)}{\alpha} \right) \right)$$

where σ^2 is the size of local neighborhood in the toroidal grid, $\delta(i,j)$ is the distance function for the datapoints to be clustered, L is the set of datapoints currently inside the σ^2 neighborhood in the grid. α is data specific scaling parameter. The modified LF function f^* is given as follows:

$$f^*(i) = \begin{cases} f(i) & \text{if } \forall j \left(1 - \frac{\delta(i,j)}{\alpha} \right) \\ 0 & \text{otherwise} \end{cases}$$

This modified LF neighborhood function inflicts heavy penalties for high dissimilarities thus improving the spatial separation between the clusters.

3 Simulated Annealing

Annealing is the process of slowly cooling molten solids to get them in a crystallized state. If the object is cooled rapidly, its particle do not reach an equilibrium state and the crystals formed are small, thus reducing the overall strength of the object. Therefore, the cooling plan in this procedure is very important. In optimization settings, the simulated annealing algorithm we model the particles of the cooling object as the parameters of a possible solution of the given problem. The temperature is modeled as a variable which defines the transition probability of the selected solution. Each run of the algorithm consists of multiple iterations until an equilibrium (convergence) is reached. After that we change the transition probability of the solution by lowering the temperature. We stop the runs when the lower bound of temperature reached. See algorithm 3

Algorithm 3: Simulated Annealing

1 Choose a random configuration X_i, select the initial system temperture T_0 and decide the cooling schedule (usually exponential)
2 Evaluate Cost $C(X_i)$
3 **while** *temperature ¿ lower bound* **do**
4 Perturb X_i to get a neighbor configuration X_{i+1}
5 Evaluate Cost $C(X_{i+1})$
6 **if** $C(X_{i+1}) < C(X_i)$ **then**
7   Set the new solution configuration to $C(X_{i+1})$
8 **else**
9   Set the new solution configuration to $C(X_{i+1})$ with probability p

$$p = e^{-(C(X_{i+1})-C(X_i))/T_k}$$

10 **end**
11 **if** *Equilibrium reached at T* **then**
12   Reduce the system temperature to

$$T_{k+1} = (\frac{T_1}{T_0})^k T_k$$

13 **end**
14 **end**

3.1 Solving Job Shop Scheduling with Simulated Annealing

The objective of the Job Shop Scheduling problem is to find out the optimal sequence of jobs on a given set of machines so that the makespan is minimized. JSSP is a NP Hard problem and Simulated Annealing can be used to solve it. Van Laarhoven et al. (1992) have shown that SA performs very well on JSSP instances. They modeled the configuration as the set of machine specific Job sequences with the makespan time as their costs. The neighborhood structure is defined as the set of configurations that can be obtained by reversing the order of two successive operations (on the same machine) on a critical path in the disjunctive graph. The perturbations in the SA give one of the neighbor solutions. This way the SA procedure can be adopted to solve any JSSP instance. Similarly, SA can be used to other combinatorial optimization problems like TSP and Physical Chip Design [See Kirkpatrick et al. (1983)]

4 Particle Swarm Optimization

In Particle Swarm Optimization, a number of particles are initially scattered in the search space of a given problem. The fitness of each particle can be evaluated using its location in the search space. Now, in each iteration every particle displaces itself to a new position. This new position is determined by the history of best positions of this particle as well as some other particles in the swarm. It is in fact the weighted mean of the particle's own best position and the neighbors best position. The weights are picked uniformly from range of positive numbers. Eventually, the swarm as a whole is likely to move close to the optimum of the fitness function. See Algorithm 4 for a generic PSO technique.

Algorithm 4: Particle Swarm Optimization

---

1 Initialize a population of particles at random positions in the search space
2 **while** *termination conditions not met* **do**
3   For each particle i evaluate the fitness function at its current position
4   Update the best fitness value $pbest_i$ and position p_i for this particle as per step 3
5   Identify the best fitness particle g in the neighborhood
6   Update the position and velocity of the particle i as per the following equations

$$v_i \leftarrow v_i + U(0, \phi_1) \otimes (p_i - x_i) + U(0, \phi_2) \otimes (p_g - x_i)$$
$$x_i \leftarrow x_i + v_i$$

7 **end**

This works quite well if the solution space of the problem is continuous, unlike the discrete valued solution spaces of problems like TSP, JSP etc. Defining velocities is a crucial part of the procedure, but according to Clerc (2010), where a version of PSO for TSP is presented, velocity can be viewed as an operator which when applied to the current position gives a new position from the neighborhood.

There are many other variants of the Standard PSO, one in which the velocity of the particles is bounded, another in which the updates are determined not only by the particle's own history and its best neighbor but its entire neighborhood - Fully Informed Particle Swarm Optimization [Poli et al. (2007)]. FIPSO has been demonstrated to work better than the standard PSO - but it is sensitive to the way the neighborhood is defined.

4.1 Applications of Particle Swarm Optimization

4.1.1 Solving Job Shop Scheduling with Particle Swarm Optimization

In PSO, the configuration and neighborhood definitions are the same as those in Simulated Annealing. However, we need to define the notion of velocity, the distance between two configurations and the sum of two velocities. The velocity itself can be defined as a permutation function that gives a new machine specific sequence of operations from the old one. Thus, the velocity can be defined as a list of pair of operations that belong to same machine, these pairs are to be exchanged in the machine specific ordering to get the new position i.e. a list of transposition. The distance between two positions is the velocity that must be applied to the first position to reach the second. Finally, the sum of two velocities is the order preserving truncated union of the transposition sets used to define the two velocities. With these definitions at hand we can go ahead and use PSO to solve JSSP. See Shao et al. (2013)

4.1.2 A Hybrid of SA and PSO to solve JSSP

In a hybrid solution proposed by Shao et al. (2013), we initialize a large number of particles using an approach proposed by Pezzella et al. (2008). The new position for each particle is found by simulated annealing approach. Thus, the structure of the overall algorithm remains like that of PSO (with SA embedded in it).

The advantage of this hybrid framework is that it takes the best from SA - local search and PSO - global search. Thus, avoiding local optima better than traditional SA, maintaining diversity in solutions and identifying better neighbors than the traditional PSO.

5 Evolutionary Algorithms

Evolutionary Algorithms are a class of general purpose algorithms that are designed to solve optimization problems that are large, complex, discontinuous, non-differentiable, multi-modal and possibly constrained as well. EAs are highly parallelizable, simple and are known to almost always reach a near optimum or the global optimum.

Any Evolutionary Algorithm generally comprises of the following components - a population generator, a population selector, fitness estimator and reproduction operators (crossover and mutation). The pseudocode of a generic EA can be given as Algorithm 5

Algorithm 5: A Generic Evolutionary Algorithm [Du and Swamy (2016)]

---
1 $t = 0$
2 Randomly generate initial population $P(0)$
3 **while** *termination condition not met* **do**
4   Evaluate fitness of each individual of $P(t)$
5   Select individuals as parents from $P(t)$ based on their fitness
6   Apply search operators (crossover and mutation) on parents and generate $P(t+1)$
7 $t := t + 1$
8 **end**

$P(t)$ denotes the current set of individuals in the population. Each individual in the population is called a chromosome which represents a solution to the given problem. The chromosome is denoted as a string of elements called genes which represent individual parameters of the solution. An individual's chromosome is termed as the genotype and genotype when denoted along with fitness and other properties of the corresponding solution is called a phenotype.

5.1 Encoding a solution as a Chromosome

There are different techniques to represent a solution as a chromosome, each having their pros and cons. The performance of these algorithms depends on the encoding technique used, because the crossover and mutation operators are applied on these representations only to explore the solution space. Some of the notable representation techniques are Binary Coding, Gray Coding, Delta Coding and Fuzzy Coding.

5.1.1 Binary Coding

Genetic Algorithms use Binary Encoding wherein each chromosome is an array of genes x. Each of these genes x_i represents some parameter of the solution in form of binary string of length l. To recover the actual value of the parameter whose value lies in the interval $[x_i^-, x_i^+]$ we use

Equation 1 [Du and Swamy (2016)]

$$x_i = x_i^- + (x_i^+ - x_i^-)\frac{1}{2^{l_i} - 1}(\sum_{j=0}^{l_i-1} s_j 2^i) \qquad (1)$$

Binary Coding is a fixed length and static coding which sometimes leads to slower convergence and lower accuracy when it reaches near the optimal solution. Therefore, new dynamic coding techniques were developed, such as Delta and Fuzzy Coding.

5.1.2 Delta Coding

Delta Coding is a dynamic coding technique in which the representation of the search space changes in after each iteration. It starts with Binary Coding but modifies the representation by changing the bit length of the solution parameters after each iteration. The solution parameters in subsequent iterations are denoted with respect to the S_{best} from the previous iteration. The distance from S_{best} is called the Delta Value. This creates a new hypercube with S_{best} as its origin at each iteration which is then searched using the crossover operators. In order to maintain diversity in the population in order to explore the search space, we re-initialize the population in each iteration. See Algorithm 6 [Mathias and Whitley (1994)].

Algorithm 6: Delta Coding

1 Initialize the population randomly $P(0)$
2 **while** *trials < max.Trials & fitness > threshold* **do**
3   **while** *Diversity Metric > 1* **do**
4   Apply Crossover Operators to get candidates for $P(t+1)$
5   Evaluate fitness function for each individual
6   Insert valid off-springs to $P(t+1)$
7   **end**
8   Set the best new solution as S_{best}
9   Re-initialize Population
10   **if** *Delta Value == 0* **then**
11   **if** *Parameter Length > Original Parameter Length* **then**
12   Encode parameter using 1 additional bit
13   **end**
14   **else**
15   **if** *Parameter Length > Lower Bound* **then**
16   Encode parameter using 1 less bit
17   **end**
18   **end**
19 **end**

5.1.3 Fuzzy Coding

In fuzzy coding, for each parameter of the solution we choose one or more from a set of fuzzy sets (NM, Ms, ZR, PS, PM). Each solution parameter has a membership function (fuzzy sigmoid, fuzzy gaussian or fuzzy normal). These membership functions are used to choose values from selected fuzzy sets corresponding to the solution parameter. The value of the solution parameter

is denoted by the weighted mean of these chosen vales as per the membership function weights. The membership functions for the fuzzy set NM are given as follows [Sharma and Irwin (2003)]:

$$\text{Fuzzy Sigmoid}: y = -ae^{-(\frac{x-c}{\frac{d}{3}})^2} - b - 1$$

$$\text{Fuzzy Gaussian}: y = ae^{-(\frac{x-c}{\frac{d}{3}})^2} + b$$

$$\text{Fuzzy Normal}: y = 1 - \frac{x-c}{d}$$

IN this coding technique, the genetic algorithm need only optimize the fuzzy sets and the membership function corresponding to each solution parameter. As the actual value of the parameter is indirectly obtained using the chosen fuzzy sets and the membership function, the genetic operators (crossover and mutation) are independent of the actual values. We only need to specify the type of the parameter and its range beforehand.

5.2 Selection and Replacement

In every iteration, we need to choose the set of parents from the population, this can be done by using various policies. Some of the important selection policies are Roulette Wheel Selection, Ranking Selection, Tournament Selection and Elitist Selection.

5.2.1 Roulette Wheel Selection

In this policy, the selection probability of an individual is proportional to its fitness value. We can think of this policy as a roulette wheel which is partitioned as per the fitness of each individual.

$$P_i = \frac{f(x_i)}{\sum_{i=1}^{N_P} f(x_i)} \qquad\qquad i = 1, 2, ..., N_P$$

5.2.2 Ranking Selection

In this policy, the individuals are sorted in ascending order according to their fitness and the best one is assigned rank 1. This policy applies a uniform selective pressure on all the individuals unlike the roulette wheel policy. The probability of the selection of ith individual as a parent is given by:

$$P_i = \frac{1}{N_P}(\beta - 2(\beta - 1)\frac{i-1}{N_P - 1}) \qquad\qquad \beta \in [0, 2]$$

5.2.3 Tournament Selection

In this policy two individuals i and j are picked up randomly with replacement. The probability that the individual i is selected is given by:

$$P_i = \frac{1}{1 + exp(\frac{f_j - f_i}{T})}$$

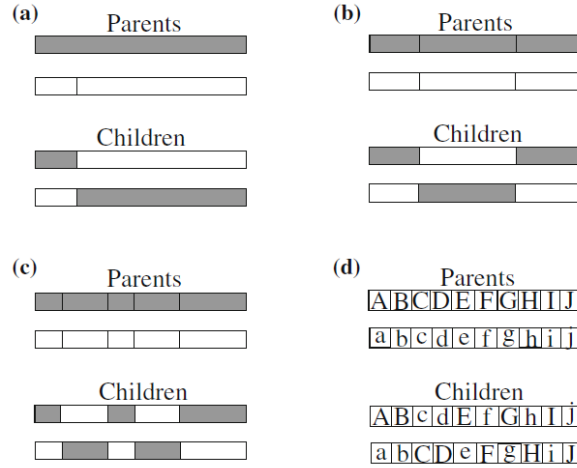We stop after the required number of parents have been selected.

Figure 4: Crossover Operators - a) One Point b) Two Point c) Multi Point d) Uniform Crossover.
Image taken from [Du and Swamy (2016)]

5.2.4 Replacement

After choosing the parents and obtaining the offspring, we replace some individuals from the population. There are several policies to accomplish this too - complete generational replacement, random replacement, replace worst policy, replace oldest policy and replce the similar parent policy.

5.3 Reproduction

Reproduction is simulated by Crossover and Mutation operators. Crossover operator takes two individuals as parents and gives two new individuals by swapping some parts of the chromosomes of the parents. Mutation is a unary operator,it needs only one individual as a parent to create an offspring.

There are different types of crossover operators - one point crossover, two point crossover, multi-point crossover and uniform crossover. [See Figure 4] Uniform Crossover swaps are more disruptive and exploratory hence, suitable for small populations. Two point crossover is generally good for large populations, It has been demonstrated in many experiments that Two Point crossover works better than One Point crossover in most of the cases.

Mutation is another genetic operator it randomly chooses one or more bits from the chromosome and flips them to get a new individual. Mutations introduce the necessary noise to perform hill climbing in genetic algorithms.

5.4 Applications of Evolutionary Algorithms

Evolutionary Algorithms have many applications in search and optimization. Some variants of EAs are clubbed with machine learning algorithms to find their optimal tuning parameters so as to find the best model. EAs can perform symbolic regression which are often very difficult for traditional regression algorithms.
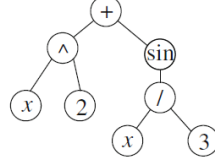
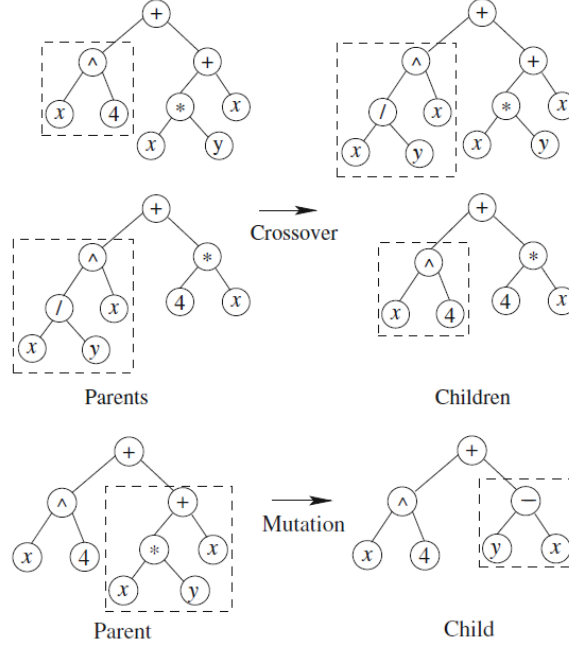Figure 5: Tree Encoding for $f(x) = x^2 + sin(x/3)$. Image taken from [Du and Swamy (2016)]

Figure 6: Crossover and Mutation in Symbolic Regressio. Image taken from [Du and Swamy (2016)]

5.4.1 Symbolic Regression with Genetic Algorithms

Genetic Programming is a variant of Evolutionary Algorithm which can be used for symbolic regression. The encoding used in genetic programming is in form of hierarchical trees. We can represent mathematical function with the help of these trees. See Figure 5. Now, all weed to perform symbolic regression is to define the crossover and mutation operators. See Figure 6.

5.4.2 Support Vector Regression with Genetic Algorithms

Chen and Wang (2007) have used Genetic Algorithms for tuning the parameters of the traditional Support Vector Regression algorithm to forecast tourism data. Neural Networks have been shown to work very well on this problem. However, they have some drawbacks such as large number of parameters to train, risk of overfitting and difficulty in obtaining a stable solution. Support Vector Regression is also known to work excellently for non-linear regression problems but it needs careful setting of its parameters to avoid bad performance and overfitting. But we don't have any general rule to tune these parameters, it comes down to knowledge of

the experimenter and trial and error methods for this.

This is the part where genetic algorithms come into picture, Chen and Wang (2007) applied GA to find the optimal parameters for SVR. Their results are at par with the Neural Network framework used for the tourism forecasting.

6 Conclusion

We have discussed some very beautiful algorithms inspired by nature which can be used to solve search and optimization problems. Also, the problems we discussed are of great practical importance, take Aircraft Conflict Resolution as an example and Job Shop Scheduling. Solving these problems can save lives on one hand and time and money on the other. Yet, this is not all, there is a whole world of such algorithms, algorithms which were designed by observing and modelling the miracles that nature does. Algorithms like Bee Hive Optimization, Cuckoo Search, Bacterial Foraging, Glow-worm Swarm Optimization, Biomolecular Computing and so on, the list is almost endless. All of them helping us to solve complex problems, mostly problems which are NP Hard.

References

Leonora Bianchi, Marco Dorigo, Luca Maria Gambardella, and Walter J Gutjahr. A survey on metaheuristics for stochastic combinatorial optimization. *Natural Computing*, 8(2):239–287, 2009.

Christian Blum and Michael Sampels. Ant colony optimization for fop shop scheduling: a case study on different pheromone representations. In *Evolutionary Computation, 2002. CEC'02. Proceedings of the 2002 Congress on*, volume 2, pages 1558–1563. IEEE, 2002.

Kuan-Yu Chen and Cheng-Hua Wang. Support vector regression with genetic algorithms in forecasting tourism demand. *Tourism Management*, 28(1):215–226, 2007.

Maurice Clerc. *Particle swarm optimization*, volume 93. John Wiley & Sons, 2010.

Marco Dorigo and Christian Blum. Ant colony optimization theory: A survey. *Theoretical computer science*, 344(2-3):243–278, 2005.

Marco Dorigo, Gianni Di Caro, and Luca M Gambardella. Ant algorithms for discrete optimization. *Artificial life*, 5(2):137–172, 1999.

Marco Dorigo, Marco Birattari, and Thomas Stutzle. Artificial ants as a computational intelligence technique. *IEEE computational intelligence magazine*, 1:28–39, 2006.

Ke-Lin Du and MNS Swamy. *Search and Optimization by Metaheuristics*. Springer, 2016.

Nicolas Durand and Jean-Marc Alliot. Ant colony optimization for air traffic conflict resolution. In *ATM Seminar 2009, 8th USA/Europe Air Traffic Management Research and Developpment Seminar*, 2009.

Julia Handl, Joshua Knowles, and Marco Dorigo. Ant-based clustering and topographic mapping. *Artificial life*, 12(1):35–62, 2006.

Scott Kirkpatrick, C Daniel Gelatt, Mario P Vecchi, et al. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

Keith E. Mathias and L. Darrell Whitley. Changing representations during search: A comparative study of delta coding. *Evol. Comput.*, 2(3):249–278, September 1994. ISSN 1063-6560. doi: 10.1162/evco.1994.2.3.249. URL `http://dx.doi.org/10.1162/evco.1994.2.3.249`.

F Pezzella, G Morganti, and G Ciaschetti. A genetic algorithm for the flexible job-shop scheduling problem. *Computers & Operations Research*, 35(10):3202–3212, 2008.

Riccardo Poli, James Kennedy, and Tim Blackwell. Particle swarm optimization. *Swarm intelligence*, 1(1):33–57, 2007.

Xinyu Shao, Weiqi Liu, Qiong Liu, and Chaoyong Zhang. Hybrid discrete particle swarm optimization for multi-objective flexible job-shop scheduling problem. *International Journal of Advanced Manufacturing Technology*, 67, 2013.

Sanjay K. Sharma and George W Irwin. Fuzzy coding of genetic algorithms. *IEEE Transactions on Evolutionary Computation*, 7(4):344–355, 2003.

Peter JM Van Laarhoven, Emile HL Aarts, and Jan Karel Lenstra. Job shop scheduling by simulated annealing. *Operations research*, 40(1):113–125, 1992.

Takeshi Yamada and Ryohei Nakano. Job shop scheduling. *IEE control Engineering series*, pages 134–134, 1997.