An Introduction to Evolutionary Computation

Lecture notes for Contemporary Intelligent Information Techniques in 2013

Akira Imada
Brest State Technical University, Belarus

(last modified on)

December 19, 2014

1 What does GA look like?

We now assume to solve a problem which includes n variables. That is, our task is to determine an optimal set of n variables. Then we design GA as follows.

1.1 Individual is represented by a chromosome

Represent a series of x_i as a *population* of strings. Each of these strings is refered to as *chromosome* or sometimes called *individual*.

Then we start an evolution as follows, expecting better solutions from generation to generation.

1. **(Initialization)** Generate an initial *population* of p individuals at random.
2. **(Fitness Evaluation)** Evaluate fitness of each chromosome and sort the chromosomes according to its fitness from the best to the worst.
3. **(Selection)** Select two chromosomes
   - Here, from the best half of the population at random, which is called a *Truncate Selection*.
4. **(Reproduction)** Produce a child by the following two operations:
   - *Uniform Crossover*, for example
   - *Mutation*
5. Create the next generation by repeating the steps from 3 to 4 n times.
6. Repeat the steps from 2 to 5 until (near) optimal solution is obtained.

1.2 How we select parents?

Hopefully, *the better the fitness the more likely to be selected.*

1.2.1 Threee different versions of Selection

• *Truncation Selection (Simplest of the three):* Select parents from the best some-percentage of the population.

• *Roulette Wheel Selection (or Fitness Proportionate Selection):*
With trancate selcetion, low fitness chromosomes have no chance to reproduce. In nature,

however, good children sometimes emerge from not good parents. So let's give a chance to a bad chromosomes to create children, though not with equal probability but proportionate to their fitness. This is called *fitness proportionete selection*, or *roulette wheel selection.*

Select such that the probability to be selected is proportional to the fitness value.

To be more specific, sort the individual from low to high and calculate cumulative value of fitness as follows: Then create a random number r from 0 to 1, and if $r < 0.3750$ then select #1, else if $r < 0.6250$ then select #2 , else if $r < 0.7500$ then select #3 and so on.

• *Tournament Selection* Assume we have the original μ parents and their μ children. The fitness value of each of the 2μ individuals are compared to those of q individuals which are chosen randomly from the whole 2μ points at every time of the comparison Then the 2μ individuals are ranked according to the number of wins, and the best μ individuals survive (q-tournament selection).

individual after sort	fitness	cumulative value of fitness
#1	0.3750	0.3750
#2	0.2500	0.6250
#3	0.1250	0.7500
#4	0.1250	0.8750
#5	0.0625	0.9375
#6	0.0625	1.0000

Table 1: Individuals are selected using random numbers from 0.0000 to 1.0000 according to the cumulative value of fitness after sorting the population.

Note that even the worst fitness individual have a chance to be selected under Roulette Selection however few it might be, while under Trancate Selection worse individual have no chance to be selected. Tornament Selection could also select a worse individual except for the worst q individuals. We can control the probability of selecting worse individual by changing q.

1.3 How parents produce a child?

1.3.1 Cross-over

So-called *crossover* is exproited for the reproduce children. Here, we see three different versions of crossover bellow.

1.3.2 Mutation

We should give a mutation to introduce new genes. This is to avoid for individuals in the population to be trapped into a *local minimum*. The probabability for mutation to occur is small — typically 1/number-of-genes. To be more specific,
If a randomly generated number from 0.0 to 1.0 is smaller than the mutation rate then mutate otherwise do nothing.

2 All One Problem

To feel how the evolutionary algorithm works, let's try a toy program. Assume we have 20 genes in one chromosome. Each of the genes corresponds to a trait of us such as blue or brown eyes. All genes are binary. Also assume 1 is a good gene and 0 is a bad one.

Then starting with a population of random chromosomes, observe the evolution. Fitness can be estimated the number of 1 in the chromosom.
An example of the chromosome is

1 0 0 1 0 1 1 1 0 0 1 0 1 0 0 0 0 1 1 0.

The number of '1' is the fitness of this chromosome.
If your program works properly, you will get the best chromosom

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

after a several generations.

Figure 2: The generation vs fitness curve in the case of 20 genes.

Figure 3: The generation vs fitness curve in the case of 800 genes.

3 Commonly Used Test Functions

Below we study two multi-dimensional test-functions whose location of the global minimum was already known. It is, therefore, useful to explore this test-function to learn how evolution works inside the PC.

3.1 Sphere model

3.1.1 Defined on 20-Dimensional space

Probably the simplest one is

$$y = x_1^2 + x_2^2 + x_3^2 + \cdots + x_{20}^2.$$

Or, equivalently,

$$y = \sum_{i=1}^{20} x_i^2 \tag{1}$$

which is difined, for example, on each $x_i \in [-5.12, 5.12] \quad (i = 1, 2, \cdots, 20)$. This function is an extention of well known $y = x^2$ to its 20-dimensional version. The unique global minimum is located on the Origin and no local minimum. Hence, this might be a good start to try a GA.

An example of chromosome is

3.2 4.4 -2.1 0.5 -3.8 -2.5 1.7 5.1 -0.3 -2.1 -3.8 5.0 0.4 4.2 -5.0 -1.3 3.3 4.0 -1.4 -3.9

By translating the i-th gene into x_i we will get all the value of $x_1, \cdots x_{20}$, and we can calculate the value of y. This is the *fitness* of this chromosome.

3.1.2 Defined on 1-dimensional space

To be able to imagine let's think its 2-dimensional version.

$$y = x^2$$

Then the graph is a well-know *parabola*. See Figrue. 4.

3.2 Rastrigin Function

3.2.1 Defined on 20-Dimensional space

Now we observe a so-called the Rastrigin Function, which resembles a parabora but has many local minima wevey where.

$$y = \{x_i^2 - \cos(2\pi x_1)\} + \{x_2^2 - \cos(2\pi x_2)\} + \cdots + \{x_{20}^2 - \cos(2\pi x_{20}\}).$$

Or, equivalently,

Figure 4: A two-dimensional version of the sphere model.

$$y = nA + \sum_{i=1}^{n} (x_i^2 - A\cos(2\pi x_i)), \quad x_i \in [-5.12 - 5.12].$$

Ruggedness might be controlled by modifying the value of A.

3.2.2 Defined on 1-dimensional space

A two dimensional example ($n = 1$):

$$y = 3 + x^2 - 3\cos(2\pi x).$$

Figure 5: A 2-D version of Rastrigin function

4 Neural Network to solve even-parity problem

Here we determine a configuration of weight values of a neural network. Now think of a neural network to solve the *even-N-parity* Boolean function, that is, iff the number of "1" is even the output should be "1" otherwise "0". Assuming N-N-1 structure of feed forward neural network, we should determine $N^2 + N$ weight values so that any binary input results in the proper output.

Our chromosome in this case has $N^2 + N$ genes. Fitness can be evaluated by giving each of the neural net work corresponding to the chromosome all the possible 2^N query and counting the number of its correct output.

Figure 6: A 5-5-1 structure of Feedforward neural network

Now we see the set of input and output of the even-5-parity Boolean function.

input	output
00000	1
00001	0
00010	0
00011	1
00100	0
00101	1
00110	1
00111	0
01000	0
...	...
11110	1
11111	0

Table 7: Input and output of even-5-parity Boolean function.

5 Traveling Salse-person Problem

Asuuming N cities all of whose cordinate are given, the Traveling Salse-person Problem (TSP) is a problem in which a sales-person should visit all of these cities once but only once and objective is to look for the shortest tour.

Assume we start from a city and return to the city after visiting all the other cities only once. When the number of cities is N, the number of all the possible routes is $(N-1)!/2$ if we don't count the route with exactly opposite order of the cities of the route already count.

Let's try a TSP with 4 cities created at random in x-y cordinate where $0 < x, y < 10$. Then we calculate the distances between all the possible combinations of two cities, which can be shown by a distnace matrix. As the number of possible routes is just $3!/2 = 3$ we can directly calculate the distance every such routes. See an example in Figure 9.

Figure 8: An example of four cities. Map, distance matrix, and all the possible routes with total distance of each route .

Problem is, the number of possible routes will explode as the number of cities N increases. How many routes do you guess when $N = 1000$ for example? Now we apply a genetic algorithm to this problem[1].

[1]Here we explore the TSP by an GA. But there seems to be more direct way of computation called Ant Colony Optimization (ACO). ACO is an optimization technique we borrowed the metaphore of an intelligence of ant society as their collective behaviour. Ants are good at seeking a shortest path from their nest to a food when one of their colleagues finds it he communicates with others by using a chemical called pheromon. If we have a time we will study ACO later.

5.1 To solve TSP by a Gentic Algorithm

5.1.1 How to encode for a crossover to be valid?

First, we have to design our chromosome representing a tour? If we represent a candidate solution with a list of cities to be visited in order, such as the chromosome of the tour A-C-F-D-G-E-B is

$$(ACFDGEB) \tag{2}$$

Are the the results of crossover and mutation feasible? The answer is No! For example, the possible child of two parent $(ACFDGEB)$ and $(AGBFCDE)$ by *one-point-crossover* could be $(ACFFCDE)$ and this is not a feasible tour because C and F are visited twice and B is not visited. Or, if we give a standard mutation to $(ACFDGEB)$, for example, by replacing 4th gene with other randomly chosen city, such as $(ACFAGEB)$, which is not a feasible tour either.

Then, in order for the result of crossover and mutation to be feasible what representations are possible?

One idea is:

Algorithm 1 (Traveling Salesperson Problem) *Encoding a legitimate route to a chromosome made up of any integer.*

Step-1. Set $i = 1$.
Step-2. If i*-th gene is* n *then* n*-th city in the list is the city to be currently visited.*
Step-3. Remove the city from the list.
Step-4. Set $i = i + 1$ *and repeat Step-2 to Step-4 while* $i \leq n$.

For example, when the list of cities is

{A, B, C, D, E, F, G, H, I }

chromosome: (112141311) is the tour:

A-B-D-C-H-E-I-F-G.

Try some one-point crossover on two parents (112141311) and (515553321).

5.1.2 Mutation

Then next, how we design mutation? How about, for example, specifying two points at random and reverse the gene order?

5.1.3 When to stop a run?

Finally, when, on earth, we stop a run? We don't know the optimum. The answer is let's observe the evolution of fitness. We can expect the run converges to the optimum or near-optimum.

Figure 9: A possible mutation in Traveling Salesperson Problem.

Figure 10: An example of fourteen cities and its one option of tour.

5.1.4 TSP with a huge number of cities

I found that 13,509 real cities in USA are given with their cordinates in the web-page http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95. Why don't we try this very challenging problem by ourselves. The plotted these cities are shown in Figure 3.

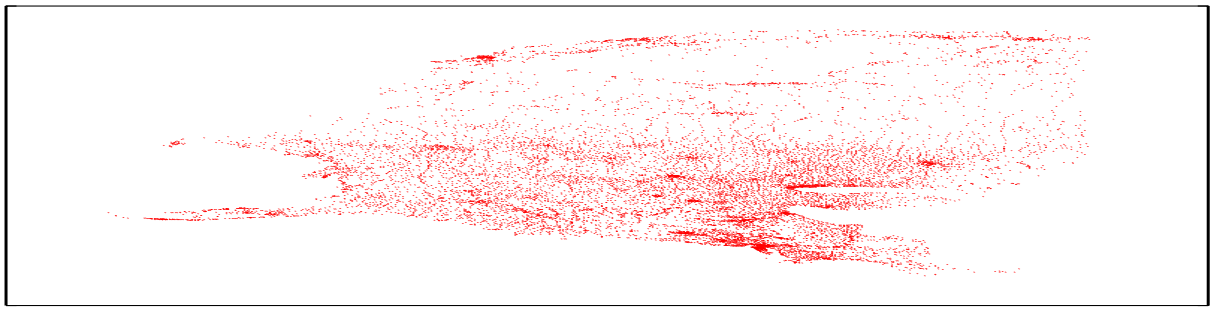


Figure 11: An example of 13509 real cities' location in US. Plotted with the data taken from http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95.

6 Robot Navigation in GridWorld

Assume now that we want to make an agent, or a robot, in a gridworld, a possible chromosome can be made up of integer gene from 1 to 4 where 1, 2, 3, and 4 correspond to one cell movement of the agent to north, south, east and west. Take a look at the below as an example.

Our chromosome is made up of 4 different genes move (i) up, (ii) down, (iii) to right , and (iV) to left. See an example below.

Figure 12: An example of a chromosome and the path represented by it.

We now take a look at two such problems more specifically in the next subsection.

6.1 Exploration of a gridworld with a limited energy

Search for a path of minimum Manhattan distance

.

Search for a path to the start point after a maximum exploration

It would be not so difficult for us human to find such a route. But how about a computer? The question now would be, "How we designed fitness function?"

We will return this topics of "multiple fitness functions later.

6.2 Jeep Problem – From "A camel in a Desert" to Landrover in the Mars"

Assume we have a Jeep at the base which locates at the edge of a desert. The Jeep has a tank which can be filled with a maximum of one-unit of gasoline. With one unit

Figure 13: In the grid-world of 96 starting from (24,24) a robot walks aiming the goal at (72,72) of which the robot had no *a-priori* information. Left: The path of minimum length among 100 trials by random walk. Right: Minimal path the robot found after an evolutionary learning as shown in Fig. 3. (Marginal area is omitted.)

of gasoline, the jeep can move one unit of distance. The Jeep can only fill gasoline at the base.The jeep can carry containers to put its gasoline on the desert for a future use, Assume the tour should be on the strait line in the desert. [2]

The question is "How far the jeep can penetrate in to the desert on the straight road when n unit of gasoline is available at the base.

For example when $n = 2$, the best strategy is to start the base with one unit of gasoline in its tank and go 1/3 unit distance (it has spent 1/3 unit of gasoline to reach the point, then put 1/3 unit of gasoline in the container there (now jeep has 1/3 unit of gasoline in the tank) and go back to the base. Exactly when the jeep arrive to base all gasoline filled at the start was spent. Then geep fill the 2nd unit of gasoline given, go 1/3 unit fill the gasoline he had put before and the tank is again full, then go forward until all the gasoline in the tank will be spent. Therefore the maximum distance the jeep can go is 4/3 unit of distance.

Guess the maximum distance when $n = 2$. We already know the maximum distance with n unit ofgasoline is D_n is expressed as the recursive relation $D_n = D_{n-1} + 1/(2n - 1)$.

Our interest is on whether an evolutionary computation can find a almost best strategy, say, $n = 5$ in which maximum distance is 1323/945=1.4. (If my calcuration is correct. Try it by yourself).

[2]The problem first appeared as *"a camel carrying grain in a desert"* as the 52nd problem in the *"Propositions ad acuendos inventes"* (in Latin) attributed to Alcuin of York (around in B.C. 732–804). And now a jeep in a dessert, further, landrver in the Mars.

7 Evolving Strategy — Iterated Prisonner's Dilemma

So, *To be or not to be? — That is the question.* Not only Shakespere but in many literarture works *dilemma* is their theme. The Opera "Tosca" by Puccini is one of those typical examples.[3]

7.1 When dilemma happens?

Assume n person are in the following game. Each of these n pearson is in the separate booth, where communication would not be available and not visible with each other. In each of those boothes button are fascilitated. You all are in the booth for one minute. If no one does not push the button, all of you will be given 100$ each. If, on the other hand, someone push the button, or more people do so, the first one who push the button will be given 10$ and other will not be given any money. What would you do, if you were one of these n people?

7.1.1 Condition to be a dilemma

What if the money given in case all do not touch button is 10 dolor, and otherwise the first person who push the button will be given 100 dolor? In this case no dilemma will be arosen. Push the button immediately without hesitation.

7.2 Prisoner's Dilemma

In the community of *Game Theory* we have the problem called Prisoner's Dilemma[4] which is formulated as follows.

Problem (Prisoner's Dilemma) *Two newly arrested prisoners A and B are offered a deal:*

[3]Matt Ridley once wrote in his book, "The Origin of Virtue – Human Instincts and the Evolution of Cooperation." Penguin Books (1996) about this opera. It reads: *In Puccini's opera Tosca, the heroine is faced with a terrible dilemma. Her lover Cavaradossi has been condemned to death by Scarpia, the police chif, but Scarpia has offered her a deal. If Tosca will sleep with him, he will save her lover's life by telling the firing squad to use blanks. Tosca decides to deceive Scarpia by agreeing to his request, but then stabbing him dead after he has given the order to use blanks. She does so, but too late discovers that Scarpia chose to deceive her too. The firing squad does not use blanks: Cavaradossi dies. Tosca commits suicide, and all three end up dead.* The book is regarding a Game Theory. The author continues: *Though they did not put it this way, Tosca and Scarpia were playing a game, indeed the most famous game in all of game theory, an esoteric branch of mathematics that provides a strange bridge between biology and economics. The game has been central to one of the most exiting scientific discoveries of recent years: nothing less than an understanding of why people arenice to each other. Moreover, Tosca and Scarpia each played the game in the way that game theory predicts they should, despite the disastrous outcome for each. How can this be?*

[4]Proposed by Merrill Flood and Melvin Dresher in the 1950's

- *If A confesses and B does not, A will be reliesed and B will get 5 years in jail, and vice versa.*
- *If both confess, then both will get 4 years in jail.*
- *If both do not they will each get 2 years.*

Cleary, "0 year in prison" would be the best reward from the personal point of view. If we think of the both prisoners' benefit, then the case "both in jail for 2 years" would be better than the case "both in jail for 4 years" unless you expected the prison as a "free hotel."

7.2.1 Condition to be a dilemma

Now let me quote descriptions from a website[5] where the parameter's is as follows: R is a Reward for mutual cooperation. Therefore, if both players cooperate then both receive a reward R. (e.g., 3 points). If one player defects and the other cooperates then one player who defects receives T - the Temptation to defect - (e.g. 5 points), and the other player who cooperates receives M - the Minimum payoff (e.g., zero). If both players defect then they both receive P - the Punishment for mutual defection (e.g., 1).

If one player defects and the other cooperates then one player receives the Temptation to defect payoff (5 in this case) and the other player (the cooperator) receives the Sucker payoff (zero in this case). If both players defect then they both receive the Punishment for mutual defection payoff (1 in this case).

The question arises: what should you do in such a game?

Suppose you think the other player will cooperate. If you cooperate then you will receive a payoff of 3 for mutual cooperation. If you defect then you will receive a payoff of 5 for the Temptation to Defect payoff. Therefore, if you think the other player will cooperate then you should defect, to give you a payoff of 5.

But what if you think the other player will defect? If you cooperate, then you get the Sucker payoff of zero. If you defect then you would both receive the Punishment for Mutual Defection of 1 point. Therefore, if you think the other player will defect, you should defect as well.

So, you should defect, no matter what option your opponent chooses.

Of course, the same logic holds for your opponent. And, if you both defect you receive a payoff of 1 each, whereas, the better outcome would have been mutual cooperation with a payoff of 3. The choices of an individual is less than that could have been achieved by two cooperating players, thus the dilemma and the research challenge of finding strategies that promote mutual cooperation.

[5] http://www.prisoners-dilemma.com/whatisit.html. Author's name is not visible in the page.

In defining a prisoners dilemma, certain conditions have to hold. The values we used above, to demonstrate the game, are not the only values that could have been used, but they do adhere to the conditions listed below.

Firstly, the order of the payoffs is important. The best a player can do is T (temptation to defect). The worst a player can do is to get the sucker payoff, S. If the two players cooperate then the reward for that mutual cooperation, R, should be better than the punishment for mutual defection, P. Therefore, the following must hold.

$$T > R > P > S$$

Secondly, players should not be allowed to get out of the dilemma by taking it in turns to exploit each other. Or, to be a little more pedantic, the players should not play the game so that they end up with half the time being exploited and the other half of the time exploiting their opponent. In other words, an even chance of being exploited or doing the exploiting is not as good an outcome as both players mutually cooperating. Therefore, the reward for mutual cooperation should be greater than the average of the payoff for the temptation and the sucker. That is, the following must hold.

$$R > (S + T)/2$$

However dilemma is not so serious like Hamlet's "*To be or not to be? That is a problem.*" It's better to always confess.

If the game is to be iterated, on the other hand, we have to see the game differently. Like any negotiation it's likely to have a dilemma – corporate or betray?

7.2.2 Iterated Prisoner's Dilemma

The next question then is how about if the game is repeated? This is called the *Iterated Prisoner's Dilemma (IPD)*. In this case *strategy* to get a higher award as a result arises: What would be the optimal next action? For example, *Always Defect* strategy, or *Tit-for-Tat* strategy where the player cooperates on the first play, and afterward the same action as the opponent in the previous game.

Here, strategy determins the next action based on three previous moves of the two players in a raw. Number of all possible previous three games is $2^6 = 64$ — 64 combination of Cooperate and Defect. Namely, all those possible combinations of 6 previous moves can be represented with a 64 bit binary chromosome. For example, if a history of 6 previous actions of opponent and the player is C-d-D-d-C-d we express the history by the binary number 100010 where "C" and "c" are expressed by 1 and "D" and "d" are expressed by 0 and uppercase "C" and "D" are opponent's and lowercase "c" and "d" is playere's Cooporation and Defection, respectively.

Then the next action when the hisory was (000000) is put on the 1st bit with 0 being defection and 1 being cooperation. The same is repeated, that is, the next action when

the history was (000001) is put on the 2nd bit and so on. No need to repeat but, the 3rd bit is the action after the history of (000010) and the 64-th bit, the last bit, is when the history was (111111) Thus we can represent a strategy with a 64-bit binary chromosome.

Then each individual (chromosome) competes with each of other randomly chose p individuals, and the number of it wins against others is counted. This number of wins is the fitness value of the individual, which is called a p-tournament selection.

7.2.3 An example of chromosome

Assume now, for example, the game is

A	$\cdots$	0	1	1	Now my turn. Which should I choose 1 or 0?
B	$\cdots$	0	1	0	

The history, in this case is

$$0, 1, 1, 1, 0, 0$$

So, this binary 011100 is translated into decimal 28.

Now if A's chromosome is

00101011110010101010100110101001010111010000111101110000111111101

the action should be the 28th bit from the lowest (right most) bit, that is, 1. Hence corporate.

Figure 14: A two-dimensional version of the Schwefel's Function's graph.

7.2.4 Tournament selection

Now assume the number of population is N (e.g., 40). Each individual choose T (e.g., 5) other individuals to play a game from the one to the next. So fitness of the individual is the number of win, or the total amount of reward points during these T games.

8 Sorting Network – How many minimum comparisons are necessary?

Which is clever? – Human or Computer?

When we write an algorithm, we often come accross a necessity to sort a set of items in the order of some criteria. How, for example, do we create codes for sorting 16 integer inputs in ascending order? We pick up 2 items from one item to the next, compare, and swap them if the order is not the preferable one

Algorithm 2 (A Sorting Algorithm) *Now we assume to sort N numerical items from the smallest to the largest.*

- *For i = 1 to N-1*
  - ⋆ *For j = i+1 to N*
    - · *If item(i) < item (j) Then swap item(i) and item(j)*

Now let us represent the sorting above by a graphics in the following way.

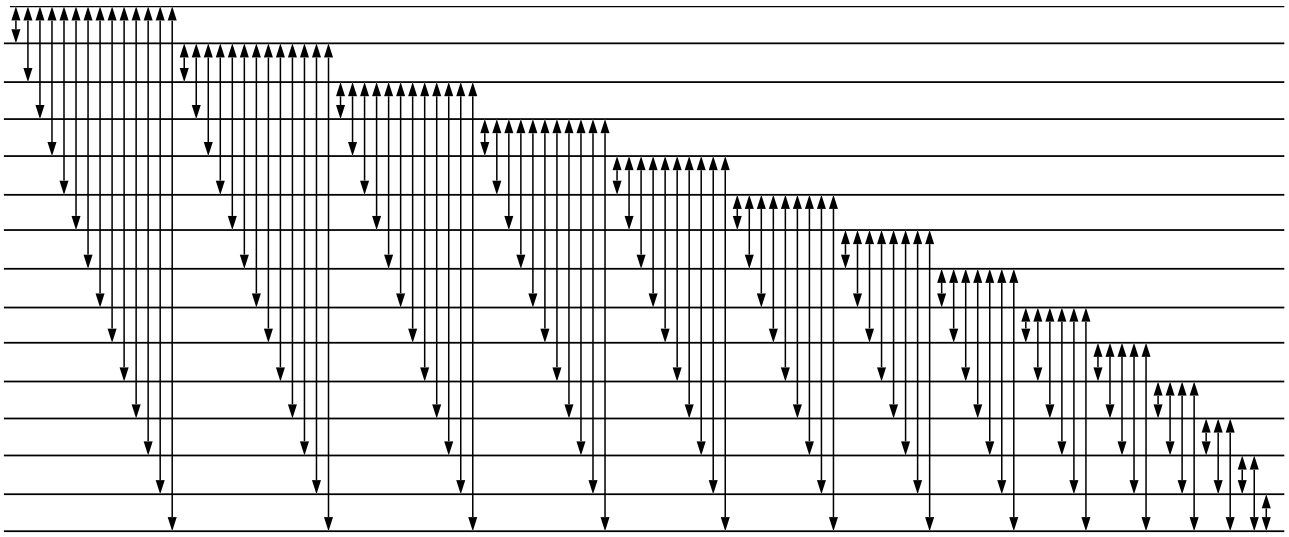


Figure 15: A typical network for sorting, not very efficient though.

The total number of comparison in this case is 120, though this is not a very efficient way. Then problem is what is the minimum number of comparisons with which any arbitrary set of 16 inputs are correctly sorted. That is,

Problem (Sorting Network) *The task is to sort n items. For the purpose, the i-th and j-th element are compared and swap if necessary. and the goal is to find an algorithm which correctly sorts all n items with the minimum number of comparisons.*

It might be interesting to overview a little history of this topic. In 1960's, in a comunity of computer algorithms, there was a competision of what is the minimum number of comparisons when, say, $(n = 16)$? The result was

- ⋆ 65 comparisons Bose and Nelson (1962).
- ⋆ 63 by Batcher and by Floyd and Knuth (1964).
- ⋆ 62 by Shapiro (1969)
- ⋆ 60 by Green (1969)

See the Figure bellow.

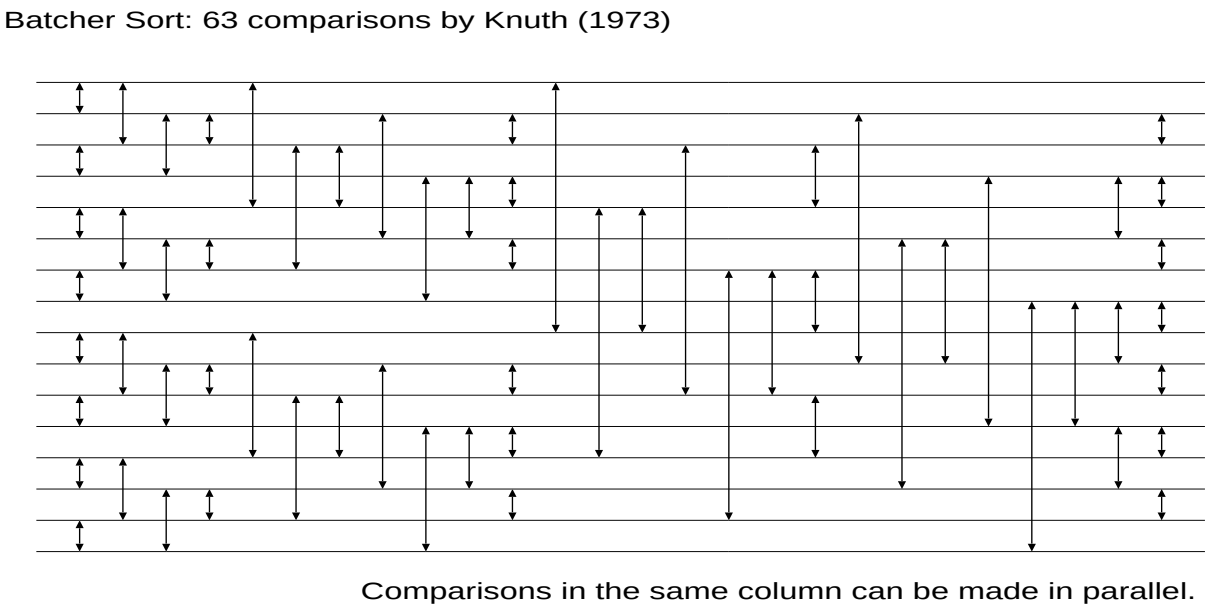


Figure 16: Batcher's proposition of sorting network with 63 comprisons (1964)

Up to now, however, we have had no proof for this optimality. Then let's make an Evolutionary Computation surch for this nimimum number. Would it work better than by human? Hillis (1992) challenged this. Hillis's innovations was that he employed *Diploidy Chromosome* as follows. We will show this more in detail later. Here, we show a simple version of GA implementation.

Now assume one chromosome corresponding to one sorting network is made up of 140 genes each of which takes an integer value from 01 to 16 permitting overlaps, such as

(12 01 05 04 16 12 04 14 01 02 06 ..... 07 15 08 10)

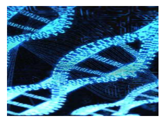
where an odd number gene and the next righthand side even number genes represent a comparison. In the example above

12 <=> 01; 05 <=> 04; 16 <=> 12; .....; 08 <=> 10)

Then one chromosome include 70 comparisons at most. Why at most? Because it could include a same comparison multiple time. Hence, the minimum number of comparison is one, which is very unlikly though.

9 Sorting Network Revisited

9.1 Let's be more biological - Exploitation of Diploidy Chromosomes



(1001 1000 1010 1101 1110 0100 1110 0011) (1001 1000 1010 1101 1110 0100 1110 0011)	(1001 1000 1010 1101 1110 0100 1110 0011) (1001 1000 1010 1101 1110 0100 1110 0011)	(1001 1000 1010 1101 1110 0100 1110 0011) (1001 1000 1010 1101 1110 0100 1110 0011)
(1001 1000 1010 1101 1110 0100 1110 0011) (1001 1000 1010 1101 1110 0100 1110 0011)	(1001 1000 1010 1101 1110 0100 1110 0011) (1001 1000 1010 1101 1110 0100 1110 0011)	(1001 1000 1010 1101 1110 0100 1110 0011) (1001 1000 1010 1101 1110 0100 1110 0011)
(1001 1000 1010 1101 1110 0100 1110 0011) (1001 1000 1010 1101 1110 0100 1110 0011)	(1001 1000 1010 1101 1110 0100 1110 0011) (1001 1000 1010 1101 1110 0100 1110 0011)	(1001 1000 1010 1101 1110 0100 1110 0011) (1001 1000 1010 1101 1110 0100 1110 0011)
(1001 1000 1010 1101 1110 0100 1110 0011) (1001 1000 1010 1101 1110 0100 1110 0011)	(1001 1000 1010 1101 1110 0100 1110 0011) (1001 1000 1010 1101 1110 0100 1110 0011)	(1001 1000 1010 1101 1110 0100 1110 0011) (1001 1000 1010 1101 1110 0100 1110 0011)
(1001 1000 1010 1101 1110 0100 1110 0011) (1001 1000 1010 1101 1110 0100 1110 0011)	(1001 1000 1010 1101 1110 0100 1110 0011) (1001 1000 1010 1101 1110 0100 1110 0011)	(1001 1000 1010 1101 1110 0100 1110 0011) (1001 1000 1010 1101 1110 0100 1110 0011)

Figure 17: An example of Hillis's set of diploidy chromosomes.

- Each individual consists of 15 pairs of 32-bit chromosomes.
- Each chromosome consists of eight 4-bit strings (called *codons*).

(0001 0010 0101 1000 0000 0100 1111 1001)
(0011 0100 0101 1000 1101 1100 1111 1001)

- Each codon represents an integer between 0 and 15 indicating which item is to be compared out of 16 items. That is, the above example is interpreted as.

(01 02 05 08 00 04 15 09)
(03 04 05 08 13 12 15 09)

- Each adjacent pair of codons in a chromosome specifies a comparison between two elements. Thus each chromosome encodes four comparisons, e.g.,

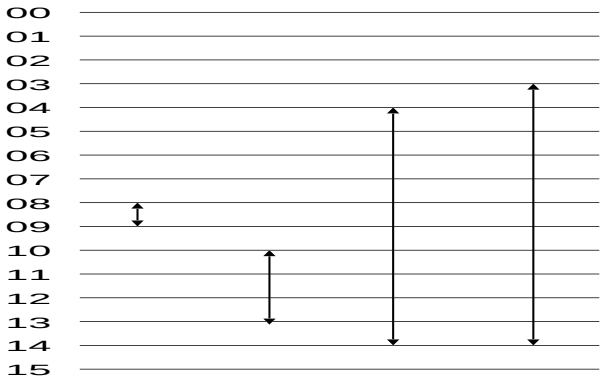


Figure 18: Four comparisons specified by the chromosome (09 08 10 13 14 04 14 03).

(09 08 10 13 14 04 14 03)

indicates the four comparisons below.

- The chromosome pairs is read off from left to right.
- If these adjacent two codons are identical at the same corresponding two positions of the chromosome pair – this is called *homozygous* – then only one pair of numbers is inserted in the phenotype. If it encodes different pairs of numbers – *heterozygous* – then both pairs are inserted in the phenotype. So in the previous example:

(01 02 05 08 00 04 15 09)
(03 04 05 08 13 12 15 09)

means the following six comparisons:

01⇔02, 03⇔04, 05⇔08, 00⇔04, 13⇔12, 15⇔09

- Thus 15 pairs of chromosomes produce a phenotype with 60-120 comparisons. The more homozygous positions, the fewer comparisons.
- When two individuals are selected, one-point-crossover takes place within each chromosome pair inside each individual.
- For each of the 15 chromosome pairs, a crossover point is chosen at random and a single chromosome (called *gamete*) is formed.
- Thus 15 gametes from each parent are created.
- Each of the 15 gametes from the first parent is then paired with the gamete of the corresponding position from the second parent to form one offspring.

9.2 Pressure to homozygousity

Homezygous pair is more likely to survive than heterozygous pair, that is, two genes at the same location in a pair of chromosome will be more likely to the same one after evolution.

For example, the probability of (1, 1) pair to be (1, 1) is 1/2, while the probability of (1, 0) pair to be (1, 0) is 1/4. The former is calculated as

$$1 \times ((1/4) \times (1/2) + (1/4) \times (1/2) + (1/4) \times 0),$$

while the latter as

$$(1/2) \times ((1/4) \times (1/2) + (1/4) \times (1/2) + (1/4) \times 0).$$

Hence, we can expect a more homozygous gene pair after a longer evolution, If, in our context, all the pair become to have the same chromosome, it implies the number of comparison is 60.

10 Visualization of high-dimensional space

Visualization of data in a high-dimensional space is important. Maybe you have already learned such methds like *Kohonen's Self Organizing Map (SOM)* or *Principal Components Analysis (ICA).*

10.1 Why we need to reduce the dimension?

We, human, couldn't imagine the world of more than 3-dimensional space. In many scientic field, however, it is crucially important to grasp an image in high dimensional space. This is not only in scientic fields but also in real world around us.

Let me show an example. We now assume to assign newly employed soldires to appropriate mission according to their examination, say, of Mathematics and English.

	A	B	C	D	E	F	G	H	I	J	K	L	M
Mathematics	95	32	89	52	12	20	3	99	42	91	26	95	60
English	92	90	21	48	14	5	11	97	50	92	89	13	55

Table 19: A fictitious result of 2 examinations given to newly employed soldires.

The task of this classifiyng soldiers will be easier if you visualize the data.

Figure 20: A visualization. It's easy to classify soldires into 5 groups.

What if, then, we have one additional score for each soldier, say, phisical examination. In order to make a task like an espionage, it would be better to have a strong phisical capability. In this situation we have to classify them with 3-dimensional data, or on 3-dimensional space if we want make it like the above mentioned 2-dimansuional case. Moreover, to be more practical, assume we have a set of scores of 10 different examinations. In this case, we cannot visualize any more in a usual sence.

So, visualization of high-dimensional space, or dimension reduction technique is very important topic, and so far many such techniques has been proposed, among which Kohonen's Self Organizing Map is very popular above all.

10.2 Sammon Mapping by GA

Here we learn about Sammon Mapping. Sammon Mapping is a mapping a set of points a in high-dimensional space to the 2-dimensional space with the distance relation being preserved as much as possible, or equivalently, the distances in the n-dimensional space are approximated by distances in the 2-dimensional distance with a minimal error.

This method was proposed in 1980's as an optimization problem to which they approached by Operations Research technique suchas *Steepest Descend*, which is not so simple. Here, on the other hand, we employ Evolutionary Computatins which is quite simple. Let's see now what is the original Sammon Mapping look like.

Algorithm (Sammon Mapping)

1. *Assume N points are given in the n-D space.*
2. *Calculate distance matrix R ($N \times N$) whose i-j element is the Euclidean distance between the i-th and j-th point.*
3. *Also think of a tentative N points in the 2-D space that are located at random at the beginning.*
4. *The distance matrix Q is calculated in the same way as R.*
5. *Then the error matrix $P = R - Q$ is defined.*
6. *Search for the locations of N points in the 2-D space that minimizes the sum of element P.*

This is an optimization problem which we now can solve quite simply by using EC. That is, *by creating N points in 2-D space each of which corresponding N points in the n-D space with the distance relation being preserved as much as possible, or equivalently, such that the n-D distances are approximated by 2-D distances with a minimal error.*

In an actual GA implementation of Sammon Mapping, chromosomes might be made up of n genes each of which corrisonds to $x-y$ coordinate of a candidate solution of n optimally distributed points in 2-dimensional space. Uniform crossover is employed and from time to time mutation is given by replacing one gene with other random $x-y$ coordinate. See the Figure 2. See also the Figure bellow.
Examples in $49^2 = 2401$ dimensional space:

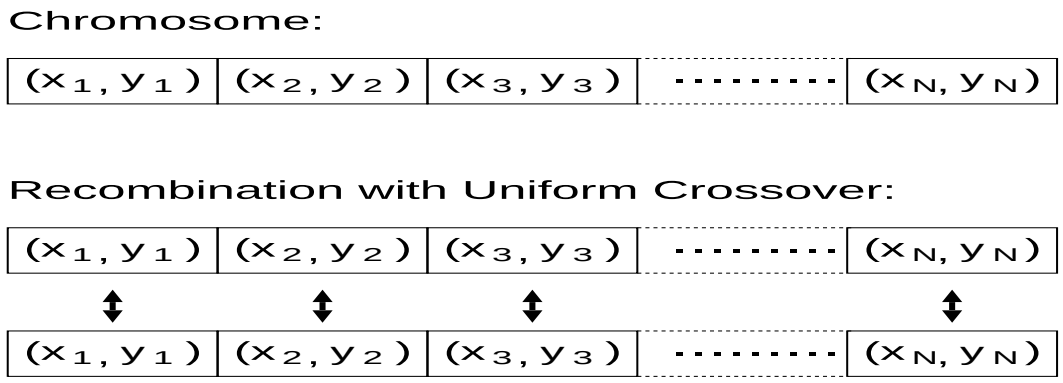


Figure 21: A chromosome representation and uniform crossover

Figure 22: Six Examples of Mapping from 2401-dimensional space to the 2-dimensional space. Further explanations are shown in the text.

11 Multi Modal Problem

What if we have multiple meaningful different solutions?

In Traveling Salesperson Problem, we are interested in the result of minimum path, even if we have multiple possible paths. However, we sometimes are interested in all of possible solutions at a time in a run. For example, when we want a set of fuzzy rules for designing a fuzzy controller. The topic of this section is regarding this problem. Let's start with a simple mathematical functions.

11.1 Yet another Testfunction

A 2-D function but multi peaks

The question is how we design our chromosomes. In the multi-dimensional function $y = f(x_1, x_2, \cdots, x_n)$ our genes might be continuos value each of which corresponds to the independent variable x_i $(i = 1, 2, \cdots n)$, that is, our chromosomes are made up of n genes. On the other hand how should we design our chromosomes when the number of independent variable is only one. A chromosome with only one gene? How we crossover two parents?

O.K. we usually use binary chromosome in this situation. Any (decimal) real-valued variable $x_i \in [a, b]$ could be encoded by n-bit binary strings where a and b is represented by $(00\cdots0)$ and $(11\cdots1)$, respectivley, and therfore accuracy (or granularity) is $(b - a)/(2^n - 1)$. For example, if our concern is the above $x \in [0, 1]$ then 10 bit of binary strings from 0000000000 to 1111111111 are expresses decimals with the precision of 1/1024. Or you might use and compare *Gray Code* where gray-code $a_1a_2\cdots a_n$ is tlanslated from binary number $b_1b_2\cdots b_n$ as

$$a_i = \begin{cases} b_i & \text{if} \quad i = 1 \\ b_{i-1} \oplus b_i & \text{otherwise} \end{cases} \tag{3}$$

where $\oplus$ is addition modulo 2. In gray code a pair of adjusent decimals are different only with Hamming distance 1, while in the standard binary encode this does not hold.

The test-functions we studied in Section. 3 are what evolutionary computations are especially good at, since we can treat high dimensional function whatever large it may be, simply by setting the number of genes in a chromosome to the dimensionality.

Then what should we do, if we are interested in a 2-D function? For example,

$$y = \sin^6(5\pi x) \tag{4}$$

or

$$y = -2((x - 0.2)/0.8)^2 \sin^6(5\pi x) \tag{5}$$

are interesting functions in order for us to observe how randomly created chromosomes in the 1st generation evolve to find peaks. See Figure 2.

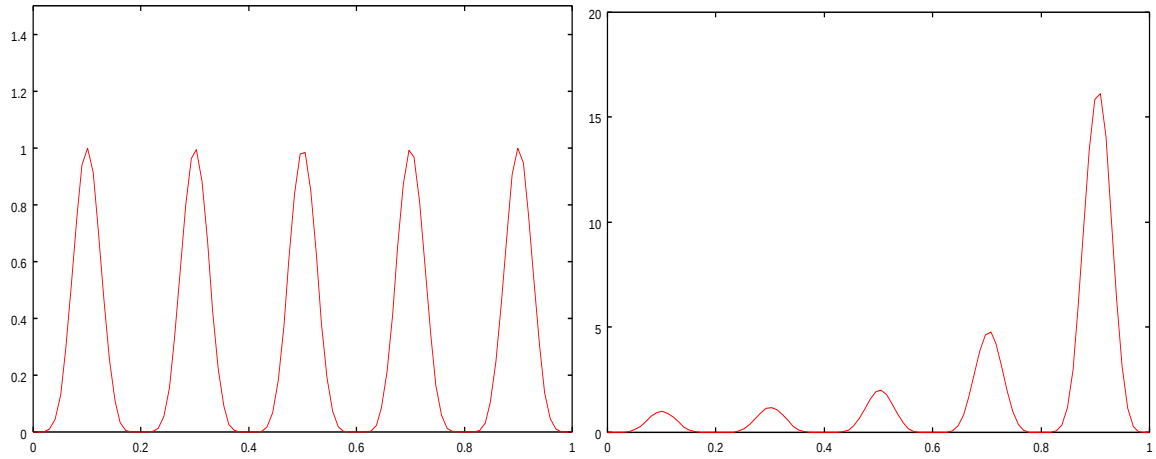


Figure 23: A multi-peak 2-D function and its variation

11.2 Multi-modal Optimization.

Sometimes we have multiple solutions. But EC usually converges only one solution out of them. Hence, to get those multiple solution we run the algorithm multiple time. Here we learn the technique in which individual construct niches and each species found a different solution at a run.

For the purpose, multiple *species* will be created and maintained in a population. These species independently search for a peak (hopefully an optimum solution), construct their *niche* and stay around the peak during a run.

In comparatively early days, essentially the following three methods were proposed. So-far proposed methods are

- Fitness sharing (Goldberg & Richardson, 1987)
  - · Similar individuals share fitness with each other.
- Crowding (De Jong, 1975)
  - · Similar individuals are replaced with random individuals
- Species Method.
  - · Mating is restricted to among similar individuals.

These days, IMHO, the following two are popular among others.

- Detterministic Crowding (Mahfoud, 1992)
- Sequential Niching

Let's see some of the aspects more in detail.

Fitness Sharing Fitness of each individual is derated by an amount related to the number of similar individuals in the population. That is, shared fitness $F_s(i)$ of the individual i is

$$F_s(i) = \frac{F(i)}{\sum_{j=1}^{\mu} s(d_{ij})}$$

where $F(i)$ is fitness of individual i; d_{ij} is distance between individual i and j; Typically d_{ij} is *Hamming distance* if in *genotypic space Euclidean distance* if in *phenotypic space* and $s(\cdot)$ is called *sharing function* and defined as:

$$s(d_{ij}) = \begin{cases} 1 - (d_{ij}/\sigma_{\text{share}})^{\alpha} & \text{if } d_{ij} < \sigma_{\text{share}} \\ 0 & \text{otherwise} \end{cases}$$

where σ_{share} is interpreted as size of niche, and α determines the shape of the function. The denominator is called *niche count*. You see shape dependency of $s(d_{ij})$ on α in Figure 11.2.

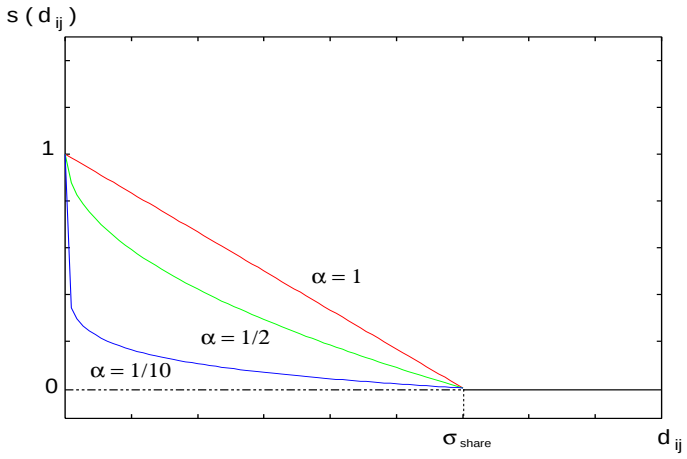


Figure 24: A shape dependency of $s(d_{ij})$ on α.

To be short (not so short though): Similar individual should share fitness. The number of individuals that can stay around any one of peaks (niche) is limited.
The number of individuals stay near any peak will theoretically be proportional to the *hight* of the peak

Deterministic Crowding: If the parents will be replaced or not with their childeren will be determined under a criteria of the distance between parents and children,

Figure 25: A typical two cases of distance between parents and children.

Algorithm Assuming crossover, mutation and fitness function are already defiened

1. *Choose two parents,* p_1 *and* p_2*, at random, with no parent being chosen more than once.*
2. *Produce two children,* c_1' *and* c_2'*.*
3. *Mutate the children yielding* c_1 *and* c_2*, with a crossover.*
4. *Replace parent with child as follows:*

   - *IF* $d(p_1, c_1) + d(p_2, c_2) > d(p_1, c_2) + d(p_2, c_1)$
     - *IF* $f(c_1) > f(p_1)$ *THEN* *replace* p_1 *with* c_1
     - *IF* $f(c_2) > f(p_2)$ *THEN* *replace* p_2 *with* c_2
   - *ELSE*
     - *IF* $f(c_2) > f(p_1)$ *THEN* *replace* p_1 *with* c_2
     - *IF* $f(c_1) > f(p_2)$ *THEN* *replace* p_2 *with* c_1

where $d(\zeta_1, \zeta_2)$ *is the Hamming distance between two points* (ζ_1, ζ_2) *in pattern configuration space. The process of producing child is repeated until all the population have taken part in the process. Then the cycle of reconstructing a new population and restarting the search is repeated until all the global optima are found or a set maximum number of generation has been reached.*

Sequential Niche: Single run is repeated sequentially, keeping the best individual at each run.

Algorithm

1. *Define niche radius* r*.*
2. *Define modified-fitness function* $m(\mathbf{x})$ *by equating it to the original fitness function* $f(\mathbf{x})$ *here at the start.*

3. *Run the GA and pick up the best individual at the end of the run.*
4. *Update* $m(\mathbf{x})$ *as* [6]

$$m_{n+1}(\mathbf{x}) = m_n(\mathbf{x}) \cdot g(\mathbf{x}, \mathbf{s}_n) \tag{7}$$

where n *is the number of so-far run,* s_n *is the best individual in the* n*-th run and*

$$g(\mathbf{x}, \mathbf{s}_n) = \begin{cases} (d_{xs}/r)^\alpha & \text{if } d_{xs} < r \\ 1 & \text{otherwise} \end{cases} \tag{8}$$

is called derating function where d_{xs} *is a distance between* $\mathbf{x}$*) and* $\mathbf{s}_n$*) while* m_0 *is the original raw fitness function of each individual.*

5. *Run the GA using the modified fitness function and keep the best individual found in the run,*
6. *Update the modified fitness function*
7. *If the raw fitness of the best individual exceeds the solution threshold, (See also below) then display this as a solution.*
8. *If all solutions have not been found, then return to step 5.*

- *Solution Threshold is*
  - *Lower fitness limit for maxima of interest, assuming we know how many peaks. If it's not of the case, set the threshold to zero.*

Excersize *Like Figure 11.2, draw a graph of* $y = (x/r)^\alpha$ *with* $r = 1$ *and* $\alpha = 0.5, 1, 2, 4, 8$ *to know what* $g(\mathbf{x}, \mathbf{s}_n)$ *looks like.*

It would be interesting to try a multi-modal EC to the following two test functions.

(1) $y = \sin^6(5\pi x)$

(2) $y = -2((x - 0.2)/0.8)^2 \sin^6(5\pi x)$

[6] This is called a *Power Derating Function* when we think of another alternative called *Exponential Derating Function:*

$$g_e(x, s) = \begin{cases} \exp((\log m(x, s)) \cdot (r - d_{xs}/r)) & \text{if } d_{xs} < r \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

12 Multi Objective Genetic Algorithm (MOGA)

So far we have learned how to get the possible solution(s) which fulfills one objective function for the problem, that is, the goal is maximize the fitness function. In real world problem, however, we have usually multiple objectives or criteria to be fulfilled simultaneously.

Those objectives sometimes conflict with each other. Like "time" and "money": The more we want to earn money, the less time to spent the money; or "reliability" of the product and "cost" to produce it in a manufactural factory. Or, suppose an Opera Company trys to employ one Soprano singer. The criteria is voice, beauty-or-not), slim-or-not, language-capability (Italian, German, etc). However God tend not to give us two talents at a time, alas.

Then, first of all, when we have multiple objective function, we must define an important concept of parate optimal or equvalently non-dominated solution.

Definition (Parate Optimal or Non-dominated Solution) *A candidate solution is called a non-dominated iff there is no ohter better solution w.r.t. all the objectives.*

To be more specific, assume we have n objective functions;

$$f_1(\mathbf{x}), f_2(\mathbf{x}), f_3(\mathbf{x}), \cdots f_n(\mathbf{x})$$

where $\mathbf{x}$ is a candidate solution. Now if a new candidate solution $\mathbf{y}$ improves all the objetives for $\mathbf{x}$, i.e.,

$$f_i(\mathbf{y}) > f_i(\mathbf{x}) \text{ for } \forall i$$

we say

"$\mathbf{y}$ *dominate* $\mathbf{x}$."

When no such $\mathbf{y}$ exists, we say

"$\mathbf{x}$ *is non-dominated*" or "*Parete Optimum.*"

A toy example: We now assume the two objective functions as follows.

$$\begin{cases} f_1(x) = x^2 \\ f_2(x) = (x-2)^2 \end{cases}$$

- x=0 is optimum w.r.t. f_1 but not so good w.r.t. f_2.
- x=2 is optimum w.r.t. f_2 but not so good w.r.t. f_1.

	test-1	test-1	test-1	test-1	test-1	dominated by	dominates	rank
A	9	9	9	8	7	0	3	1
B	5	4	5	3	6	1	2	2
C	3	3	4	2	3	2	1	3
D	2	2	3	1	2	3	0	4
E	1	1	1	1	9	0	0	4

Table 26: An example of who dominates whom and how rank is counted.

- Any other point in between is a compromise or trade-off and is a Pareto-optimum.
- But the solution x=3, e.g., is not a Pareto-optimum since this point is not better than the solution $x = 2$ w.r.t. either objective.
- If we plot in the f_1-f_2 space, an increase in f_1 in some reagion means a decrease in f_2, or vice versa which implys that the solutions in the region are Parete optimum, while in other region an increase in f_1 make f_2 increas (decrease). See Figure **??**. This f_1-f_2 space is called a *Trade-off Space*.

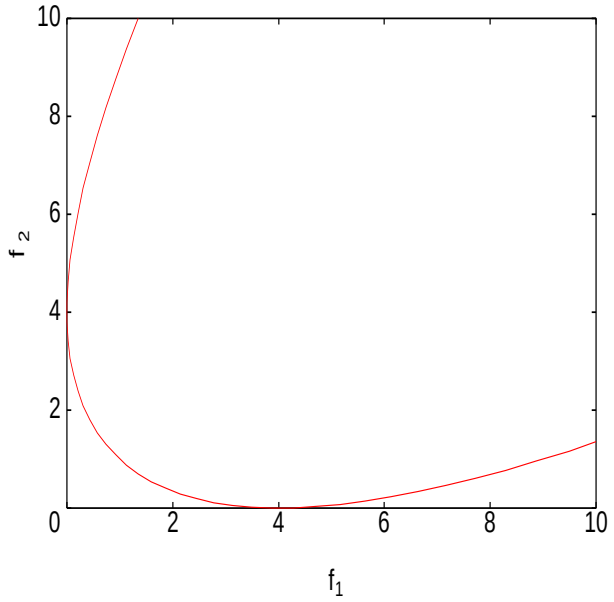


Figure 27: Trade-off space for $f_1(x) = x^2$ and $f_2(x) = (x-2)^2$.

Thought Experiment: What if we plot all individuals of generation 0 and, say, generation 100?

How an implementation goes? Evolution is rather similar more or less to a GA with single fitness function. The main difference is we have multiple objective function. Hence we merge these multiple objective function into one fitness function. So far many ideas have been proposed. Among all:

- Weighted sum approach.
  - The fitness function is caluculated as

$$f(\mathbf{x}) = \sum_{i=1}^{N} w_i f_i(\mathbf{x}) \tag{9}$$

    where w_i expresses an importance of the i-th objectives.
  - Note that for any set of weight > 0, the optimum is always a non-dominated solution but opposite is not always true.
- The minimax approach
  - The fitness function is caluculated by minimizing the maximum of n objective functions.
- Target vector approach
  - The fitness function is caluculated by minimizing the vector

$$(f_1, f_2, f_3, \cdots, f_n)$$

    from a pre-difined goal.
- Median/Average ranking approach
  - The rank $r(\mathbf{x}_i)$ of each individual x_i in the population w.r.t. i-th objective function is calculated. Then the fitness is defined as median/average of these $r_i \;\; (i = 1, \cdots, n)$.
- Parete ranking approach
  - Ranking is according to "*how many individuals in the population they are dominated by.*

We now take a look at a typical implemetation of MOGA.

Algorithm (A Multi Objective GA)

1. *Initialize the population.*
2. *Select individuals uniformly from population.*
3. *Perform crossover and mutation to create a child.*

4. *Calculate the rank of the new child.*
5. *Find the individual in the entire population that is most similar to the child. Replace that individual with the new child if the child's ranking is better, or if the child dominates it.* [7]
6. *Update the ranking of the population if the child has been inserted.*
7. *Perform steps 2-6 according to the population size.*
8. *If the stop criterion is not met go to step 2 and start a new generation.*

Exercise 1 (Parate Optimal Solutions)
Try the algorithm above with two objective functions $y_1 = (x-2)^2$ and $y_2 = (x-4)^2$ as follows:

1. *Create 20 10-bit binary chromosomes, assuming each chromosome represent x-cordinate ranges from 0 to 6 with $(0000\cdots00)$ and $(1111\cdots11)$ being corresponding to 0 and 6, respectively.*
2. *Calculate y_1 and y_2 for each of 20 x's represented by these 20 chromosomes.*
3. *Create a table with 5 columns: (i) chromosome, (ii) its x value, (iii) its y_1 value, (iv) its y_2 value, (v) how many these (y_1, y_2) dominates others (rank).*
4. *Plot these 20 points on each of two graphs (e.g., red and blue color).*
5. *Create next generation by applying the algorithm on these 20 chromosomes.*
6. *While 20 points are different from previous generation Do 2-4 Else stop.*

Give me such table and graph at the first generation, at two intermediate generations, and at the final generation.

[7] Step 5 implies that the new child is only inserted into the population if it dominates the most similar individual, or if it has a lower ranking, i.e. a lower degree of dominance.

The restricted replacement strategy also constitutes an extreme form of elitism, as the only way of replacing a non-dominated individual is to create a child that dominates it.

The similarity of two individuals is measured using a distance function.

13 Fitness Landscape

In Figure reffig:fitness-l a Concept of Fitness Landscape is plotted on also fictiteous domain of 2-dimensional space. For all the possible points in search space, fitness value is calculated and plotted. If the domain is more than 3-dimensional, we could not visualize it, but we could imagin it as a hyper-surface. This (hyper-)surface is called a fitness landscape, usually include peaks and the top of the highest peak corresponds to the global solution, and top regeon of other lower peaks correspond to local optima.

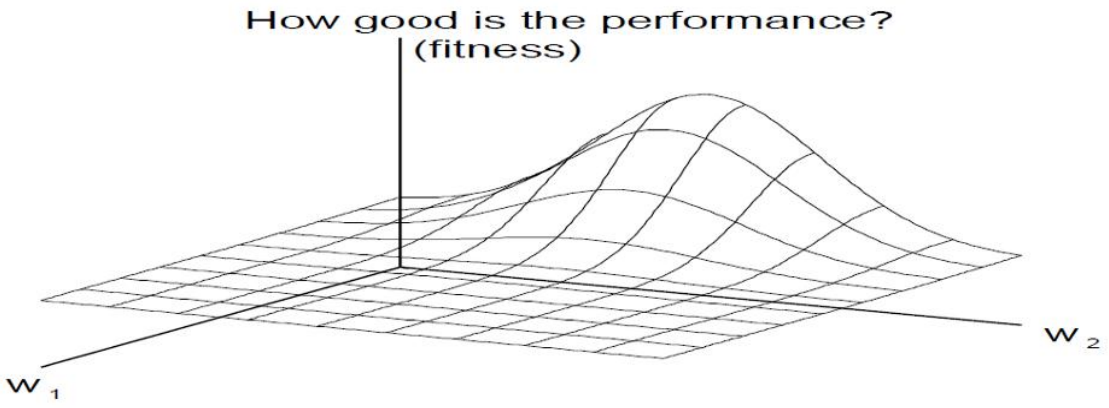


Figure 28: A conceptual plot of fitness value defined on a fictitious 2-dimensional space.

13.1 Hill-climbing

— Random Mutation Hill-climbing (RMHC)—

(1) choose a string at random and call this current-hilltop

(2) choose a locus at random to flip. If the flip leads to an equal or higher fitness then set current-hilltop to the resulting string

(3) goto step (2) until an optimum string has been found or until a maximum number of evaluations have been performed.

(4) return the current-hilltop

(cont'd)

— Steepest Ascent Hill-climbing (SAHC) —

(1) choose a string at random and call this current-hilltop

(2) going from left to right, flip each bit recording the fitness

(3) if any of the resulting strings give a fitness increase then set current hilltop to the resulting string giving the highest fitness increase (ties are decided at random)

(4) if there is no fitness increase, then save current hilltop and goto (1), otherwise to (2) with the new current-hilltop.

— Next Ascent Hill-climbing (NAHC) —

(1) choose a string at random and call this current-hilltop

(2) same as SAHC except that as soon as a fitness increase is found go to step (2) without evaluating any more bit flips with the new current-hilltop and with the bit position being the one where the previous fitness increase was found

(3) if no such increases goto (1)

13.2 A population of hill-climbers

13.2.1 Evolutionary Programming (EP)

13.2.2 Evolution Strategy (ES) – aiming more effectiveness.

13.3 Peculiar Landscapes

However, what if a landscape doesn't have such gradiant information?

13.4 even-n-parity revisit

It is known that an even-n-parity constructed using only XOR and EQ have fitness 0.0, 0.5 or 1.0. That is to say...

14 A Needle in a Haystack Problem

Most banks nowadays facilitate their ATM (automated teller machine) in which we may have a personal account to which we can access with PIN-code, usually four digits of decimal numeral. For security reason, if we failed to enter the PIN correctly more than three times in a row, the PIN would loose its validity thereafter. Then what we are curious is, "How many trials would be needed for random challenges to reveal the secret PIN if an infinite number of trials were permitted?" Let's formalize this problem.

14.0.1 Breaking a PIN

Assuming p-bit octal [8]numeral is employed to construct a PIN, only one out of those 8^p possible combinations is the secret PIN. No one except for the owner of the PIN knows it. Then question is, "How many average trials-and-errors will be needed for a non-owner to know the PIN under a specific strategy?"

This might be reminiscent of the famous problem called *a needle in a haystack* which was originally proposed by Hinton & Nowlan in 1987 [**?**]. The needle in the proposal was exactly the one configuration of 20-bit binary string, that is, the search space is made up of 2^{20} points and only one point is the needle to be searched for. No information such as how close is a currently searching point to the needle, or how likely is a searching point to be the needle. See Figure 1.

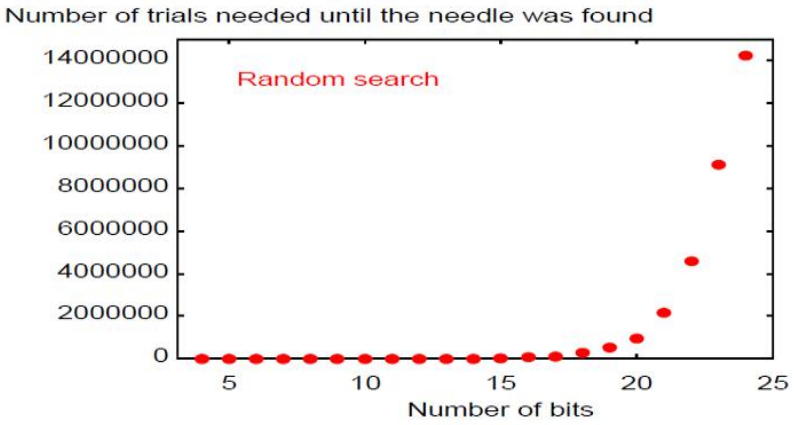


Figure 29: Search will explode exponentially as nuber of bits grows.

14.1 A-tiny-islnd-in-a-huge-lake

That is to say, if the peak is like a mesa which has a flat reagion, extremely steep sidewall, and anywhereelse is totally flat land of fitness zero — we (personally) call it *"A-Tiny-Flat-Island-in-a-Huge-Lake"*. Or, in extreme case, what if only one point in the search space has the fitness one and all other points have fitness zero — This is called a *"A-Needle-in-a Haystack"*, like in Figure

[8]You will see the reason why *"octal"* not *"decimal"* later in the sub-section concerning *"intron"* in the section EXPERIMENTS.

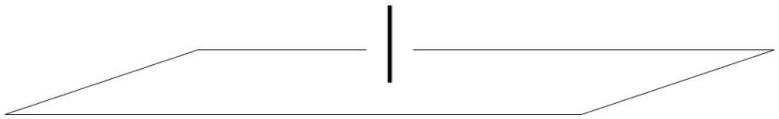


Figure 34: Conceptual figure of a needle in a haystack.

Figure 30: Conceptual figure of a needle in a haystack.

15 Why Giraffe has a long neck? — Lamarckian Inheritance & Baldwin Effect

15.1 A Needle in a Haystack — The most difficult problem.

Once Hinton & Nowlan proposed a very interesting experiment to search for A-needle-in-a-haystack. Their needle and haystack is as follows.

- A-needle ⇒ Only one configuration of 20 bits of binary string.
- Haystack ⇒ $2^{20} - 1$ search points.

For example, (10101010101010101010) is assigned fitness one, while others are fitness zero. Conceptually, we could assume that given a black box to detect a needle like and our task

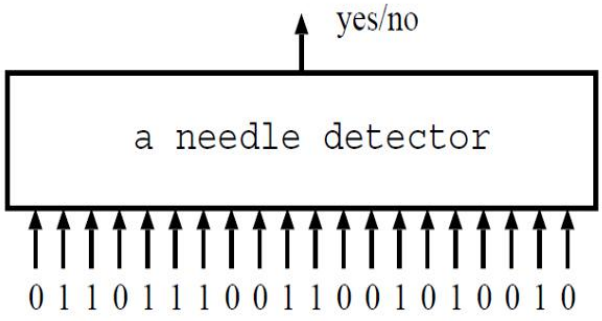


Figure 31: A conceptual black-box to detect a needle.

is to search for the 20-bit binary input which return "YES". We have only one such string out of all the possible 2^{20} inputs.

What Hinton & Nowlan proposed is a learning of each individual during its lifetime. In a biological evolution, if the result affects the next generation, this is called a *Baldwin Effect*. In the Hinton & Nowlan's proposal this is as follows:

- **lifetime learning of each individual (Baldwin Effect).**

- *about 25% are "1", 25% are "0", and the rest of the 50% are "?" .*
- *They are evaluated with all the "?" position being assigned "1" or "0" at random ⇒ learning.*
- *Each individual repeats the learning up to 1000 times ⇒ lifetime learning.*
- *If it reaches the point of fitness one at the n-th trial, then the degree to which learning succeeded is calculated as*

$$1 + 19 \cdot (1000 - n)/1000.$$

Exercise 2 *An experiment on Hinton Nowran's lifetime learning.*

1. *Set TRIAL = 1.*
2. *Create one 20-bit chromosome with its gene "1" gene "0" and gene "?" being 25%, 25%, and 50%, respectively, at random.*
3. *Set COUNTER = 0 and HIT =0.*
4. *Assign "1" or "0" at random to the gene "?" one by one at random.*
5. *If it hits the needle {10101010101010101010} then set HIT=1 and print chromosome and COUNTER, else COUNTER++.*
6. *Do 3-4 while {HIT=0 and COUNTER<1000}.*
7. *Do TRIAL++ and 2-6 while TRIAL<1000.*

Exercise 3 *How its complexity explodes exponentially?*

- ⋆ *Do the above experiment with number of bit of chromosome being 8, 10, 12, 14, 16, 18, 20, 22, ···*
- ⋆ *Report with a graph of number-of-hits Vs. number-of-bit-of-chromosome.*

16 Toward growing structure by evolution

16.1 Evolving Finite State Machine (FSM)

16.1.1 What is FSM?

We human being are living with our feeling, which let's call a *STATE*, always changing according to input from sensor like eyes, and then output some *ACTION*. For example, when my *STATE* is unhappy, an *INPUT* of "vodka" from one of my sensor – mouth changes my *STATE* into happiness, and I began to sing as an *ACTION*.

Thus, in the same way, FSM is defined as a three-tuple $state, input, action$. That is starting with a specific state, input changes it state and aut put an action. This is one step. this procedure of

(1) transition of state according to input;

(2) output an action

are repeated, and produce some behavior of the FSM.

16.1.2 A task as an example

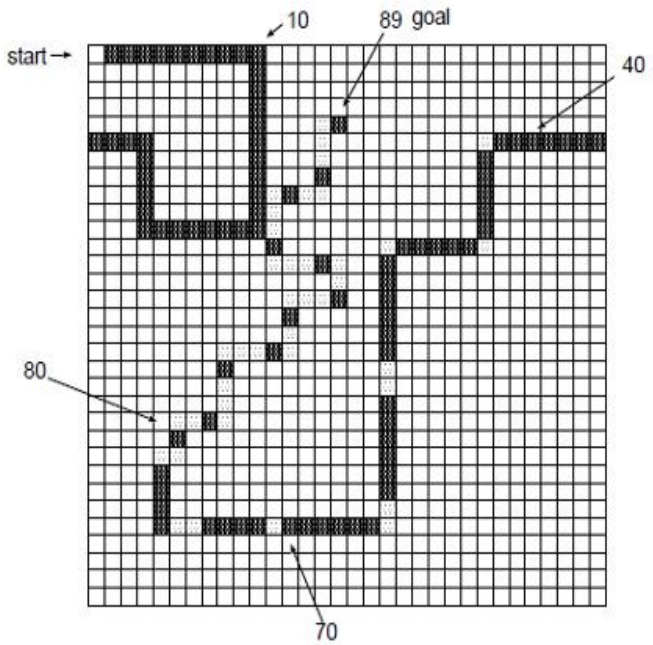


Figure 32: The John Muir trail in 32 × 32 toroidal grid.

16.1.3 FSM Aproach

An example of FSA with 4 states.

The strategy used by this FSA is to move forward whenever it sees a square of trail; when it comes to a point where it does not see the trail in the next square ahead, it turns right (without moving) and checks for trail there. If it finds trail, it moves ahead and continues, but if not, it turns right again. The FSA will turn right a total of 4 times looking trail. After that, it is facing the original direction, and will move forward anyway, even though there is no trail, and will again be prepared to search in all 4 directions again.

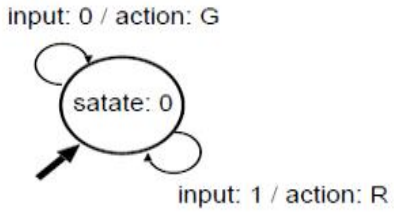


Figure 33: An exapmle of simplest FSM.

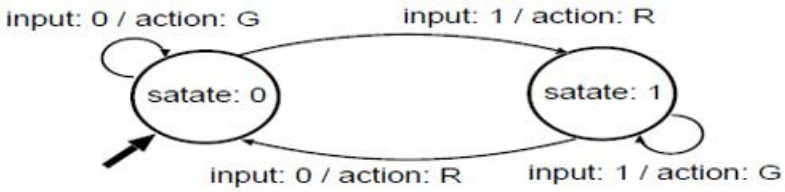


Figure 34: Yet another example of FSM. How do you think its behavior?

Table 35: An example of how to encode transition table into chromosome.

Old State	Input	New State	Action
00	0	00	01
00	1	01	11
01	0	01	01
01	1	11	11
10	0	10	10
10	1	11	11
11	0	10	10
11	1	10	11

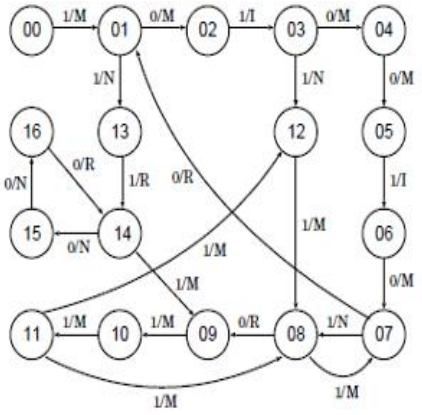


Figure 36: A result of emerging FSM after evolution

16.2 Genetic Programming (GP)

The title of this section suggests *Genetic Programming* suggests *Evolution of Program*. What we want here is when we have specific task and need a program to solve the task, we start with a population of random programs and then evolve them. We expect from generation to generation a better programs than previous ones and perfect programeventually emerges.

As previous evolution, (i) we create a population of random chromosomes; (ii) evaluate fitness of each chromosome; (iii) select chromosomes so that better two are more likely to be selected; (iv) produce children by crossover and mutation (v) repeat (iii) to (iV) until the next generation includes the same number of chromosomes as previous generation; (vi) repeat (ii) to (v) until fitness saturates or pre-determined generations are repeated.

But here, chromosome is not a string like so far, but *tree*.

16.2.1 How we create random tree, and how we evolve trees.

We, first of all, prepare for function set and terminal set. Then we follow the below.

(1) Choose one function from the *Function Set* at ramdom, and assign it to the root node.

(2) Assign each of arcs a function or a terminal chosen at random from *Function Set* or *Terminal Set*.

(3) If the node is a terminal, then the node will not grow any more, but instead become a *leave*. Oterwise repeat (2).

(4) These are repeated until all the end nodes are terminal set.

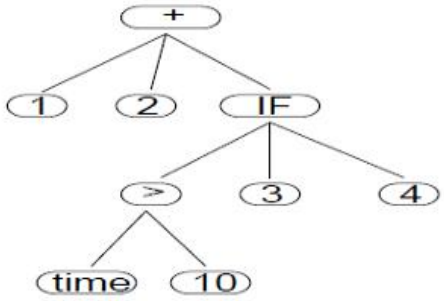
16.2.2 Evolution of program.

Like the program language *LISP*, some programming languages have a tree-structure.

16.2.3 An Example of Tree represantation of a program

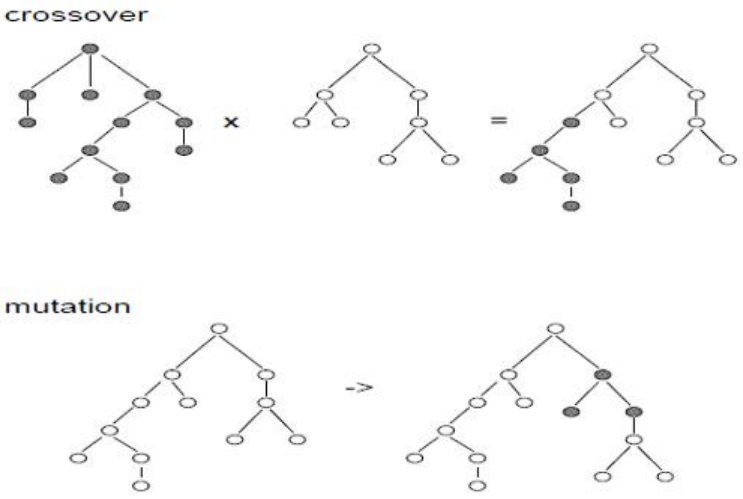
Program in LISP can be represent by a tree. The following is a simple example of one instruction from LISP program and its tree structure.

(+ 1 2 (IF (> time 10) 3 4))



16.2.4 Crossover and Mutation

We now look at how we crossover and mutate two trees. See the figure below.



16.3 Messy Genetic Algorighm (m-GA)

So far our chromosome has a fixed length. It is good to manipulate chromosomes of fixed length, but some time we want to be more flexible. So, in this section we are going to try to expand so-far-studied fixed length chromosome to exploit arbitrary lengh chromosome. Now we assume a binary chromosome such as

(1 0 0 1 1)

In the m-GA this chromosome is translated into

((1,1) (2,0) (3,0) (4,1) (5,1))

16.3.1 Gene location is not important any more.

As you easily guess, each gene is made up of a pair of (1) location of the gene and (2) its value. For example the first gene $(1, 1)$ means *the value of 1st gene is 1* and $(2, 0)$ means *the value of 2nd gene is 0* and so on. Now that we have information of location of the genes the order of genes in m-GA is not important. So, we assume

((2,0) (4,1) (1,1) (5,1) (3,0)) & ((4,1) (1,1) (5,1) (2,0) (3,0))

are equivalent.

16.3.2 Cut & Splice

Then crossover in m-GA is made by cutting the parent chromosom at any location on both parents, while in the one point crossover we have so far used we cut the both parent at the same position. Then those are crossed and spliced and create two chromosomes. For example, assuming we have two parents

((4,1) (1,1) (5,1) (2,0) (3,0)) & ((3,0) (5,0) (4,1) (1,0) (2,0))

Then, in m-GA, if we cut the 1st parents between 1st and 2nd gene and 2nd parent between 4th and 5th gene, then this will result in the two children

((4,1) (2,0)) & ((3,0) (5,0) (4,1) (1,0) (1,1) (5,1) (2,0) (3,0))

16.3.3 Over-specification & Under-specification

As the previous two chromosomes, some location of gene will be missing and some will appear multitple time. The former called *under-specification* and the latter Usually, *over-specification* case are treated as *first-come-first-serve* way, like McDonald Restaurant. That is,

((3,0) (5,0) (4,1) (1,0) (1,1) (5,1) (2,0) (3,0))

is interpreted as

((3,0) (5,0) (4,1) (1,0) (2,0)).

If we interpret this in the standard simple GA it will be

$$(0\ 0\ 0\ 1\ 0).$$

In the case of *under-specification*, one way is giving the missing genes at random. Or, sometimes it will be a good way to neglect the missing genes depending on applications thouth.

16.3.4 Growing NN structure by m-GA

Let's assume each of our messy-genes is made up of for sub-genes like

$$(\cdots, ((6)\ (8,\ 0.28,\ 1))\ ((7)\ (12,\ -0.28,\ 0))\ \cdots).$$

Let's assume the two genes shown in the above example chromosome implies

- 6th gene should be connected to 8th gene with synapse whose weight is 0.28, and the connection is active.
- 7th gene should be connected to 12th gene with synapse whose weight is -0.73, but the connection is not active.

To be more specific, n-th gene describes (1) to which neuron the n-th neurons is to be connected; (2) the weight value of the connection; (3) enable or not

16.4 NEAT

NEAT[9]

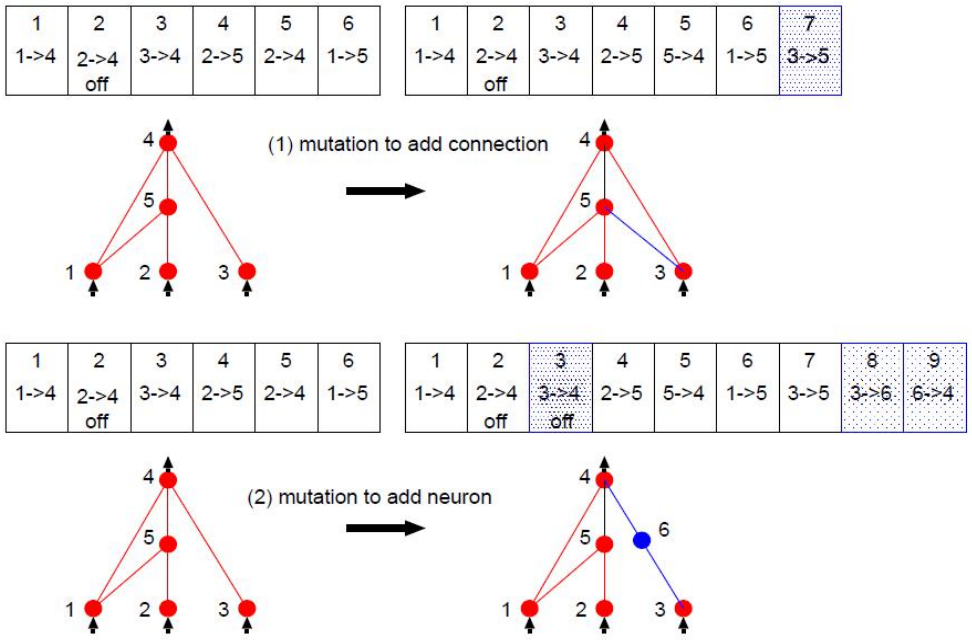


Figure 37: Two types of mutation in NEAT. (Re-drawed the figure in Stanley et al.)

16.5 Growing structure by evolution

[9]K. O. Stanley and R. Miikkulainen (2002) "Evolving Neural Networks through Augmenting Topologies" Evolutionary Computation, Vol. 10, pp. 99-127.

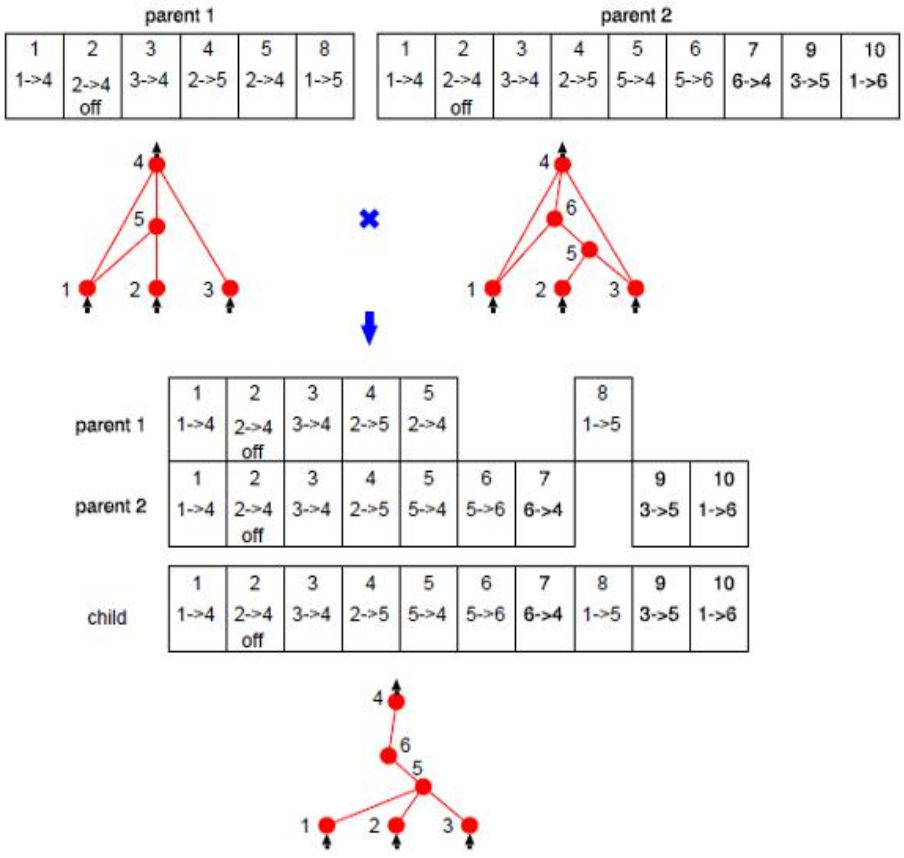


Figure 38: A crossover in NEAT. (Re-drawed the figure in Stanley et al.)

APPENDIX

I. The other commmonly used test-function

Shcwefel function

The function bellow is called Schwefel function and enormous amount of local minimum and the only unique global minimum at the Origin. Search for the global minimum when the function is difined on, say, $x_i \in [-500, 500]$.

$$y = \sum_{i=1}^{n} (x_i \sin(|x_i|)) \tag{10}$$

You might try to explore this hyper-surface by setting n to 20, for example. Then the surface is defiened on 20-dimensional Eucledean space. If, however, you want to know the image of the hyper-surface, see a two dimensional version of this function. The graph of

$$y = x \sin(|x|) \tag{11}$$

is shown in Figure 1.

Figure 39: A two-dimensional version of the Schwefel's Function's graph.

16.6 Rastrigin's Function

⋆ Rastrigin's Function

$$y = nA + \sum_{i=1}^{n} (x_i^2 - A\cos(2\pi x_i)), \quad x_i \in [-5.12 - 5.12].$$

· Ruggedness might be controlled by modifying the value of A.

· A two dimensional example ($n = 1$):

$$y = 3 + x^2 - 3\cos(2\pi x).$$

Figure 40: A 2-D version of Rastrigin function

⋆ Griewangk's Function F_8:

$$y = \sum_{i=1}^{n} x_i^2/4000 - \prod_{i=1}^{n} \cos(x_i/\sqrt{i}) + 1, \quad x_i \in [-600, 600].$$

· A two dimensional example:

$$y = x^2/4000 - \cos x + 1.$$

Figure 41: A 2-D version of Griewangk function

⋆ Ackley's Function F_9:

$$y = -20 \sum_{i=1}^{n} \exp\left(-0.2\sqrt{x_i^2/n}\right) - \exp\left(\left(\sum_{i=1}^{n} \cos 2\pi x_i\right)/n\right) + 20 + e,$$

$$x_i \in [-30, 30].$$

· A two dimensional example:

$$y = -20\exp\left(-0.2\sqrt{x^2}\right) - \exp\left(\cos 2\pi x\right) + 20 + e.$$

Figure 42: A 2-D version of Ackley function