An Introduction to Evolutionary Computation

(September — December, 2008)

Akira Imada

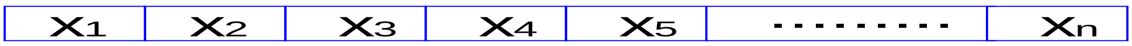
(Editing still on going – last modified on)

March 4, 2009

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:

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



- Define a *fitness* evaluation by "How good is each 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.

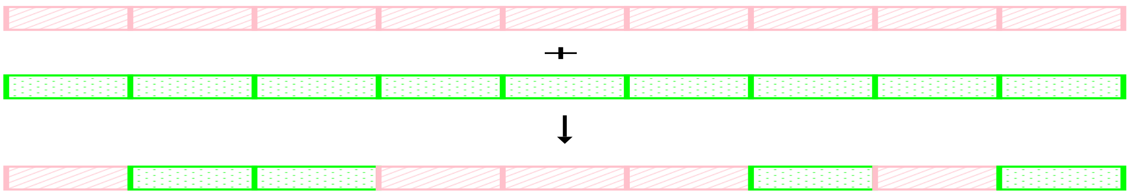
2. **(Fitness Evaluation)** Evaluate fitness of each chromosome and sort the chromosomes according to its fitness from the best to the worst.

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.

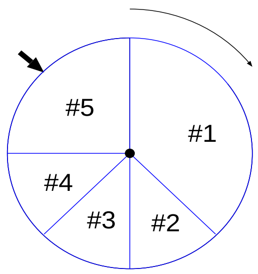
2 How we select parents?

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

- 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):
    - · Select such that the probability to be selected is proportional to the fitness value.

  For example,

	fitness
individual #1	0.375
individual #2	0.125
individual #3	0.125
individual #4	0.125
individual #5	0.250



  To be more specific, sort the individual from low to high and calculate cumulative value of fitness as follows:

individual after sort	cumulative value of fitness
#2	0.125
#3	0.250
#4	0.375
#5	0.625
#1	1.000

  Then create a random number r from 0 to 1 if $r < 0.125$ then select #1, else if $r < 0.250$ then select #2 , else if $r < 0.375$ then select #3 and so on.

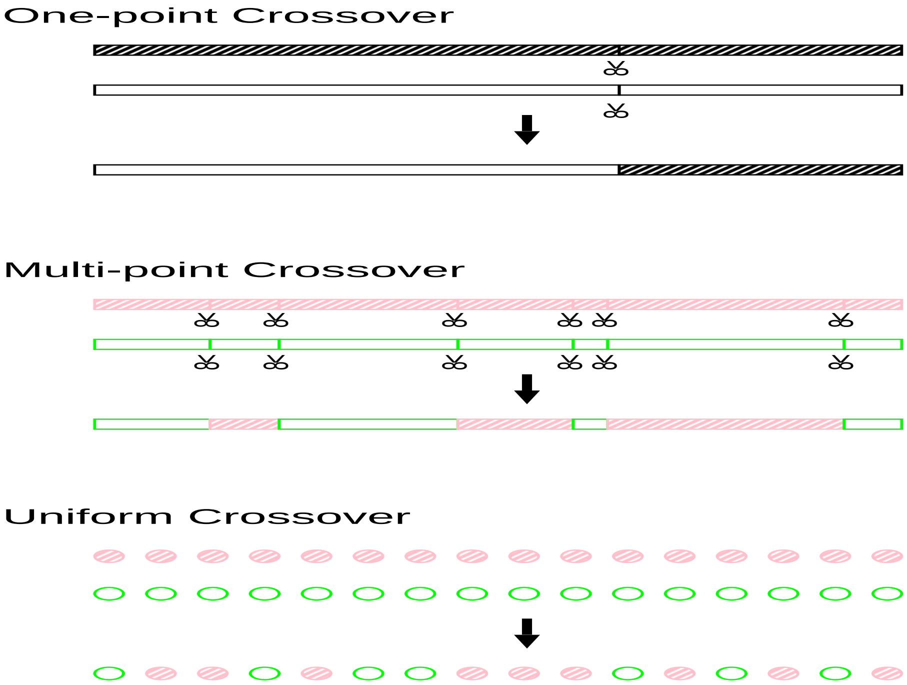
  - ⋆ Tournament Selection
    - · Assume we have the original μ parents and their μ children. The fitness value of each of the 2μ individuals are compared to those of q individuals which are chosen randomly from the whole 2μ points at every time of the comparison Then the 2μ individuals are ranked according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

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.

3 How parents produce a child?

3.1 Cross-over

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



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.*

4 Commonly Used Test Functions

Let's challange an already known solution by a GA

Bellow 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.

4.1 Sphere model

Probably the simplest one is

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

which is difined, for example, on each $x_i \in [-5.12, 5.12] \quad (i = 1, 2, \cdots, n)$. This function is an extention of well known $y = x^2$ to its n-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.

4.2 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, $x_i \in [-500, 500]$.

$$y = \sum_{i=1}^{n} (x_i \sin(|x_i|)) \tag{2}$$

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

$$y = x \sin(|x|) \tag{3}$$

is shown in Figure 1.

4.3 The other Test Functions:

⋆ Rastrigin's Function F_6:

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

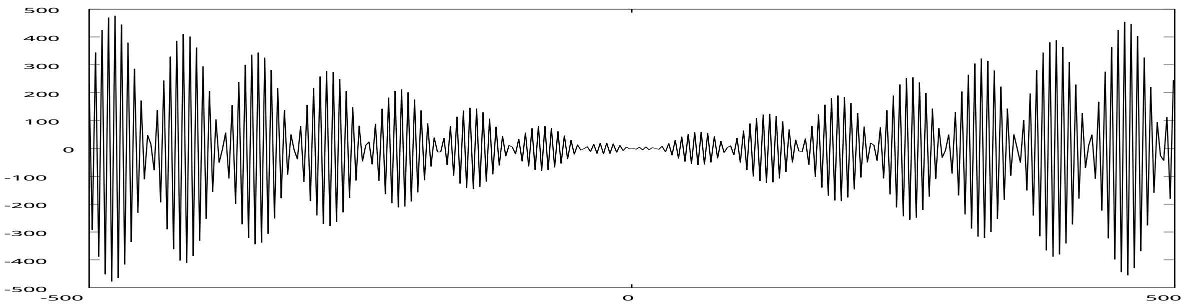


Figure 1: A two-dimensional version of the Schwefel's Function's graph.

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

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

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

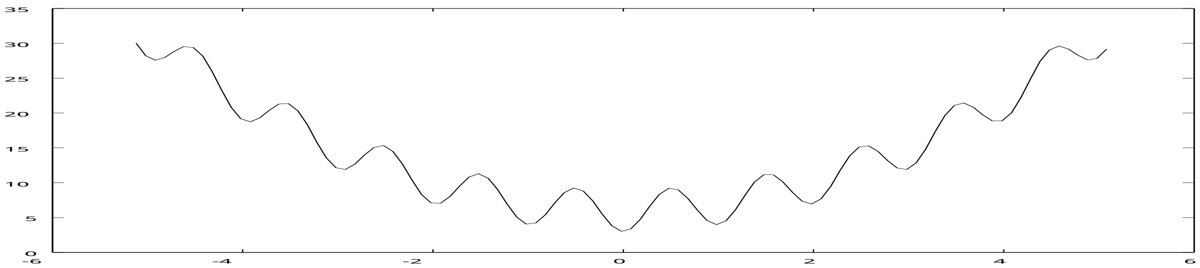


Figure 2: A 2-D version of Rastrigin function

⋆ 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:

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

⋆ Ackley's Function F_9:

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

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

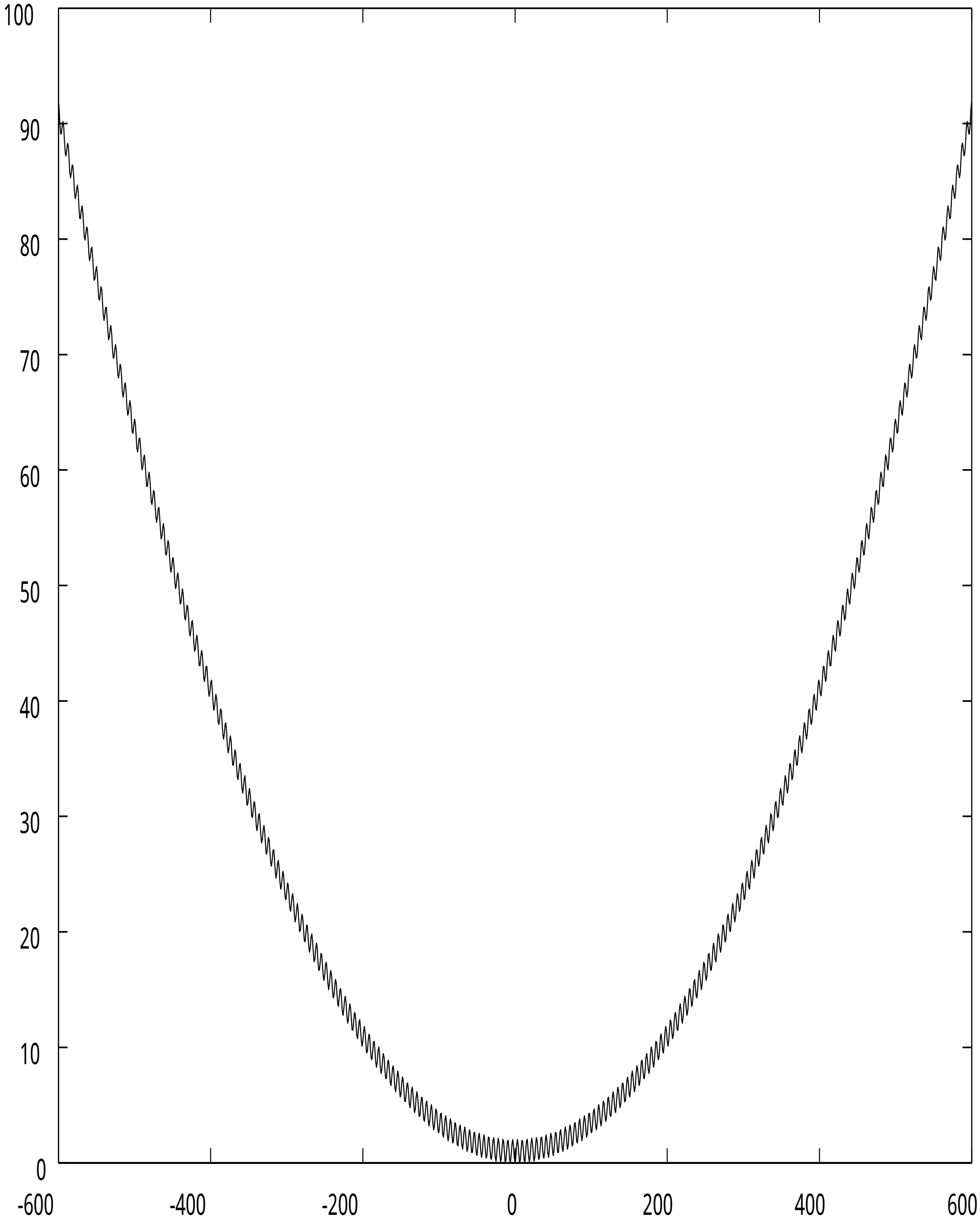


Figure 3: A 2-D version of Griewangk function

· A two dimensional example:

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

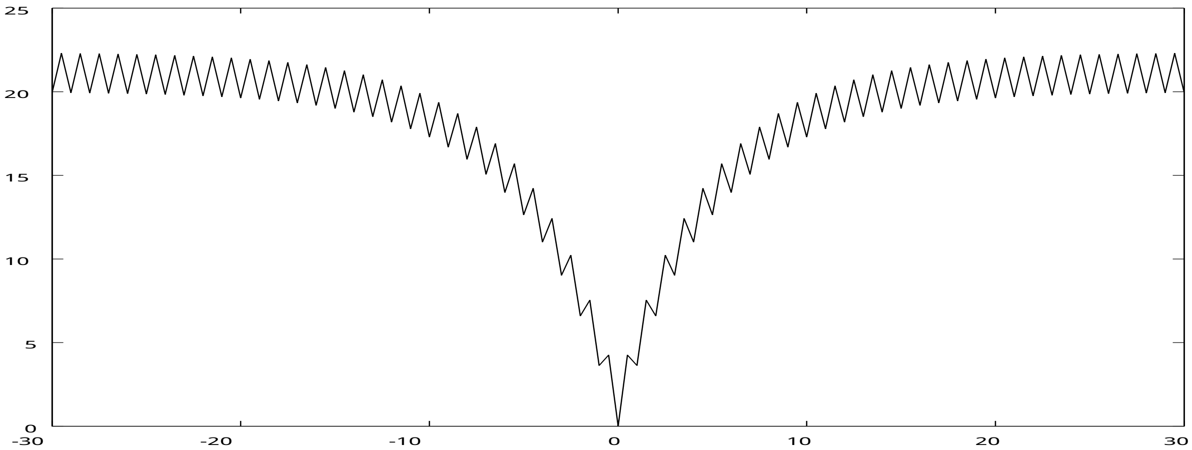


Figure 4: A 2-D version of Ackley function

5 What about a simple 2-Diminsional function

We now think of more simple function defined on just x – one dimensional variable, and let's observe the behavior of chromosomes if the function has two minima, one is a local minimum and the other the global minimum. Example is,

$$y = x^4 - 5x^3 - 6x^2 + 8x + 15$$

when difined on $x \in [-2, 5]$

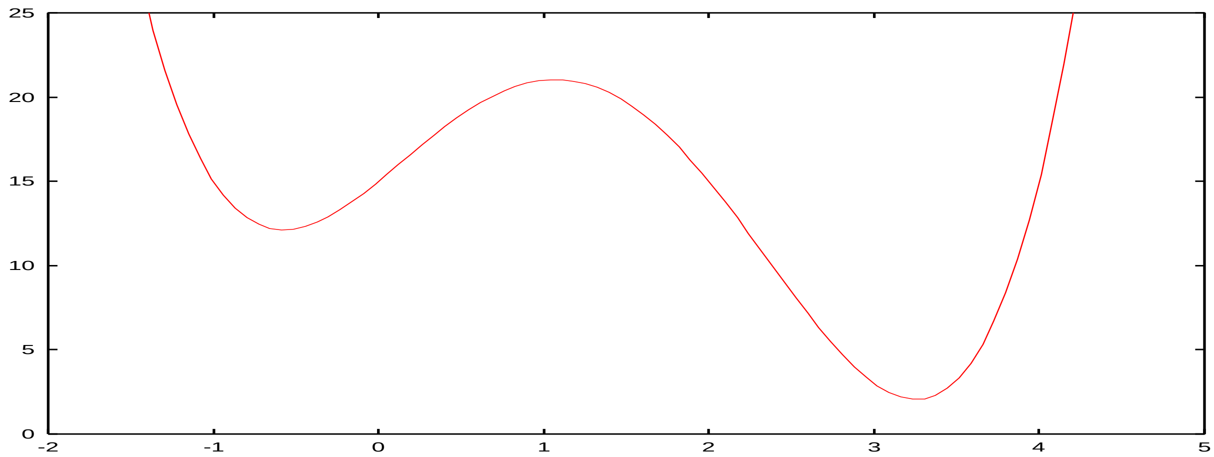


Figure 5: Yet another test function: $y = x^4 - 5x^3 - 6x^2 + 8x + 15$ with $x \in [-2, 5]$. Try to make evolutionary algorithm search for minimum y.

6 Neural Networks for XOR

This is probably a simplest examples. We now try the below.
Assuming McCulloch-Pitts neurons which take the state 1 or 0, the output Y of the neuron which receives weighted-sum of the signals X_i from other N neurons is usually specified as:

$$Y = \mathrm{sgn}(\sum_{i=1}^{N} w_i X_i - \theta),$$

where $\mathrm{sgn}(x) = 1$ if $x \geq 0$ and 0 otherwise, and w_i and θ are called *weight* and *threshold*, respectively. Here, we assume neurons take binary state but -1 or 1, instead of 0 or 1. Hence the equation is modified as

$$Y = 2\mathrm{sgn}(\sum_{i=1}^{N} w_i X_i - \theta) - 1.$$

6.1 NN to solve AND & OR

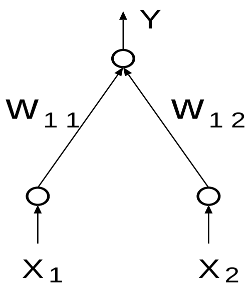
To start let's search for a neural network which solve AND & OR Boolean function.

AND

X_1	X_2	Y
-1	-1	-1
-1	+1	-1
+1	-1	-1
+1	+1	+1

OR

X_1	X_2	Y
-1	-1	-1
-1	+1	+1
+1	-1	+1
+1	+1	+1



These tasks are so easy that even we can assing an appropriate configuration of weiths. For example, $w_1 = 0.5$, $w_2 = 0.5$ and $\theta = 0.5$ for AND logic, and $w_1 = 0.5$, $w_2 = 0.5$ and $\theta = 0.5$ for OR.

But why don't we try to test the capability of an evolutionary computation.

6.2 NN to solve XOR

Well, how about NN to solve XOR? In this case, it is known that we need one more layer, called a *hidden layer.* The reason is ...

Exercise 1 *Obtain six weights values so that the above NN function as XOR.*

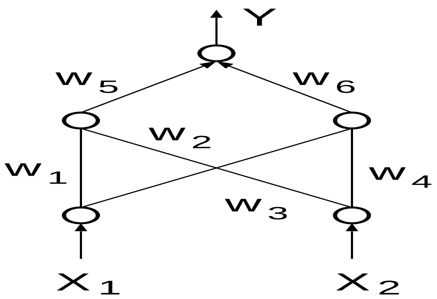
Exercise 2 *Create Pseudo code for EC to obtain the six weights above.*

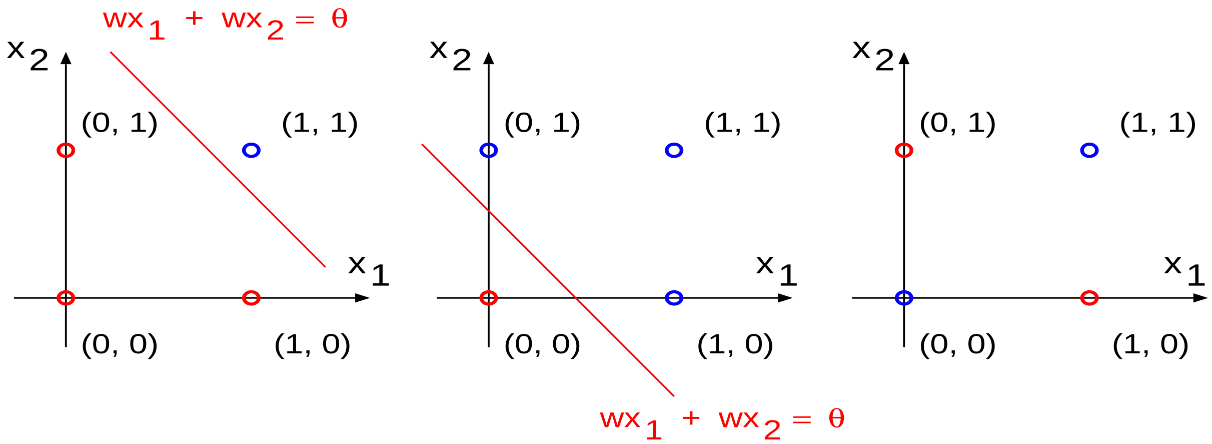
6.3 Larger & more sofisticated NN

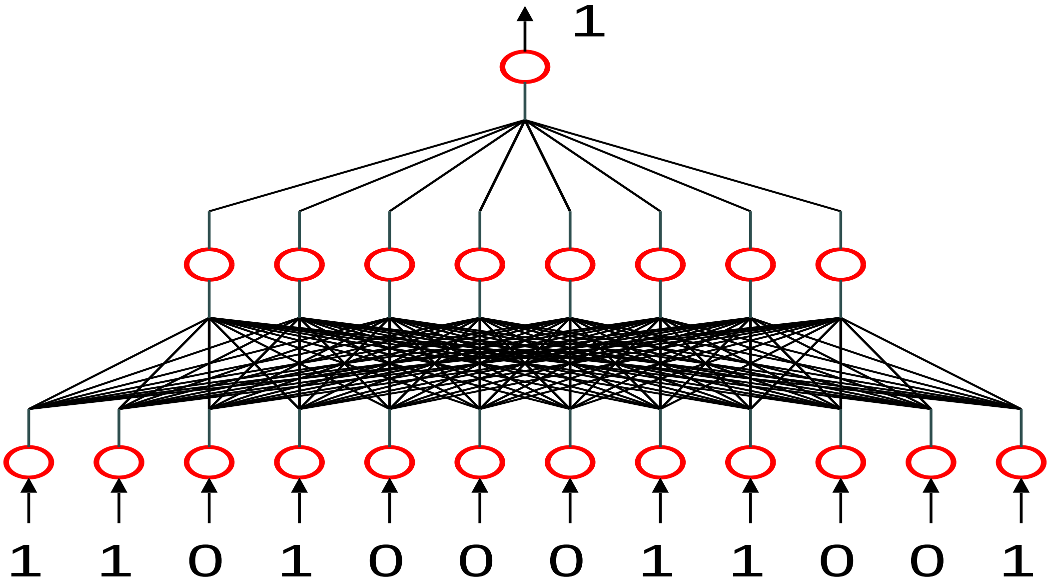
Let's extend the scale of NN. For example N parity problem

XOR

X_1	X_2	Y
-1	-1	-1
-1	+1	+1
+1	-1	+1
+1	+1	+1







7 NP-complete Combinatorial Problem

7.1 Knapsack Problem

This is one of the most popular *NP-complete Combinatorial Optimization* problems.

We now assume n items whose i-th item has weight w_i and profit p_i, then we pick up x_i of the i-th item $i = 1, 2, \cdots, n$ and x_i is non-negative integer. The goal is to maximizes

$$\sum_{i=1}^{n} x_i p_i. \tag{4}$$

such that

$$\sum_{i=1}^{n} x_i w_i < C \tag{5}$$

where C is the capacity of the knapsack.

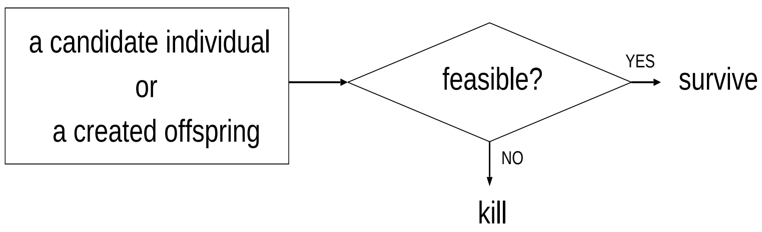
GA implementation is quite simple. Our chromosomes are in the form

$$(x_1 x_2 x_3 \cdots x_n) \tag{6}$$

with each x_i being the number of item i to be in the knapsack.

7.2 Kill infeasible chromosomes

One important aspect is if a chromosome does not fulfill the condition of Eq.(5), simply kill the choromosome and repeat the procedure which resulted in the infeasible child chromosome (cross-over, mutation, or whatever.) untill creating a feasible child chromosome.



7.3 Bin-packing

8 Combinatorial Optimization II

8.1 Linear Assignment Problem (LAP)

8.2 Quadrapul Assignment Problem (QAP)

8.3 Traveling Salse-person Problem (TSP)

Asuuming N cities all of whose cordinate are given. 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.

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.

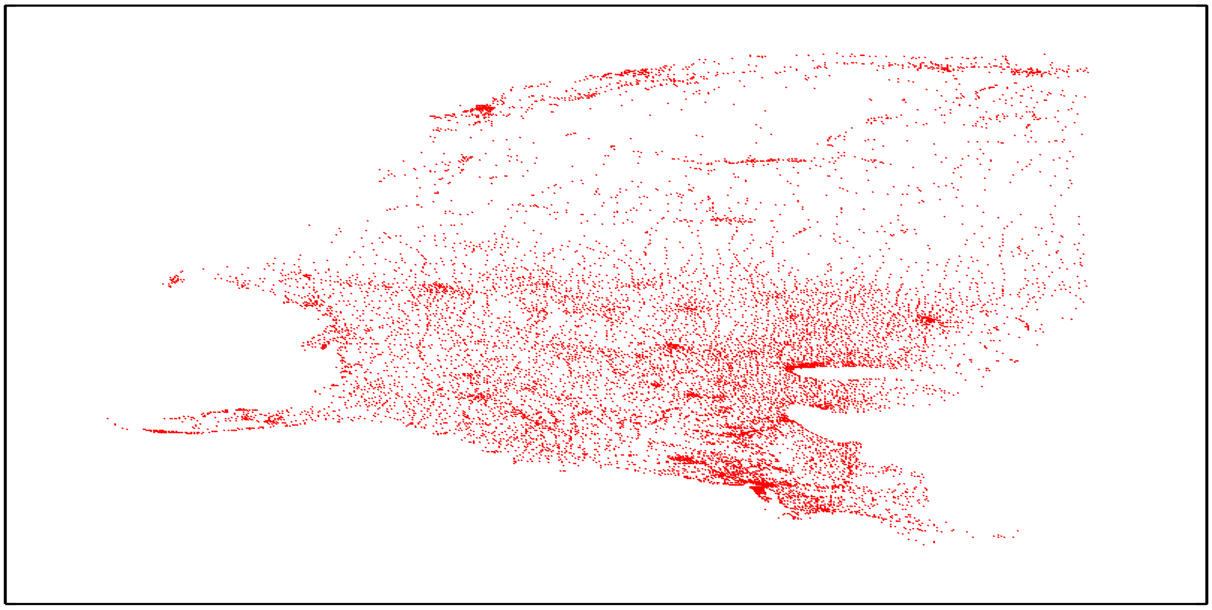


Figure 6: An example of 13509 real cities' location in US. Plotted with the data taken from http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95.

The question is how we 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

$$(ACFDGEB) \tag{7}$$

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

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).

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

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.

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.

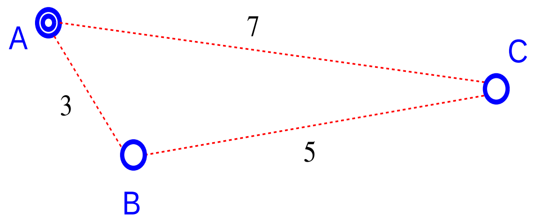


Figure 7: An example of three cities. Though rivial because we have only one tour.

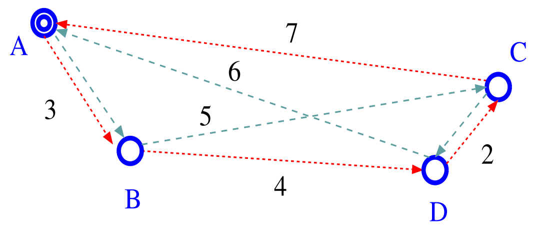


Figure 8: An example of four cities. Still trivial, but which do you think is the shortest tour?

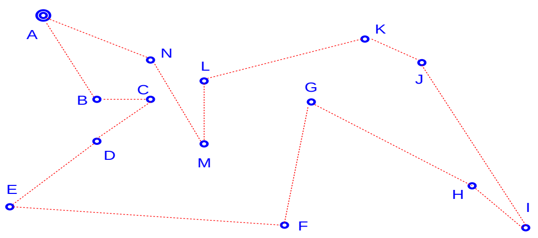


Figure 9: An example of fourteen cities and its one option of tour.

9 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.

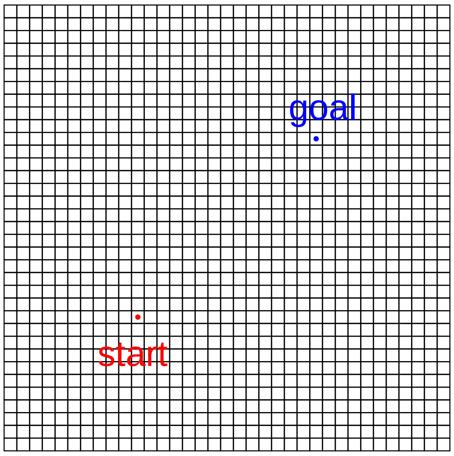


Figure 10: An example of shortest path finding without any obstacle.

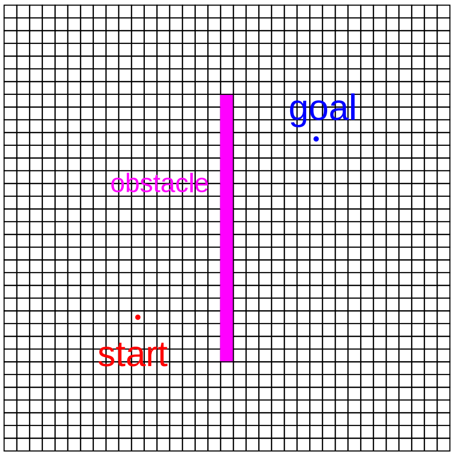


Figure 11: An example of shortest path finding with an obstacle of wall.

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.

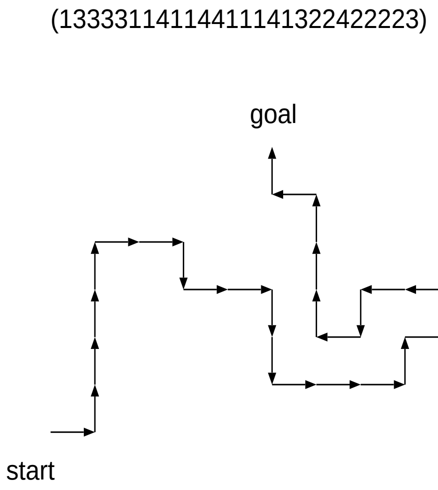


Figure 12: An example of a chromosome and the path represented by it.

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

9.1 Exploration of a gridworld with a limited energy

Search for a path of minimum Manhattan distance

.

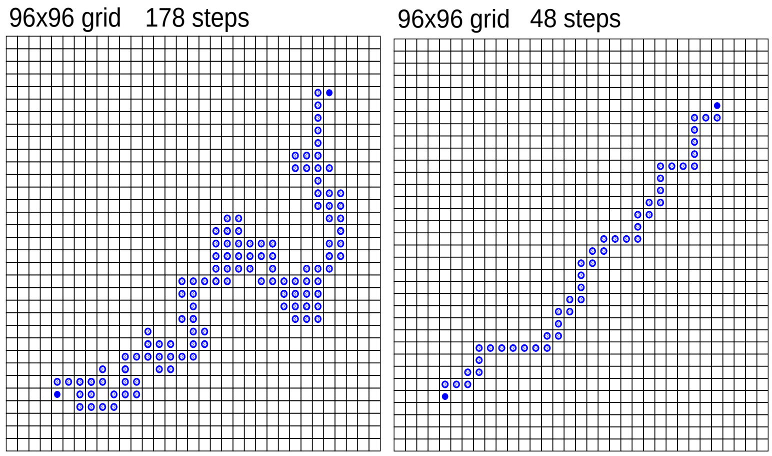


Figure 13: In the grid-world of 96 starting from (24,24) a robot walks aiming the goal at (72,72) of which the robot had no *a-priori* information. Left: The path of minimum length among 100 trials by random walk. Right: Minimal path the robot found after an evolutionary learning as shown in Fig. 3.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

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.

9.2 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 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. [1]

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).

[1] 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.

10 Sorting Network – How many minimum comprisons 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 1 (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.

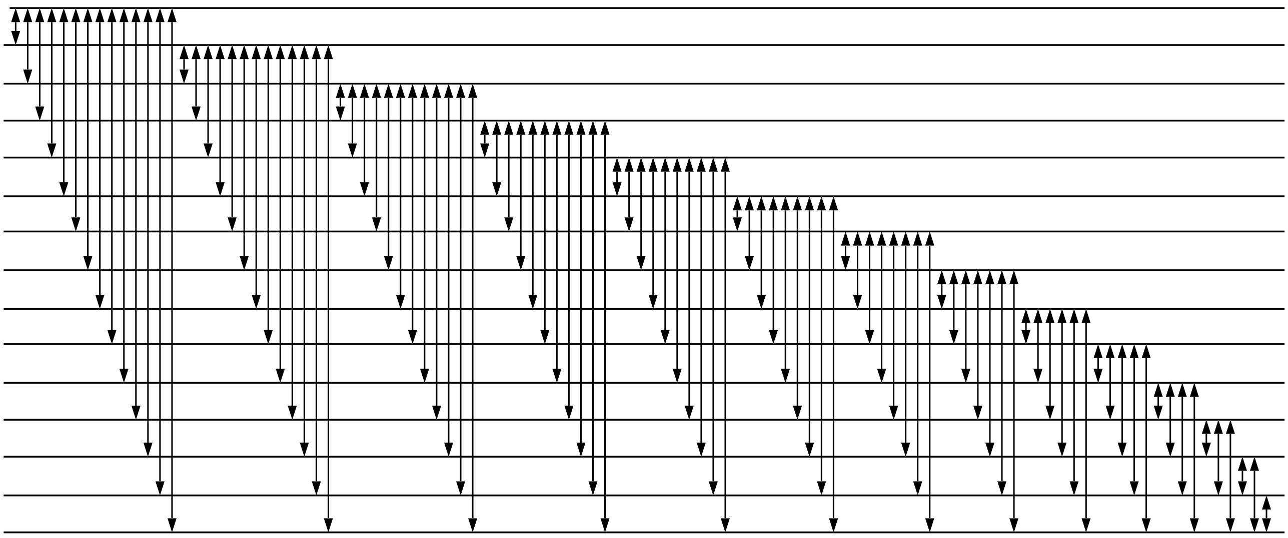


Figure 14: A typical network for sorting, not very efficient though.

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 competion 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.

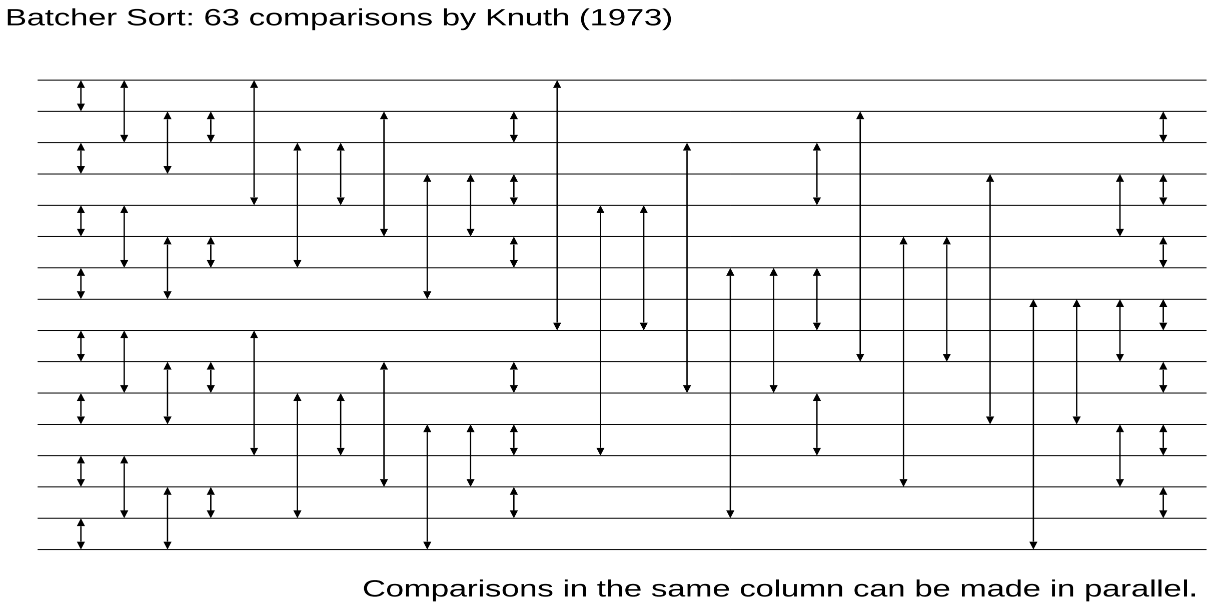


Figure 15: Batcher's proposition of sorting network with 63 comprisons (1964)

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.

11 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.[2]

11.1 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?

11.1.1 Condition to be a dilemma

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

In the community of *Game Theory* we have the problem called Prisoner's Dilemma[3] which is formulated as follows.

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

- *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.*

[2]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?*

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

· *If both do not they will each get 2 years.*

Cleary, "0 year in prison" would be the best reward, then "2 years" would be the second best, then "4 years" and "1 year" would be the worst case, unless you expected the prison as a "free hotel."

Here, let's assume those reward as 15, 10, 6 and 0, respectively. Then rewards are:

			A recieves	B recieves
A confess	&	B silent	15	0
A confess	&	B confess	6	6
A silent	&	B silent	10	10
A silent	&	B confess	0	15

Here, A might get "15" with a possible risk of getting only "6" so A could choose "10" which is larger than "6" but possibly "0" at the worst case. This raises a *dilemma*

We now think of another case. Will the following case raise a dilemma?

			A recieves	B recieves
A confess	&	B silent	15	0
A confess	&	B confess	10	10
A silent	&	B silent	6	6
A silent	&	B confess	0	15

The answer is "No." Prisoner would get either 15 (at best) or 10 (at least) by confessing while 6 or 0 by being silent. Hence no dilemma. Confess immediately! This guaranttees larger reword than being silent regardless of opponent's reaction.

So what is a condition to raise a dilemma? We now assume:

			A recieves	B recieves
A confess	&	B silent	γ_3	γ_2
A confess	&	B confess	γ_4	γ_4
A silent	&	B silent	γ_1	γ_1
A silent	&	B confess	γ_2	γ_3

Then, from the above mentioned two examples, we can say

$$\gamma_3 > \gamma_1 > \gamma_4 > \gamma_2. \tag{8}$$

			A recieves	B recieves
A confess	&	B silent	15	0
A confess	&	B confess	3	3
A silent	&	B silent	6	6
A silent	&	B confess	0	15

Then what about the following case?

This follows the above condition. But does this raise dilemma? Seems not. So, we need yet another condition:

$$2\gamma_1 > \gamma_2 + \gamma_3 \tag{9}$$

If you don't agree, then think of more extreme case:

			A recieves	B recieves
A confess	&	B silent	15	0
A confess	&	B confess	1	1
A silent	&	B silent	2	2
A silent	&	B confess	0	15

Not a good idea being silent isn't it?

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?

11.1.2 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.

12 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).*

12.1 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 1: 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.

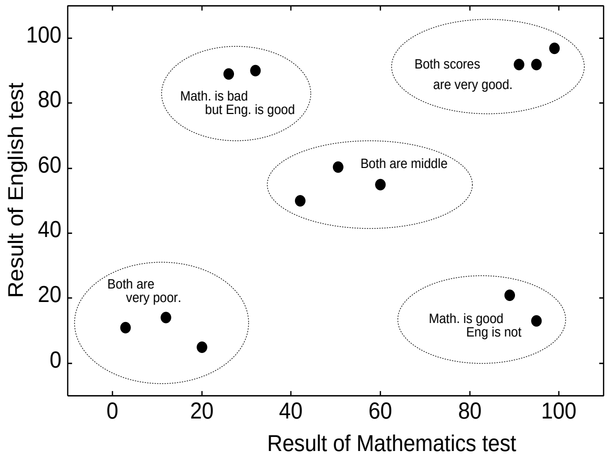


Figure 16: A visualization. It's easy to classify soldires into 5 groups.

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.

12.2 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:

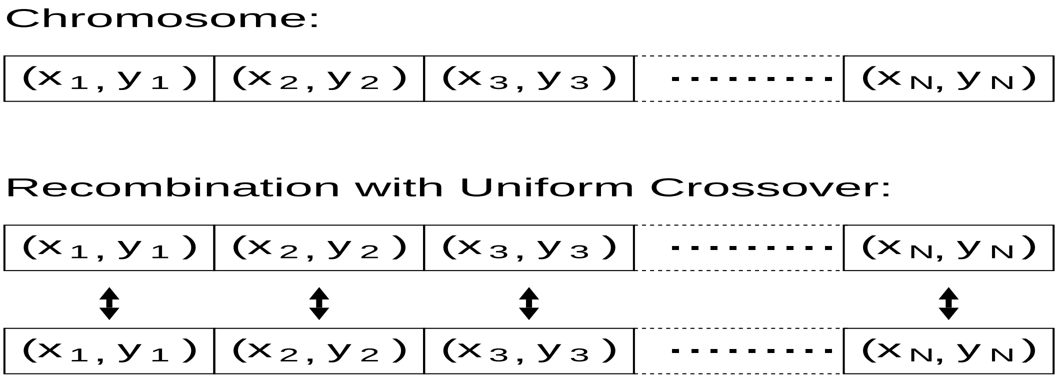


Figure 17: A chromosome representation and uniform crossover

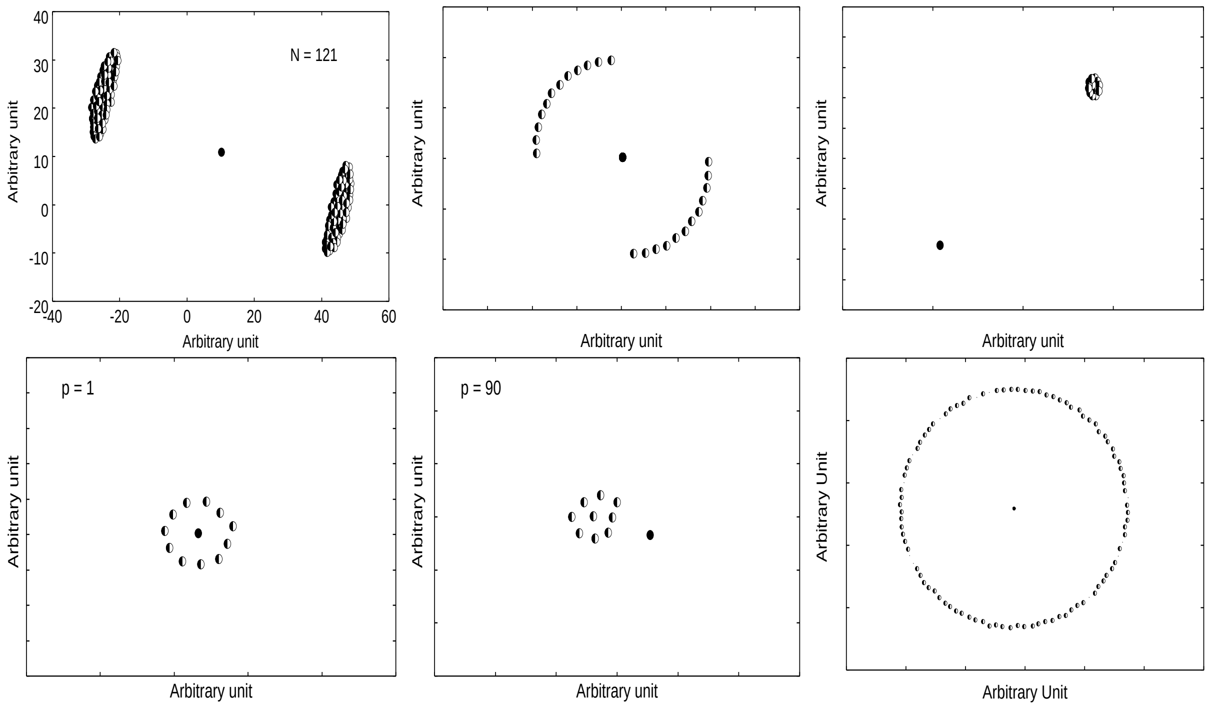


Figure 18: Six Examples of Mapping from 2401-dimensional space to the 2-dimensional space. Further explanations are shown in the text.

13 Sorting Network Revisited

13.1 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 19: 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.,

(09 08 10 13 14 04 14 03)

indicates the four comparisons below.

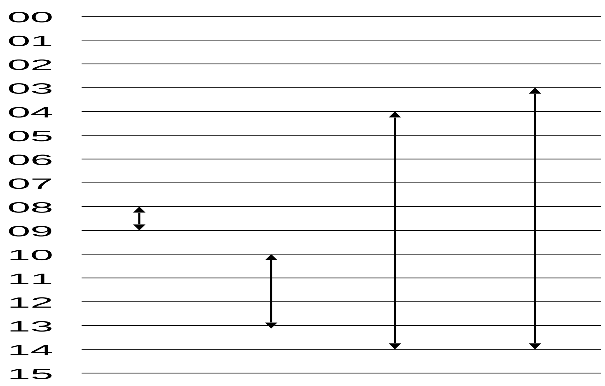


Figure 20: Four comparisons specified by the chromosome (09 08 10 13 14 04 14 03).

- 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.

13.2 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.

14 Evolving Finite State Machine (FSM)

14.1 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.

14.2 A task as an example

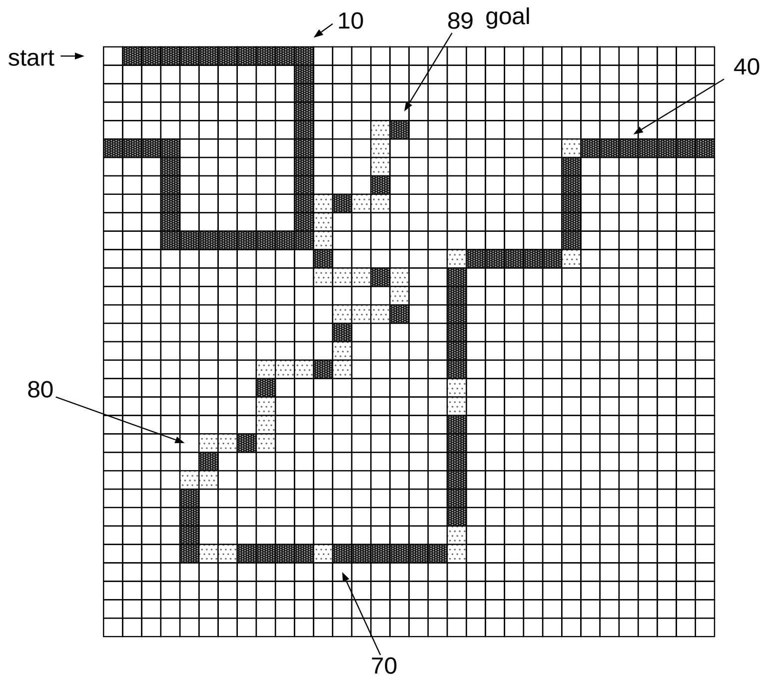


Figure 21: The John Muir trail in 32 × 32 toroidal grid.

14.2.1 NN Aproach

14.2.2 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.

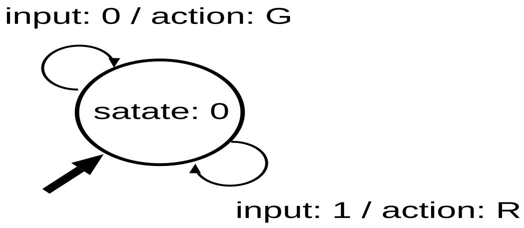


Figure 22: An exapmle of simplest FSM.

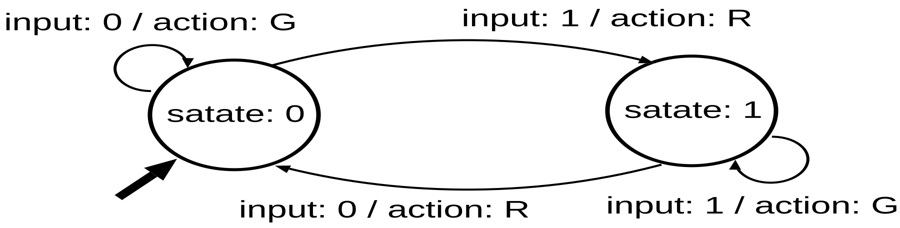


Figure 23: Yet another example of FSM. How do you think its behavior?

Table 2: 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

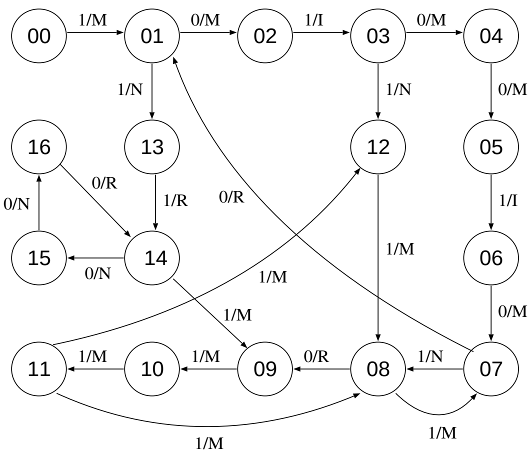


Figure 24: A result of emerging FSM after evolution

14.2.3 Hidden Markov Model (HMM)

From FAM which is deterministic to HMM which is indeterministic.

15 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.

15.1 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 $(00 \cdots 0)$ and $(11 \cdots 1)$, 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 comapre *Gray Code* where gray-code $a_1 a_2 \cdots a_n$ is tlanslated from binary number $b_1 b_2 \cdots b_n$ as

$$a_i = \begin{cases} b_i & \text{if} \quad i = 1 \\ b_{i-1} \oplus p_i & \text{otherwise} \end{cases} \tag{10}$$

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. 4 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,

$$y = \sin^6(5\pi x) \tag{11}$$

or

$$y = -2((x - 0.2)/0.8)^2 \sin^6(5\pi x) \tag{12}$$

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

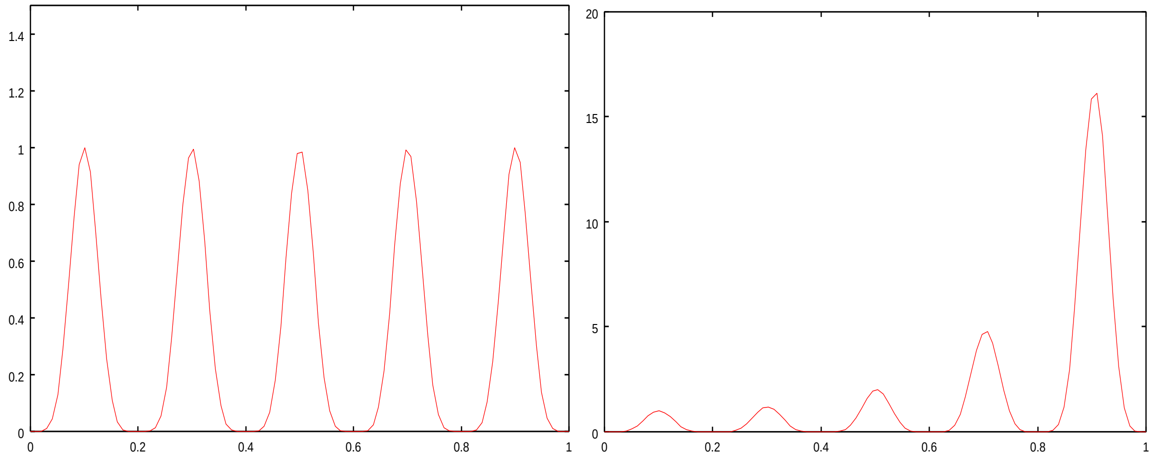


Figure 25: A multi-peak 2-D function and its variation

15.2 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 15.2.

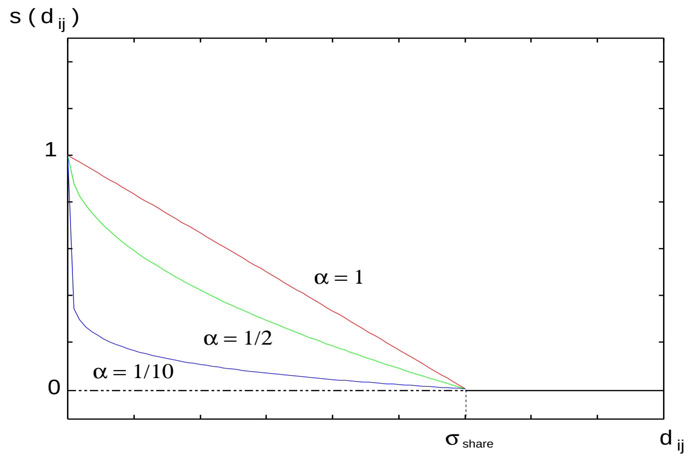


Figure 26: A shape dependency of $s(d_{ij})$ on α.

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,

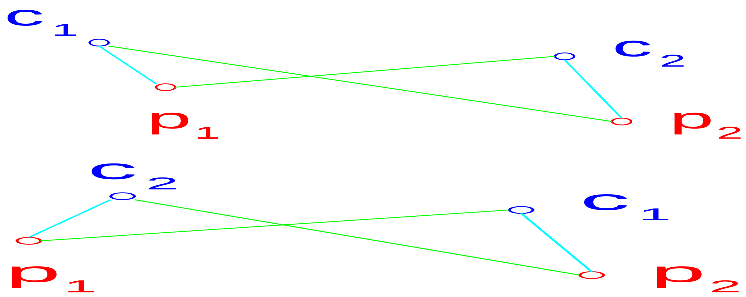


Figure 27: A typical two cases of distance between parents and children.

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* [4]

$$m_{n+1}(\mathbf{x}) = m_n(\mathbf{x}) \cdot g(\mathbf{x}, \mathbf{s}_n) \tag{14}$$

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

$$g(\mathbf{x}, \mathbf{s}_n) = \begin{cases} (d_{xs}/r)^\alpha & \text{if } d_{xs} < r \\ 1 & \text{otherwise} \end{cases} \tag{15}$$

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.*

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

$$g_e(x, s) = \begin{cases} \exp((\log m(x, s)) \cdot (r - d_{xs}/r)) & \text{if } d_{xs} < r \\ 0 & \text{otherwise} \end{cases} \tag{13}$$

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

16 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.

· 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*.

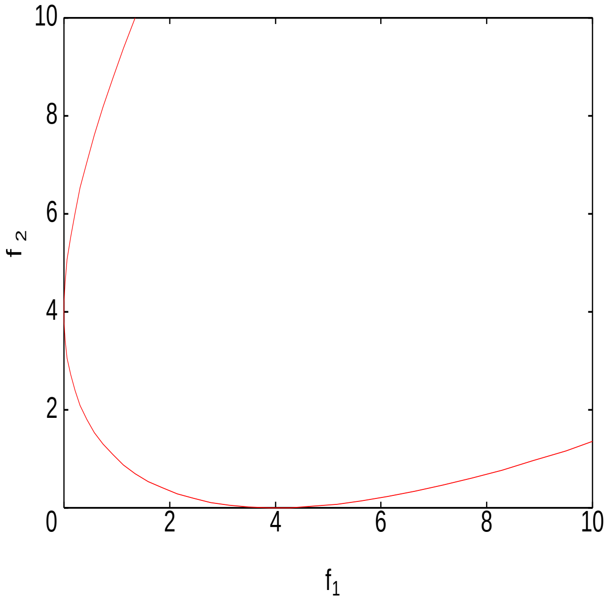


Figure 28: Trade-off space for $f_1(x) = x^2$ and $f_2(x) = (x - 2)^2$.

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

$$f(\mathbf{x}) = \sum_{i=1}^{N} w_i f_i(\mathbf{x}) \tag{16}$$

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.* [5]

[5] 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.

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.*

16.1 Robot Navigation Revisit – Mars Lover's Exploration

17 Evolution of Sturcture — From a Simple Structure to Complex Ones

17.1 Finite State Machine

Figure 29: Trade-off space for $f_1(x) = x^2$ and $f_2(x) = (x - 2)^2$.

18 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))

18.1 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.

18.2 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))

18.3 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.

18.4 Examples

18.4.1 Rule-base

18.4.2 Knap-sack Problem — revisited

18.4.3 Advantage on what?

18.5 Growing NN structure by m-GA

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

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

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

19 EP and ES — variation of GP but older than it

20 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.*

20.1 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.

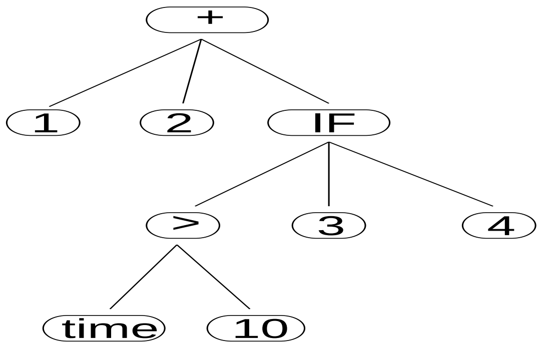
20.2 Evolution of program.

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

20.2.1 An Example of Tree represantation of a program

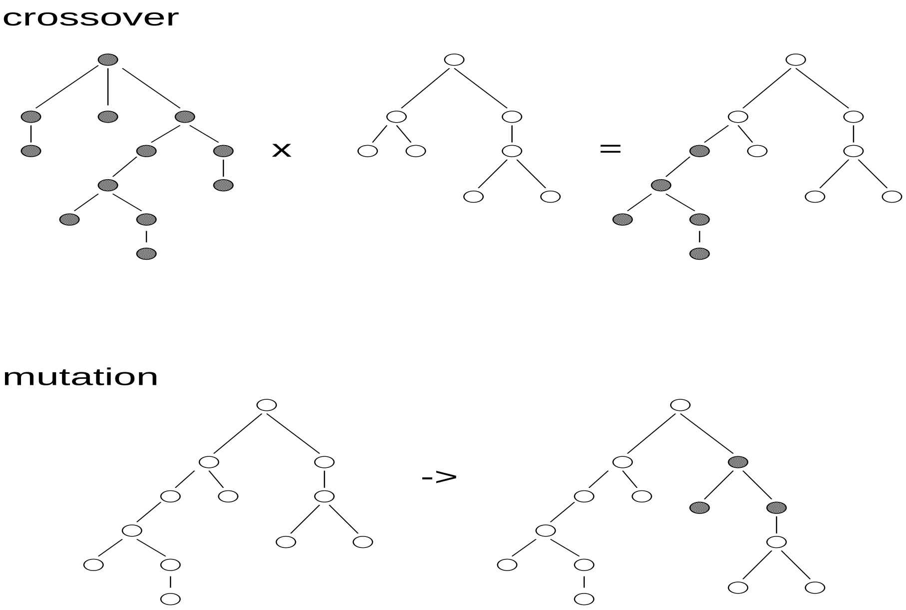
Program in LISP can be represent by a tree. The following is a simple example of one instruction from LISP program and its tree structure.

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



20.2.2 Crossover and Mutation

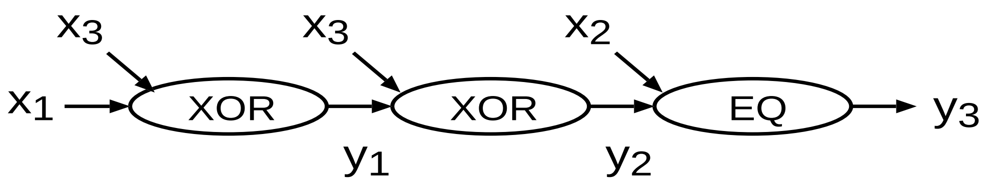
We now look at how we crossover and mutate two trees. See the figure below.

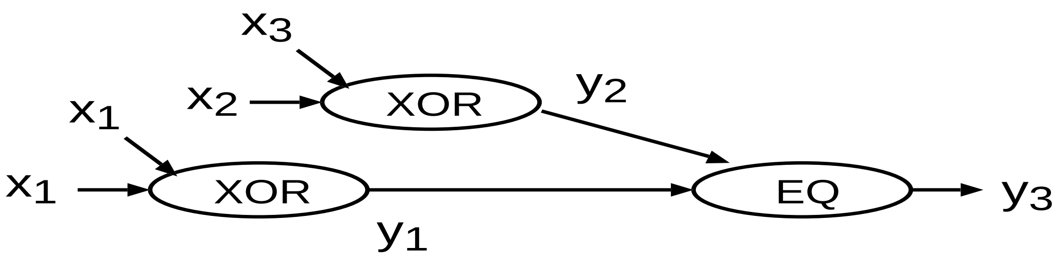


20.3 Eolution of hardware

20.3.1 Even-n-parity

Sometimes, we need automatic error-detection coding. Assume we need $(n-1)$ binary bits for our encoding. Then in order to detect an accidental assignment of a code we add one additioanl bit called parity-check-bit. 0 or 1 will be assigned so that total number of bit 1 becomes even. Hence we can know the incorrectly coded line, though some lucky mistake can avoid this detection.





21 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.

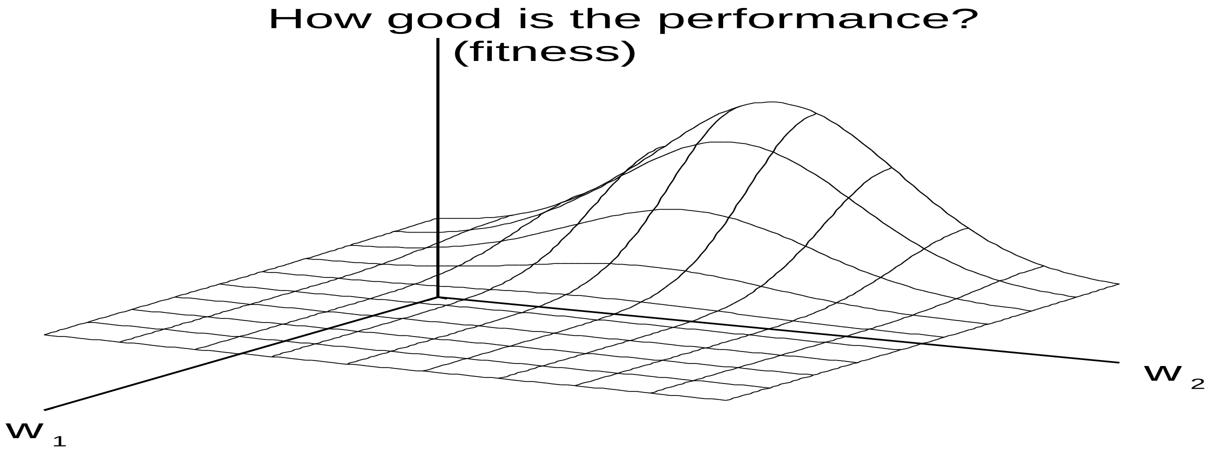


Figure 30: A conceptual plot of fitness value defined on a fictitious 2-dimensional space.

21.1 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)

21.2 A population of hill-climbers

21.2.1 Evolutionary Programming (EP)

21.2.2 Evolution Strategy (ES) – aiming more effectiveness.

21.3 Peculiar Landscapes

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

21.4 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...

22 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.

22.0.1 Breaking a PIN

Assuming p-bit octal [6]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.

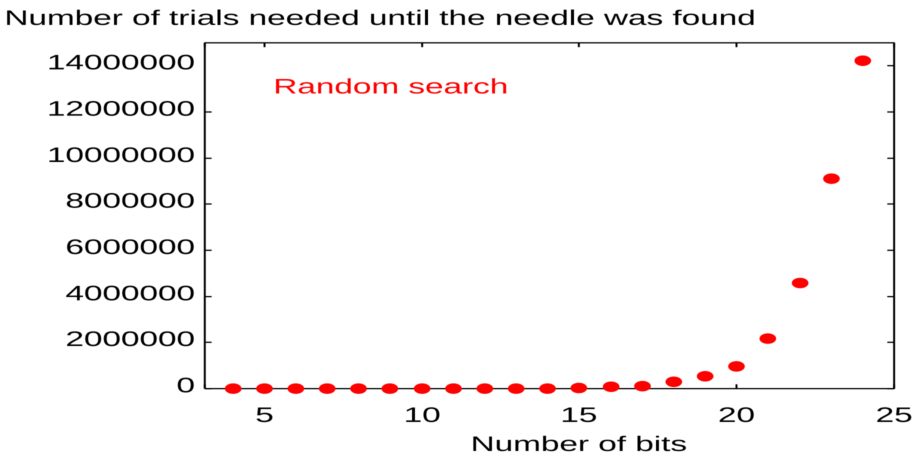


Figure 31: Search will explode exponentially as nuber of bits grows.

22.1 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

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

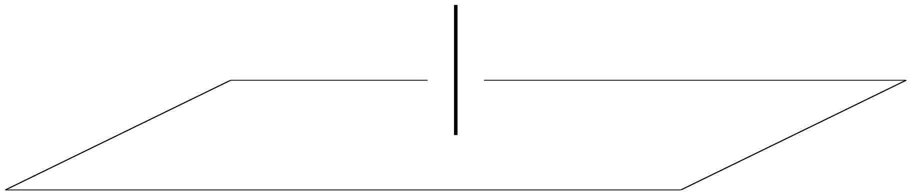


Figure 32: Conceptual figure of a needle in a haystack.

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

23.1 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, (11111111110000000000) 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

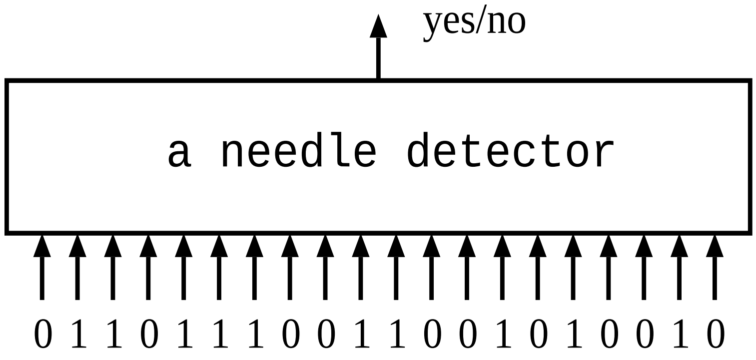


Figure 33: A conceptual black-box to detect a needle.

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*

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

24 Fuzzy Logic

24.1 Fuzzy Controller

An example of our goals.

We now assume x is speed of my car, y is distance to the car in front, and z is how strongly we push brake-pedal. Then let's controll my car with a set of rules, like

- IF x is *high* and y THEN z should be *strong*
- IF x is *medium* and y is *long* THEN z should be *medium*
- IF x is *medium low* or x is *medium* and y is *long* THEN z should be *weak*
- IF x is *low* or x is *medium low* and y is *short* or y is *medium hort* THEN z should be *medium weak*
- etc.

Then the results will be plotted like in the Figure below.

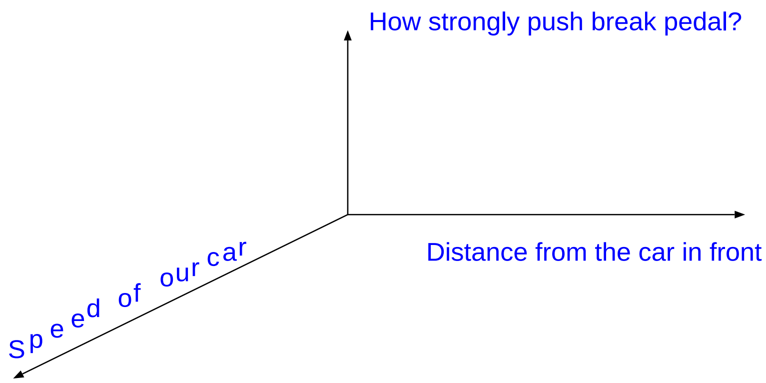


Figure 34: An example of the goal of Fuzzy controller

For example, what should we react when x = 80km/h and y = 40m?

Fuzzy set vs. Crisp set

- $0 < x < 10$
- x=12
- $\{x$ is much smaller than $10\}$
- {x is close to 12}
- Beer is either of {very-cold, cold, not-so-cold, warm}

Membership function

How x is likely to be A is expressed by a function called *membership function* $\mu_A(x)$.
{x is close to 12}

$$\mu(x) = \frac{1}{1 + (x - 12)^2} \tag{17}$$

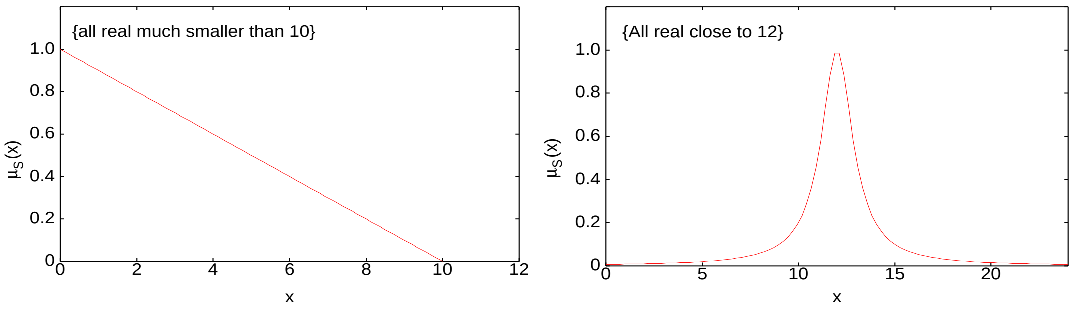


Figure 35: Examples of membership function $\{x$ is much smaller than $10\}$ (right) and {x is close to 12} (left).

AND and OR in Fuzzy Logic

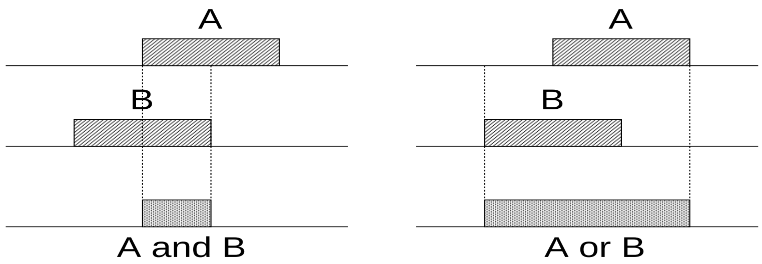


Figure 36: AND and OR in crisp set.

In Fuzzy Logic, the membership of $AandB$ and $AorB$ are specified in various way, but most popular is as

$$\mu_{AandB}(x) = \min\{\mu_A(x), \mu_B(x)\} \tag{18}$$

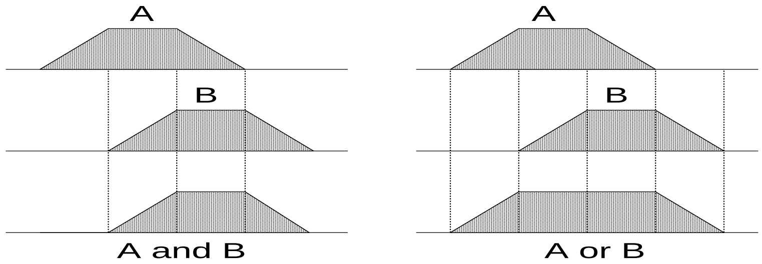


Figure 37: AND and OR in crisp set.

and

$$\mu_{AorB}(x) = \max\{\mu_A(x), \mu_B(x)\}, \tag{19}$$

respectively.
Then, how we visualize membership function unles the function is defined more than one as above two examples?

• Very cold or cold beer.

(μ is defined on temperature)

Let's try ...

looks strange isn't it? So, many another defenition has been proposed so far, *e.g.* Lucasiewics

$$\mu_{AandB}(x) = \max\{1, \mu_A(x) + \mu_B(x)\} \tag{20}$$

and

$$\mu_{AorB}(x) = \min\{1, \mu_A(x) + \mu_B(x)\}, \tag{21}$$

respectively.

bullet Young and tall.

1. 3-D graphic (z = μ is defined on x = age and y = height) 2. Matrix representation

IF-Then rule in Fuzzy Logic

In Fuzzy Logic, the membership of *IF A Then B* is specified also in many way. Here, let's take it as

Mamdani's proposal

$$\mu_{A \to B}(x) = \min\{\mu_A(x), \mu_B(x)\} \tag{22}$$

Larsen's proposal

$$\mu_{A \to B}(x) = \mu_A(x) \times \mu_B(x) \tag{23}$$

24.1.1 Towards controlling a machine

Example 1
IF x = HIGH then z = STRONG

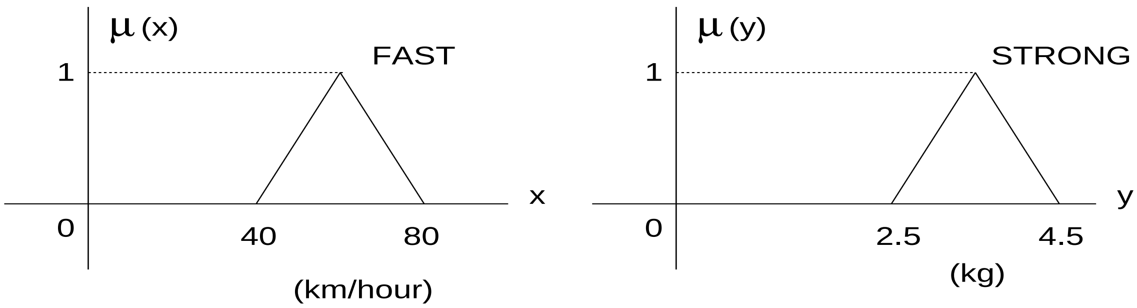


Figure 38: Examples of membership function {x is much smaller than 10} (right) and {x is close to 12} (left).

x \ y	2.5	3.0	3.5	4.0	4.5
40	0.0	0.0	0.0	0.0	0.0
50	0.0	0.5	0.5	0.5	0.0
60	0.0	0.5	1.0	0.5	0.0
70	0.0	0.5	0.5	0.5	0.0
80	0.0	0.0	0.0	0.0	0.0

Figure 39: Examples of membership function {x is much smaller than 10} (right) and {x is close to 12} (left).

Example 2 ... two rules of 1 element
R_1: IF x = slow then z = high

R_2: IF x = midium then z = midium

Example 2 ... abain 1 rule but 2 elements
R_1: IF x = slow AND y = medium THEN z = high

no more possible graphical or matrics but...
output fuzzy set when x = 40 is like ...
It's not normal

24.2 Fuzzy and GA

evalving membership function

- Triangle membership functions
- Gaussian

chromosome corresponding to rule

25 Swarm Inteligence

25.1 Ant Colony Optimization

25.1.1 Application to Data-mining

Setosa				Versicolor				Virginica			
x_1	x_2	x_3	x_4	x_1	x_2	x_3	x_4	x_1	x_2	x_3	x_4
0.65	0.80	0.20	0.08	0.89	0.73	0.68	0.56	0.80	0.75	0.87	1.00
0.62	0.68	0.20	0.08	0.81	0.73	0.65	0.60	0.73	0.61	0.74	0.76
0.59	0.73	0.19	0.08	0.87	0.70	0.71	0.60	0.90	0.68	0.86	0.84
0.58	0.70	0.22	0.08	0.70	0.52	0.58	0.52	0.80	0.66	0.81	0.72
0.63	0.82	0.20	0.08	0.82	0.64	0.67	0.60	0.82	0.68	0.84	0.88
0.68	0.89	0.25	0.16	0.72	0.64	0.65	0.52	0.96	0.68	0.96	0.84
0.58	0.77	0.20	0.12	0.80	0.75	0.68	0.64	0.62	0.57	0.65	0.68
0.63	0.77	0.22	0.08	0.62	0.55	0.48	0.40	0.92	0.66	0.91	0.72
0.56	0.66	0.20	0.08	0.84	0.66	0.67	0.52	0.85	0.57	0.84	0.72
0.62	0.70	0.22	0.04	0.66	0.61	0.57	0.56	0.91	0.82	0.88	1.00
0.68	0.84	0.22	0.08	0.63	0.45	0.51	0.40	0.82	0.73	0.74	0.80
0.61	0.77	0.23	0.08	0.75	0.68	0.61	0.60	0.81	0.61	0.77	0.76
0.61	0.68	0.20	0.04	0.76	0.50	0.58	0.40	0.86	0.68	0.80	0.84
0.54	0.68	0.16	0.04	0.77	0.66	0.68	0.56	0.72	0.57	0.72	0.80
0.73	0.91	0.17	0.08	0.71	0.66	0.52	0.52	0.73	0.64	0.74	0.96
0.72	1.00	0.22	0.16	0.85	0.70	0.64	0.56	0.81	0.73	0.77	0.92
0.68	0.89	0.19	0.16	0.71	0.68	0.65	0.60	0.82	0.68	0.80	0.72
0.65	0.80	0.20	0.12	0.73	0.61	0.59	0.40	0.97	0.86	0.97	0.88
0.72	0.86	0.25	0.12	0.78	0.50	0.65	0.60	0.97	0.59	1.00	0.92
0.65	0.86	0.22	0.12	0.71	0.57	0.57	0.44	0.76	0.50	0.72	0.60
0.68	0.77	0.25	0.08	0.75	0.73	0.70	0.72	0.87	0.73	0.83	0.92
0.65	0.84	0.22	0.16	0.77	0.64	0.58	0.52	0.71	0.64	0.71	0.80
0.58	0.82	0.14	0.08	0.80	0.57	0.71	0.60	0.97	0.64	0.97	0.80
0.65	0.75	0.25	0.20	0.77	0.64	0.68	0.48	0.80	0.61	0.71	0.72
0.61	0.77	0.28	0.08	0.81	0.66	0.62	0.52	0.85	0.75	0.83	0.84
0.63	0.68	0.23	0.08	0.84	0.68	0.64	0.56	0.91	0.73	0.87	0.72
0.63	0.77	0.23	0.16	0.86	0.64	0.70	0.56	0.78	0.64	0.70	0.72
0.66	0.80	0.22	0.08	0.85	0.68	0.72	0.68	0.77	0.68	0.71	0.72
0.66	0.77	0.20	0.08	0.76	0.66	0.65	0.60	0.81	0.64	0.81	0.84
0.59	0.73	0.23	0.08	0.72	0.59	0.51	0.40	0.91	0.68	0.84	0.64
0.61	0.70	0.23	0.08	0.70	0.55	0.55	0.44	0.94	0.64	0.88	0.76
0.68	0.77	0.22	0.16	0.70	0.55	0.54	0.40	1.00	0.86	0.93	0.80
0.66	0.93	0.22	0.04	0.73	0.61	0.57	0.48	0.81	0.64	0.81	0.88
0.70	0.95	0.20	0.08	0.76	0.61	0.74	0.64	0.80	0.64	0.74	0.60
0.62	0.70	0.22	0.04	0.68	0.68	0.65	0.60	0.77	0.59	0.81	0.56
0.63	0.73	0.17	0.08	0.76	0.77	0.65	0.64	0.97	0.68	0.88	0.92
0.70	0.80	0.19	0.08	0.85	0.70	0.68	0.60	0.80	0.77	0.81	0.96

(to be cont'd to the next page)

(cont'd)

Setosa				Versicolor				Virginica			
x_1	x_2	x_3	x_4	x_1	x_2	x_3	x_4	x_1	x_2	x_3	x_4
0.62	0.70	0.22	0.04	0.80	0.52	0.64	0.52	0.81	0.70	0.80	0.72
0.56	0.68	0.19	0.08	0.71	0.68	0.59	0.52	0.76	0.68	0.70	0.72
0.65	0.77	0.22	0.08	0.70	0.57	0.58	0.52	0.87	0.70	0.78	0.84
0.63	0.80	0.19	0.12	0.70	0.59	0.64	0.48	0.85	0.70	0.81	0.96
0.57	0.52	0.19	0.12	0.77	0.68	0.67	0.56	0.87	0.70	0.74	0.92
0.56	0.73	0.19	0.08	0.73	0.59	0.58	0.48	0.73	0.61	0.74	0.76
0.63	0.80	0.23	0.24	0.63	0.52	0.48	0.40	0.86	0.73	0.86	0.92
0.65	0.86	0.28	0.16	0.71	0.61	0.61	0.52	0.85	0.75	0.83	1.00
0.61	0.68	0.20	0.12	0.72	0.68	0.61	0.48	0.85	0.68	0.75	0.92
0.65	0.86	0.23	0.08	0.72	0.66	0.61	0.52	0.80	0.57	0.72	0.76
0.58	0.73	0.20	0.08	0.78	0.66	0.62	0.52	0.82	0.68	0.75	0.80
0.67	0.84	0.22	0.08	0.65	0.57	0.43	0.44	0.78	0.77	0.78	0.92
0.63	0.75	0.20	0.08	0.72	0.64	0.59	0.52	0.75	0.68	0.74	0.72

25.2 Job Shop Scheduling Problem (JSSP)

We assume we have m machines $M_1, \cdots, M_m$ and n jobs $J_1, \cdots, J_n$ Each job J_i is made up of n_j operations $O_{ij}(i = 1, \cdots n_j)$ which have to be processed in the order $O_{1j}, O_{2j}, \cdots O_{nj}$. Operation O_{ij} has to be processed without preemption on a dedicated machine $\mu_{ij} \in M_1, \cdots M_m$ for timu duration p_{ij}.

Each task must be processed on a single specified machine, and each job visits each machine exactly once. There is a predefined ordering of the tasks within a job. A machine can process only one task at a time. There are no set-up times, no release dates and no due dates. The makespan is the time from the beginning of the first task to start to the end of the last task to finish. The aim is to find start times for each task such that the makespan is minimised.

The operation O_{jr} requires the exclusive use of M_r for an uninterrupted period of time p_{jr}

25.2.1 An example

25.3 Artificial Honey Bee Colony

Option of the bee who returns with a nector from flower

- Decide the site was not good. Hence, do not try dance but observe other dances and choose and follow.

- Continue to explore the same flower without giving dance and hence no followers.
- Dance & recruite followers.

The dance is called *Waggle dance*. It includes the information about (i) quality of flowers; (ii) distance to the flowers; (ii) direction to the flowers.

26 Reinforcement Learning

Reinforcement Learning is a machine learning approach to solve goal-oriented problems in which agent repeats a decision making by choosing one of the pre-defiened action according to the situatin which the agent come accross from one to the next.
To simply put, might not so simply though, Reinforcement Learning is how the agent learn to attain the goal by after sequential stochastic decisions, that is, by repeating changing situation and deciding aciton.

State, Action, Policy, and Reward/Penalty

Thus, the Reinforcement Learning is made up of (i) *situation* – more frequently called a *state*, that is, a state of environment; (ii) action; (iii) *policy* – a mapping from situation to action; (iv) *reward/penalty* which is given to the agent according to the acciton chosen in the next situation.

In the lecture chess, cart-pole problem, car-on-the-hill problem, jeep problem and so on will be given as examples.

Value of state

Each state has a value called a value of state which is is defined as the sum of the reinforcements received when starting in that state and following some fixed policy to a terminal state.

Value Function

The value function is a mapping from states to state values

Optimal Policy

Then the optimal policy would therefore be the mapping from states to actions that maximizes the sum of the reinforcements when starting in an arbitrary state and performing actions until a terminal state is reached.

The goal of the agent is to find a policy that maximizes the total reward received over time To obtain such a policy, let's define *expected discounted future reward* starting in state s taking action a and following π,

$$Q^{\pi}(s,a) = E_{\pi}\langle r_{t+1} + \gamma \cdot r_{t+2} + \gamma^2 \cdot r_{t+3} + \cdots | s_t = s, a_t = a\rangle. \tag{24}$$

And Optimal Q is defined as:

$$Q^{\star}(s,a) = \max_{\pi} Q^{\pi}(s,a). \tag{25}$$

The question is how we obtain the optimul value of $Q(s,a)$. We now look at popular two options. The first one is called *Q-learning* and renewed according to

$$Q(s_t,a_t) := Q(s_t,a_t) + \alpha\{r_{t+1} + \gamma \cdot \max_{a' \in A} Q(s_{t+1},a') - Q(s_t,a_t)\}. \tag{26}$$

The second option is called *SALSA*, and renewal is by:

$$Q(s_t,a_t) := Q(s_t,a_t) + \alpha\{r_{t+1} + \gamma \cdot Q(s_{t+1},a_{t+1}) - Q(s_t,a_t)\} \tag{27}$$

in both of which the action a_t at each step t is chosen at random with probability ϵ , and the action with highest Q-Value with probability $1-\epsilon$. This is called an ϵ-greedy choice of action.