Analysis and Systems of Big Data

Pranav Parameshwaran
COE17B036

Hyperlink Induced Topic Search

Hyperlink Induced Topic Search (HITS) Algorithm is a Link Analysis Algorithm that rates webpages. This algorithm is used to the web link-structures to discover and rank the webpages relevant for a particular search.
HITS uses hubs and authorities to define a recursive relationship between web pages. Before understanding the HITS Algorithm, we first need to know about Hubs and Authorities.

- Given a query to a Search Engine, the set of highly relevant web pages are called **Roots**. They are potential **Authorities**.
- Pages which are not very relevant but point to pages in the Root are called **Hubs**. Thus, an Authority is a page that many hubs link to whereas a Hub is a page that links to many authorities.

Pseudo-Code

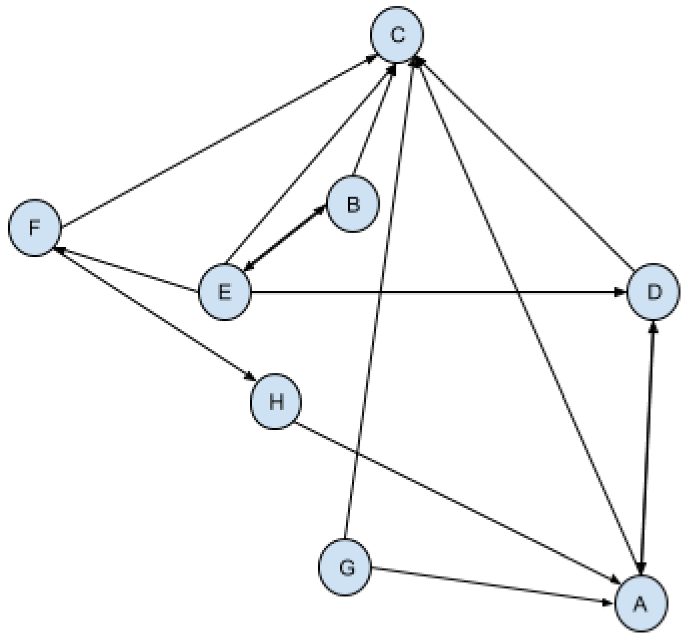
```
1. Let the number of iterations be k.
2. Each node is assigned a Hub Score =1 and an Authority Score=1
3.  Repeat k times {
        ● Hub Update : Each node's Hub score=∑(Authority Score of each node it points to).
        ● Authority Update : Each node's Authority Score = ∑(Hub Score of each node pointing to it).
```

- Normalize the scores by dividing each hub score by square root of the sum of squares of all the Hub Scores, and dividing each Authority score by square root of the sum of squares of all the Authority Scores. (optional)

}

Trace

Let us consider the following graph :



On executing HITS algorithm for k=3 times (without Normalization)

Initialization

Pages	Hub Score	Authority Score
A	1	1
B	1	1
C	1	1
D	1	1
E	1	1
F	1	1
G	1	1
H	1	1

After iteration 1,

Pages	Hub Score	Authority Score
A	1	3
B	2	2
C	1	4
D	2	2
E	4	1
F	1	1
G	2	0
H	1	1

After iteration 2,

Pages	Hub Score	Authority Score
A	2	4
B	5	6
C	3	7
D	6	5
E	9	2
F	1	4
G	7	0
H	3	1

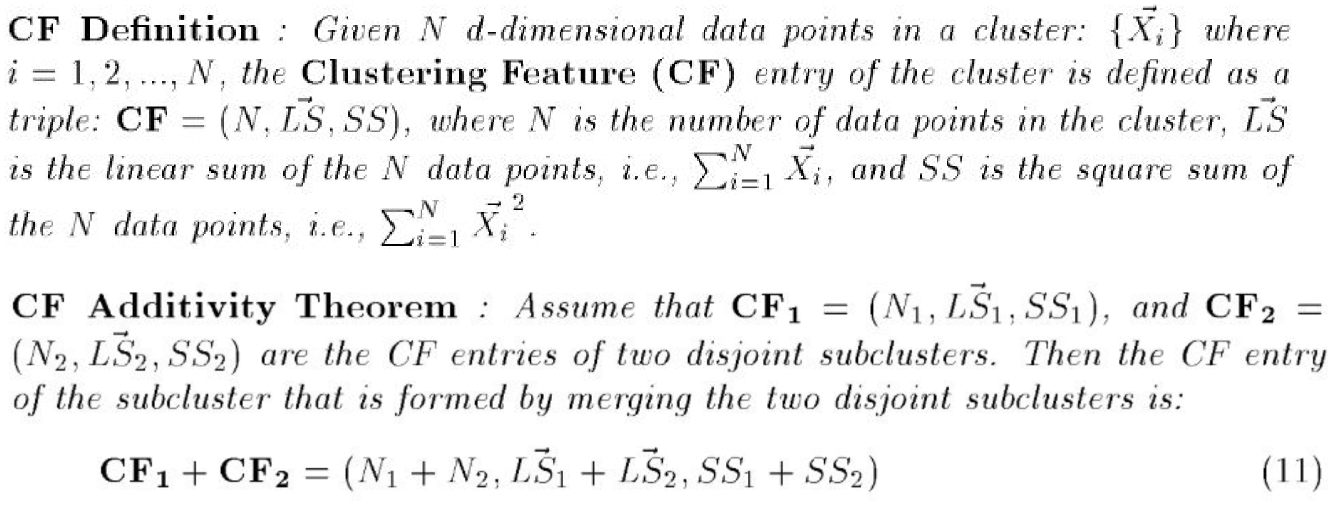
After iteration 3

Pages	Hub Score	Authority Score
A	5	13
B	9	15
C	4	27
D	13	11
E	22	5
F	1	9
G	11	0
H	4	3

BIRCH

BIRCH (balanced iterative reducing and clustering using hierarchies) is an unsupervised data mining algorithm used to perform hierarchical clustering over particularly large data-sets.An advantage of BIRCH is its ability to incrementally and dynamically cluster incoming, multi-dimensional metric data points in an attempt to produce the best quality clustering for a given set of resources (memory and time constraints). In most cases, BIRCH only requires a single scan of the database.

BIRCH summarizes the information contained in dense regions as Clustering Feature (CF) entries.



CF Definition : *Given N d-dimensional data points in a cluster: $\{\vec{X_i}\}$ where $i = 1, 2, ..., N$, the* **Clustering Feature (CF)** *entry of the cluster is defined as a triple:* $\mathbf{CF} = (N, \vec{LS}, SS)$, *where N is the number of data points in the cluster, $\vec{LS}$ is the linear sum of the N data points, i.e., $\sum_{i=1}^{N} \vec{X_i}$, and SS is the square sum of the N data points, i.e., $\sum_{i=1}^{N} \vec{X_i}^2$.*

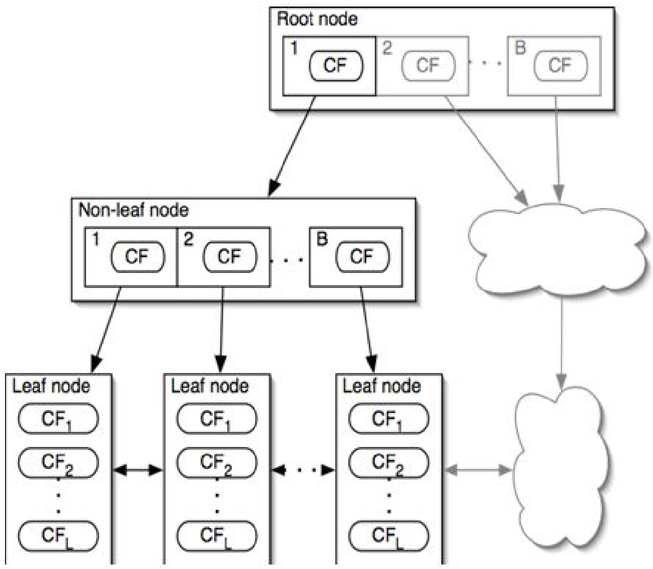
CF Additivity Theorem : *Assume that* $\mathbf{CF_1} = (N_1, \vec{LS_1}, SS_1)$, *and* $\mathbf{CF_2} = (N_2, \vec{LS_2}, SS_2)$ *are the CF entries of two disjoint subclusters. Then the CF entry of the subcluster that is formed by merging the two disjoint subclusters is:*

$$\mathbf{CF_1} + \mathbf{CF_2} = (N_1 + N_2, \vec{LS_1} + \vec{LS_2}, SS_1 + SS_2) \qquad (11)$$

CF-tree

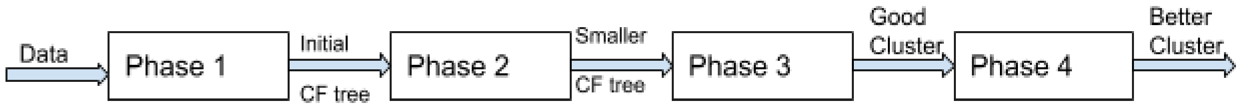
The structure of CF Tree is given below:

- Each non-leaf node has at most B entries.
- Each leaf node has at most L CF entries which satisfy threshold T, a maximum diameter of radius.
- P(page size in bytes) is the maximum size of a node.
- Compact: each leaf node is a subcluster, not a data point.



Algorithm

1. *Phase 1* : Load data into memory
   Scan DB and load data into memory by building a CF Tree. If memory is exhausted, rebuild the tree from the leaf node.
2. *Phase 2* : Condense data
   Resize the data set by building a smaller CF tree
3. *Phase 3* : Global clustering
   Use existing clustering algorithms (e.g, k-means,HC) on CF entries.
4. *Phase 4* : Cluster refining
   Fixes the problem with CF trees where same valued data points may be assigned to different leaf entries.(Refining is optional)



Applications

1. Pixel classification in images
2. Image compression
3. Works with very large data sets

Disadvantage - it handles only numeric data.

Trace

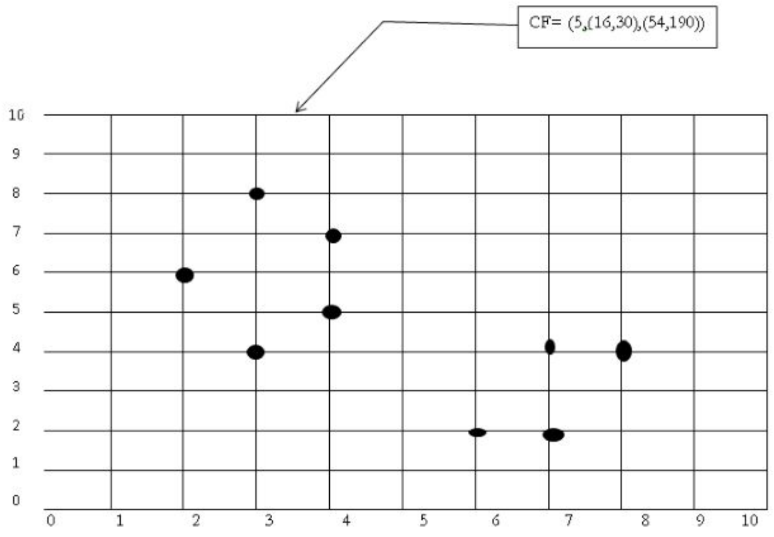
CF = (N,LS,SS)

Data = (3,4); (2,6); (4,5); (4,7); (3,8)

N: number of data points

LS: $\sum_{i=1}^{N} X_i$

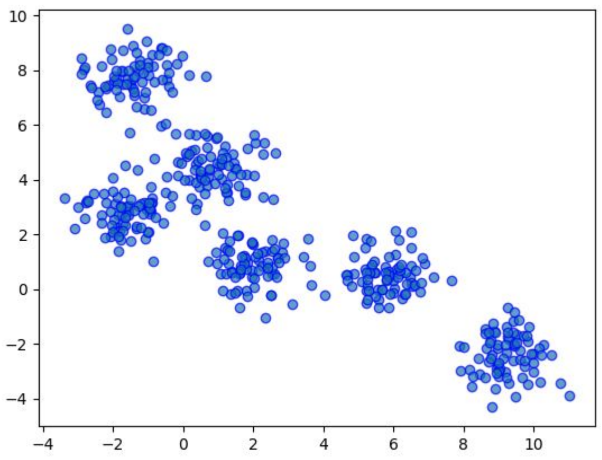
SS: $\sum_{i=1}^{N} X_i^2$

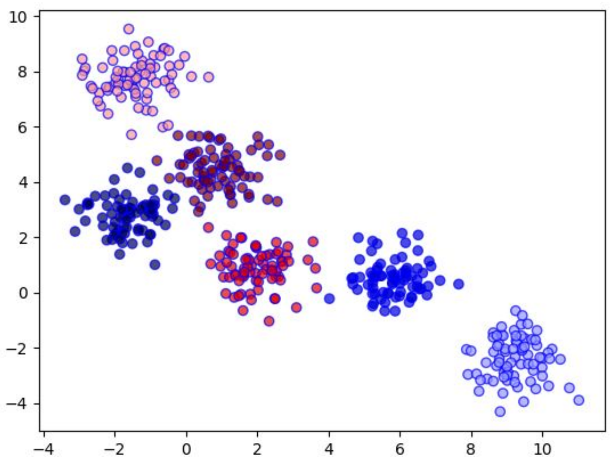


N=5

NS= (16, 30) i.e. 3+2+4+4+3=16 and 4+6+5+7+8=30

SS=(54,190)=$3^2+2^2+4^2+4^2+3^2=54$ and $4^2+6^2+5^2+7^2+8^2=190$





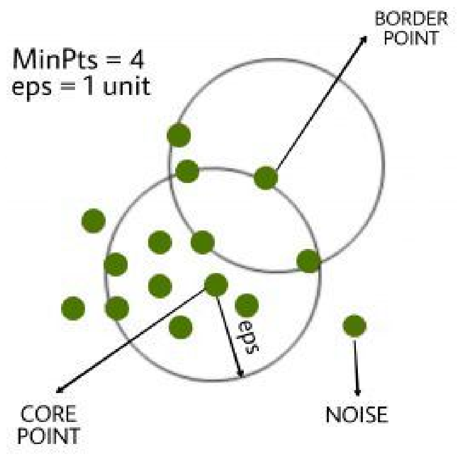
DBSCAN

There are 2 important parameters in this method:

a. Eps - defines neighbourhood around a datapoint.
b. MinPts - Min. number of neighbours within eps radius.

In this algorithm we have 3 types of data points:

1. **Core Point :** A point is a core point if it has more than MinPts points within eps.
2. **Border Point :** A point which has fewer than MinPts within eps but it is in the neighbourhood of a core point.
3. **Noise or outlier :** A point which is not a core or border point.



Algorithm

1. Find all the neighbor points within eps and identify the core points or visited with more than MinPts neighbors.
2. For each core point if it is not already assigned to a cluster, create a new cluster.
3. Find recursively all its density connected points and assign them to the same cluster as the core point.
   A point a and b are said to be density connected if there exists a point c which has a sufficient number of points in its neighbors and both the

points a and b are within the eps distance. This is a chaining process. So, if b is neighbor of c, c is neighbor of d, d is neighbor of e, which in turn is neighbor of a implies that b is neighbor of a.

4. Iterate through the remaining unvisited points in the dataset. Those points that do not belong to any cluster are noise.

PseudoCode

```
function DBSCAN(dataset, eps, MinPts)
{
    C=1
    for each unvisited point p in dataset
    {
        Mark 'p' as visited.
        Neighbours N = find the neighbouring points of p.
        if |N| >= MinPts:
            N = N ∪ N'
            if p' is not a member of any cluster:
                Add p' to cluster C
    }
}
```

GA based approach to solve Travelling Salesman problem

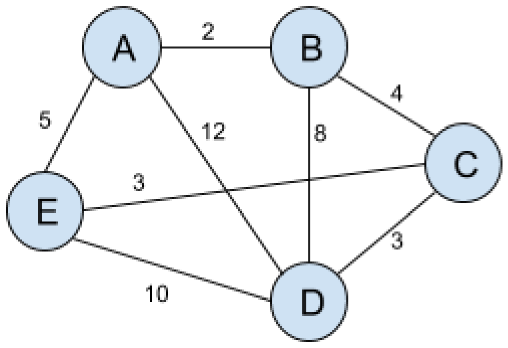
Approach: In the following implementation, cities are taken as genes, a string generated using these characters is called a chromosome, while a fitness score which is equal to the path length of all the cities mentioned, is used to target a population.

Fitness Score is defined as the length of the path described by the gene. Lesser the path length fitter is the gene. The fittest of all the genes in the gene pool survive the population test and move to the next iteration. The number of iterations depends upon the value of a cooling variable. The value of the cooling variable keeps on decreasing with each iteration and reaches a threshold after a certain number of iterations.

Pseudo-code

```
Initialize procedure GA{
    Set cooling parameter = 0;
    Evaluate population P(t);
    While( Not Done ){
        Parents(t) = Select_Parents(P(t));
        Offspring(t) = Procreate(P(t));
        p(t+1) = Select_Survivors(P(t), Offspring(t));
        t = t + 1;
    }
}
```

The dataset which we will be using is the below :



We will mark A → 0, B → 1 and so on

Initial population:

GNOME	FITNESS VALUE
043210	24
023410	∞
031420	∞
034210	31
043210	24
023140	∞
032410	∞
012340	24

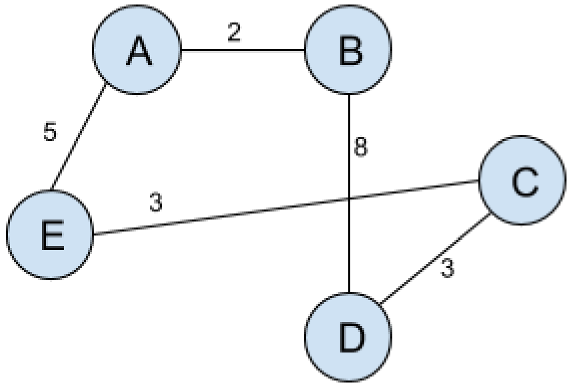
012340	24
032410	∞

After 5 generations we get the following population pool :

Current temp: 6561
Generation 5

GNOME	FITNESS VALUE
043210	24
042310	21
042310	21
013240	21
042310	21
034210	31
013240	21
042310	21
024310	∞
024310	∞

We see that 042310 has Fitness value as 21 and also other combinations also have the same score, hence there are more than one correct solution. If we consider the 042310 path we get the following graph.



Hence the traveller covers all cities in minimum distance and also visits all cities exactly once.

Regression Models

Linear Regression

Linear regression is a **linear model**, e.g. a model that assumes a linear relationship between the input variables (x) and the single output variable (y). More specifically, that y can be calculated from a linear combination of the input variables (x).

When there is a single input variable (x), the method is referred to as **simple linear regression**. When there are multiple input variables, literature from statistics often refers to the method as **multiple linear regression**.

Different techniques can be used to prepare or train the linear regression equation from data, the most common of which is called Ordinary Least Squares. It is common to therefore refer to a model prepared this way as Least Squares Regression.

Representation

A linear regression line has an equation of the form $\boldsymbol{Y} = \boldsymbol{a} + \sum_{i=1}^{n} \boldsymbol{b}_i \boldsymbol{X}_i$, where ***X*** is the input variable and ***Y*** is the predicted output variable. The slope of the line is ***b***, and ***a*** is the bias coefficient. In higher dimensions when we have more than one input (x), the line is called a plane or a hyper-plane. The representation therefore is the form of the equation and the specific values used for the coefficients.

Update of the coefficients

Learning a linear regression model means estimating the values of the coefficients used in the representation with the data that we have available.

We can apply Gradient Descent for optimizing the values of the coefficients by iteratively minimizing the error of the model on your training data.

This works by starting with random values for each coefficient. The sum of the squared errors are calculated for each pair of input and output values. A learning rate is used as a scale factor and the coefficients are updated in the direction towards minimizing the error. The process is repeated until a minimum sum squared error is achieved or no further improvement is possible.

Gradient descent for Weight W_i (The loss L is calculated as Sum of Squared error ($\hat{y}$-y))

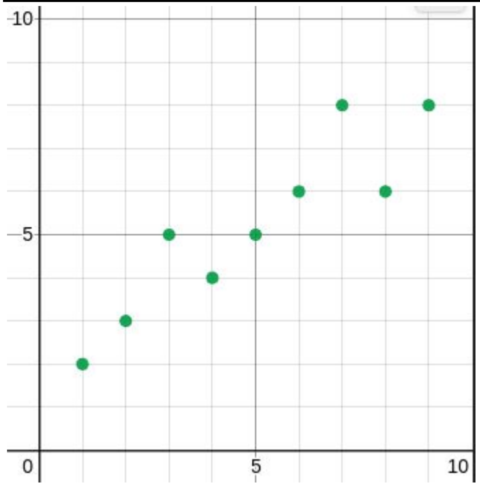
$W_i = W_i - \alpha * \frac{\delta L}{\delta W_i}$

PseudoCode

```
//Here we assume our Input Variable is j Dimensional vector.
function LinearReg(X,Y,lr,epochs)
{       Initialize $\theta_1,\theta_2$ randomly.
        m := num. of training examples
        for t = (1,epochs)
        {
                for i = (1,m)
                        $pred_i = \theta_1 + \theta_2 * X_i$
                $J(\theta_1,\theta_2) = \frac{1}{2*m} \sum_{i=1}^{m} [pred_i - Y_i]^2$
                //Update Step
                $\theta_1 = \theta_1 - lr* \frac{\partial J(\theta 1,\theta 2)}{\partial \theta 1} = \theta_1 - \frac{lr}{m} * \sum_{i=1}^{m} [(pred_i - Y_i)]$ //Bias
                $\theta_j = \theta_j - lr* \frac{\partial J(\theta 1,\theta 2)}{\partial \theta j} = \theta_j - \frac{lr}{m} * \sum_{i=1}^{m} [(pred_i - Y_i)*X^j_i]$//Weights
        }
}
```

Trace

X	Y(true value)	(pred-y)*x
1	2	2-2=0
2	2	3-2=1*2=2
3	5	4-5=-1*3=-3
4	4	5-4=1*4=4
5	5	6-5=1*5=5
6	6	7-6=1*6=6
7	8	8-8=0*7=0
8	6	9-6=3*8=24
9	8	10-8=2*9=18



Let us initialize weights as w1 = 1, w2 = 1, lr = 0.01

Epoch 1:

Pred = [2,3,4,5,6,7,8,9,10]   (pred = w1 + w2*X)

J(w1,w2) = (1/18)*Σ[(Pred - Y)]^2 = 1

W1 = w1 - (0.01/9)*8 ->   w1 = 1-0.00088= 0.9911

W2 = w2 - (0.01/9)*56 -> w2 = 1 - 0.06222 = 0.9377

Now we repeat this for several epochs until we reach convergence.

Non-Linear Regression

Nonlinear regression is a regression in which the dependent or criterion variables are modeled as a **nonlinear function** of model parameters and one or more independent variables. The data are fitted by a method of successive approximations.

Representation

A non-linear regression model has an equation of the form ***Y ~ f(X,b)***, where ***X*** is the input variable and ***Y*** is the predicted output variable. **b** is the coefficients of the input. In higher dimensions when we have more than one input (x), the line is called a plane or a hyper-plane. The representation therefore is the form of the equation and the specific values used for the coefficients.

We shall focus on Logistic Regression here although there are different types of non-linear regression e.g; exponential functions, logarithmic functions, trigonometric functions, power functions, Gaussian functions, and Lorenz curves.
The update rule for any non-linear regression is similar to that of Linear regression (the derivative term changes).

Algorithm

```
Initialize parameters
Repeat for epochs
{
    a. Compute $Y_{pred}$
    b. Calculate the cost function ($J(\theta)$)
    c. Update parameters(Gradient Descent or any other algorithm)
}
```

After updating the parameters we can use that to predict new inputs.

SLIQ

SLIQ is a decision tree classifier that can handle both numeric and categorical attributes. It uses a novel pre-sorting technique in the tree-growth phase. This sorting procedure is integrated with a breadth-first tree growing strategy to enable classification of disk-resident datasets.

SLIQ also uses a new tree-pruning algorithm that is inexpensive, and results in compact and accurate trees.
The combination of these techniques enables SLIQ to scale for large data sets and classify data sets irrespective of the number of classes, attributes, and examples (records), thus making it an attractive tool for data mining.
SLIQ achieves good scalability and performs well for datasets with a large number of examples and attributes.

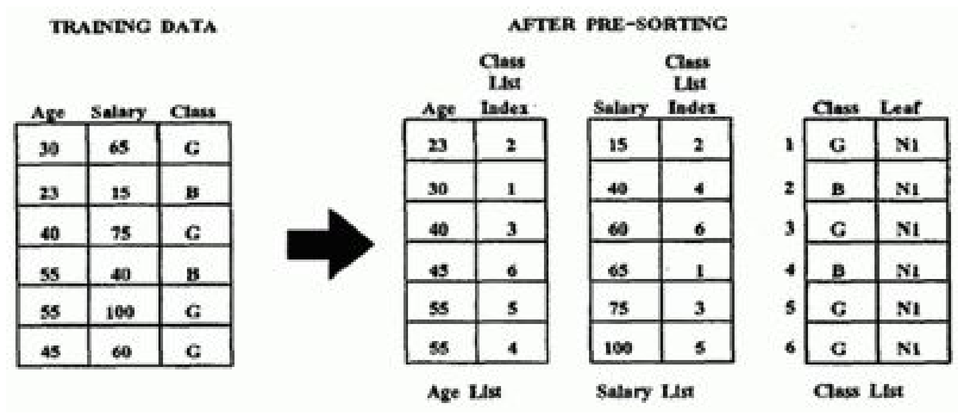
Algorithm

Key Features

1. Tree Classifier, handling both numerical and categorical attributes.
2. Pre-sort numerical attributes before tree has been built
3. Breadth first growing strategy
4. Goodness test -Gini Index
5. Inexpensive tree pruning algorithm based on Minimum Description Length(MDL)

Pre-Sorting

1. Eliminate the need to sort the data at each node.
2. Create sorted list for each numerical attribute
3. Create class list


TRAINING DATA

Age	Salary	Class
30	65	G
23	15	B
40	75	G
55	40	B
55	100	G
45	60	G

AFTER PRE-SORTING

Age	Class List Index
23	2
30	1
40	3
45	6
55	5
55	4

Age List

Salary	Class List Index
15	2
40	4
60	6
65	1
75	3
100	5

Salary List

	Class	Leaf
1	G	N1
2	B	N1
3	G	N1
4	B	N1
5	G	N1
6	G	N1

Class List

```
EvaluateSplits ()
  for each attribute A do
      traverse attribute list of A
      for each value v in the attribute list do
          find the corresponding entry in the class list, and
              hence the corresponding class and the leaf node (say L)
          update the class histogram in the leaf L
          If A is a numeric attribute then
               compute splitting index for test (A ≤ v) for leaf L
      if A is a categorical attribute then
           for each leaf of the tree do
                find subset of A with best split
```

Updating the Class List

```
UpdateLabels ()
    for each attribute A used in a split do
       traverse attribute list of A
       for each value v in the attribute list do
           find the corresponding entry in the class list (say e)
           find the new class c to which v belongs by applying
                the splitting test at node referenced from e
           update the class label for e to c
           update node referenced in e to the child corresponding to the class c
```

Trace

SLIQ Algorithm is as follows

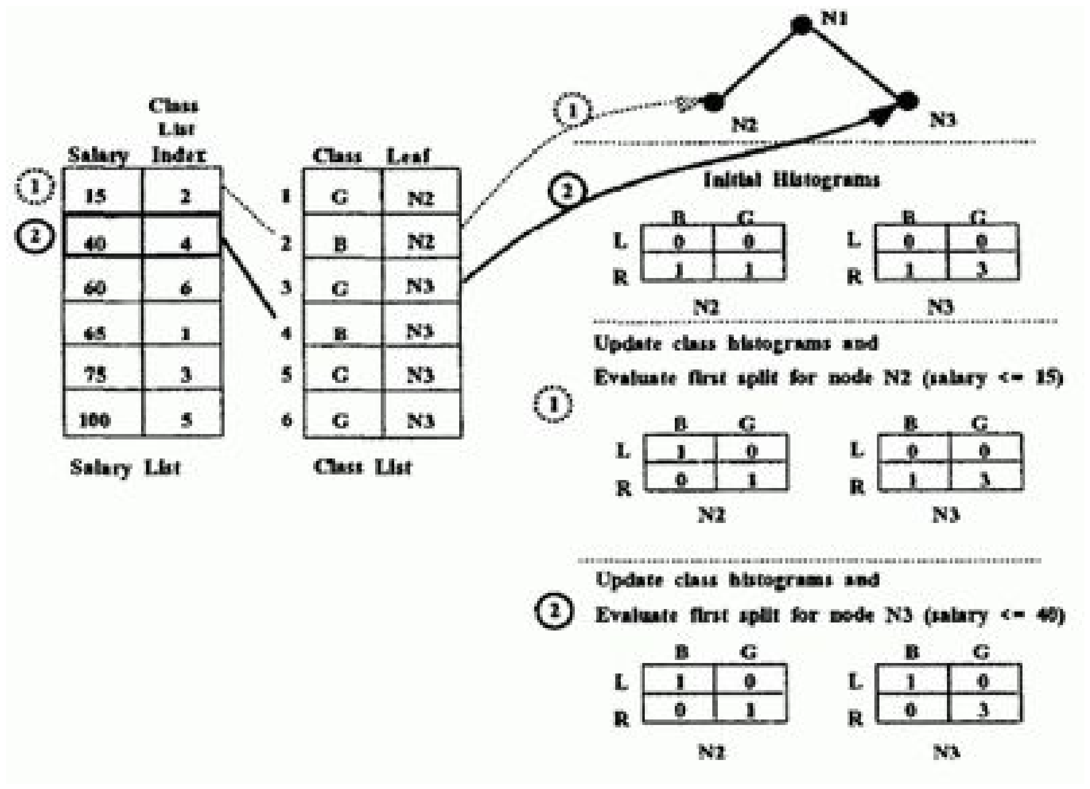
- Start Pre-sorting of the samples.
- Repat For every attribute(till stop criterion not reached)

```
{
    1. Place all nodes into a class histogram.
    2. Start evaluation of the splits.
    3. Choose a split.
    4. Update the decision tree; for each new node update its class list (nodes).
}
```

Processing of Node splits

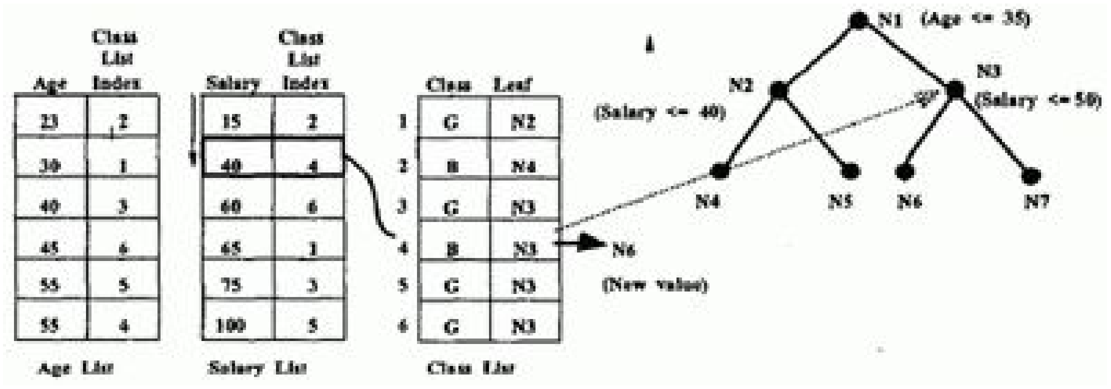
Rather than using a depth-first strategy used in the earlier decision-tree classifiers, we grow trees breadth-first. Consequently, splits for all the leaves of the current tree are simultaneously evaluated in one pass over the data.
To compute the gini splitting-index for an attribute at a node, we need the frequency distribution of class values in the data partition corresponding to the node. The distribution is accumulated in a class histogram attached with each leaf node. For a numeric attribute, the histogram is a list of pairs of the form <class, frequency>. For a categorical attribute, this histogram is a list of triples of the form <attribute value, class, frequency>.

Attribute lists are processed one at a time (recall that the attribute lists can be on disk). For each value v in the attribute list for the current attribute A, we find the corresponding entry in the class list, which yields the corresponding class and the leaf node. We now update the histogram attached with this leaf node. If A is a numeric attribute, we compute at the same time the splitting-index for the test $A \leq v$ for this leaf. If A is a categorical attribute, we wait till the attribute list has been completely scanned and then find the subset of A with the best split. Thus, in one traversal of an attribute list, the best split using this attribute is known for all the leaf nodes. Similarly, with one traversal of all of the attribute lists, the best overall split for all of the leaf nodes is known. The best split test is saved with each of the leaf nodes.


N1
N2
N3
Salary
Class List Index
Class
Leaf
Salary List
Class List
Initial Histograms
Update class histograms and
Evaluate first split for node N2 (salary <= 15)
Update class histograms and
Evaluate first split for node N3 (salary <= 40)

Updating the Class List

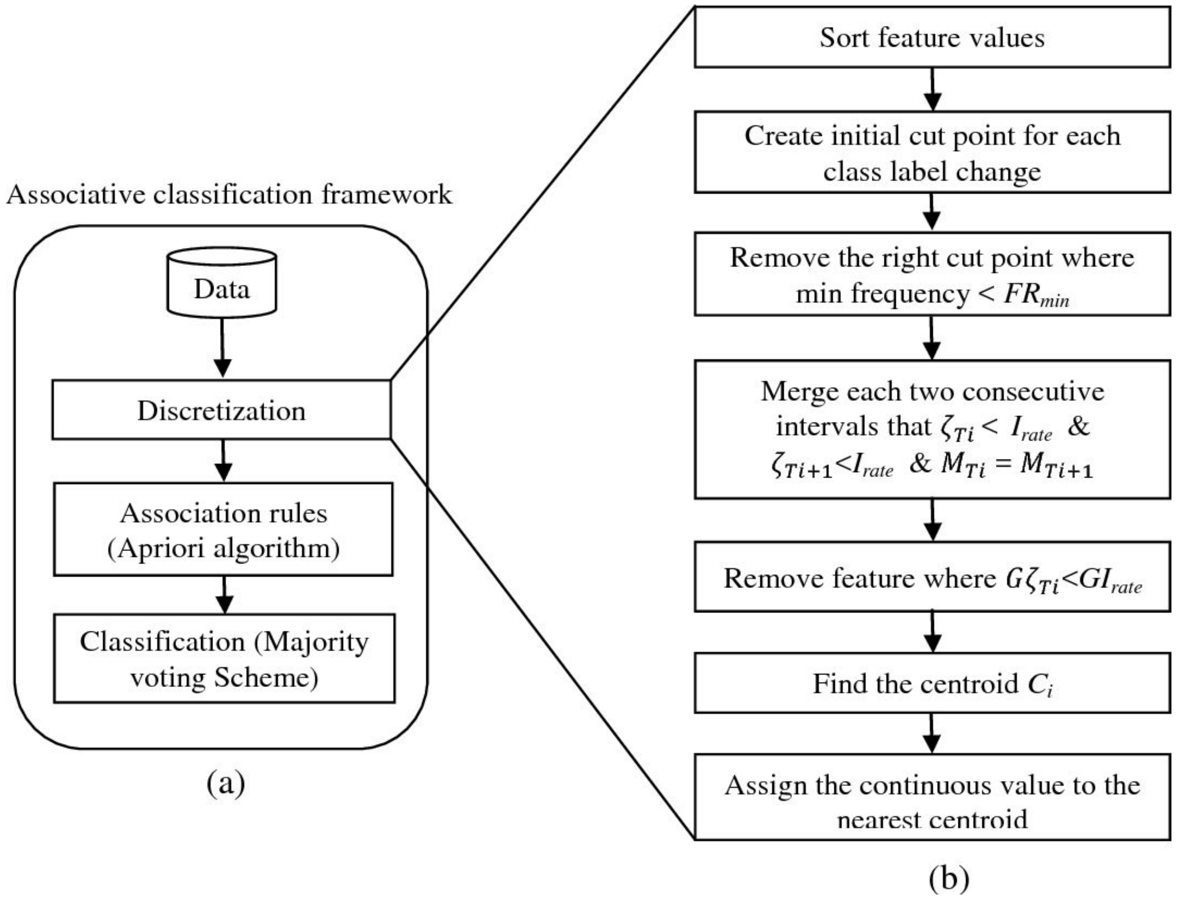
The next step is to create child nodes for each of the leaf nodes and update the class list. Figure 6 gives the update process.The salary attribute list is being traversed and the class list entry (entry 4) corresponding to the salary value of 40 is being updated. First, the leaf reference in the entry 4 of class list is used to find the node to which the example used to belong (N3 in this case). Then, the split selected at N3 is applied to find the new child to which the example belongs (N6 in this case). The leaf reference field of entry 4 in the class list is updated to reflect the new value.



While growing the tree, the above two steps of splitting nodes and updating labels are repeated until each leaf node becomes a pure node (i.e. it contains examples belonging to only one class) and no further splits are required. This optimization can easily be implemented by rewriting condensed lists when the savings from reading smaller lists outweigh the extra cost of writing condensed lists. The important thing to note about pre-sorting and breadth-first growth is that these strategies allow SLIQ to scale for large data sets with no loss in accuracy. This is because the set of splits evaluated with and without pre-sorting is identical. Pre-sorting simply eliminates the task of re-sorting data at each node and removes the restriction that the training set be memory-resident.

ARBC (Association Rule Based Classification)

Association rule based classification is one of the popular data mining techniques. The major advantage is its interpretable results that people can easily adopt for decision-making. The classification framework consists of data discretization, association rule generation, and classification. The discretization step is required to convert numerical features into a categorical format, to make it suitable for association rules mining.
Association rule mining has been studied and applied in many other domains (e.g. credit card fraud, network intrusion detection, genetic data analysis).
In every domain, there is a need to analyze data to identify patterns associating different attributes. Association rule mining addresses this need. Many association rule mining algorithms have been proposed in the data mining literature. Apriori and FP-growth are two of them.


Associative classification framework
Data
Discretization
Association rules (Apriori algorithm)
Classification (Majority voting Scheme)
(a)
Sort feature values
Create initial cut point for each class label change
Remove the right cut point where min frequency < FR_{min}
Merge each two consecutive intervals that $\zeta_{Ti} < I_{rate}$ & $\zeta_{Ti+1} < I_{rate}$ & $M_{Ti} = M_{Ti+1}$
Remove feature where $G\zeta_{Ti} < GI_{rate}$
Find the centroid C_i
Assign the continuous value to the nearest centroid
(b)

<u>Algorithm</u>

1. Create Associate Rules
   In associative classification, the focus is to produce association rules that have only a particular attribute in the consequent. These association rules produced are called class association rules(CARs). Associative classification differs from general association rule mining by introducing a constraint as to the attribute that must appear on the consequent of the rule. The CBA-RG algorithm is an extension of the Apriori algorithm. The goal of this algorithm is to find all rule items of the form <condset, y > where condset is a set of items, and y ∈ Y, where Y is the set of class labels.
2. Classifying based on the rules
   Rule items that have support greater than or equal to min_sup are called frequent rule items, while the others are called infrequent rule items. For all rule items that have the same condset, the one with the highest confidence is selected as the representative of those rule items. The confidence of rule items are calculated to determine if the rule item meets min_conf. The set of rules that is selected after checking for support and confidence is called the classification association rules(CARs).
   In association rule based classification models, rules in the model are ordered as follows:
   • if rule ri has greater confidence than rj, then ri precedes rj, or
   • if ri has the same confidence as rj, then the rule with greater support precede the other, or
   • if ri has the same support and the same confidence as rj, then the rule with then smaller number of items in the antecedent will precede, or
   • if ri has the same support, confidence and antecedent size as rj, then the order between the two rules is random.
   Rules of high confidence are thought to be good for classification.

Different Types of Selection,Cross-Over and Mutation Operations

Selection Methods

Parent Selection is the process of selecting parents which mate and recombine to create off-springs for the next generation. Parent selection is very crucial to the convergence rate of the GA as good parents drive individuals to better and fitter solutions.

However, care should be taken to prevent one extremely fit solution from taking over the entire population in a few generations, as this leads to the solutions being close to one another in the solution space thereby leading to a loss of diversity. Maintaining good diversity in the population is extremely crucial for the success of a GA. This taking up of the entire population by one extremely fit solution is known as premature convergence and is an undesirable condition in a GA.

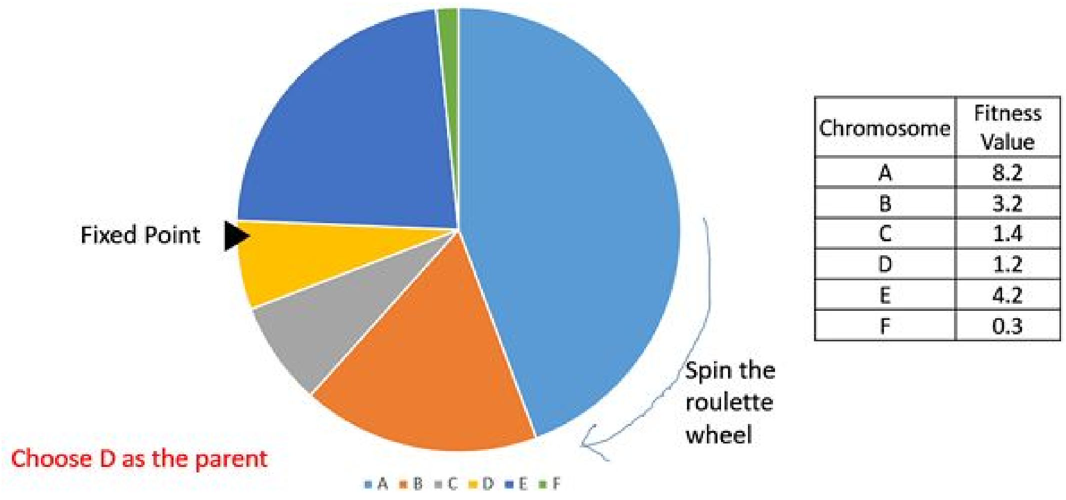
1. Fitness Proportionate Selection

Fitness Proportionate Selection is one of the most popular ways of parent selection. In this every individual can become a parent with a probability which is proportional to its fitness. Therefore, fitter individuals have a higher chance of mating and propagating their features to the next generation. Therefore, such a selection strategy applies a selection pressure to the more fit individuals in the population, evolving better individuals over time. Consider a circular wheel. The wheel is divided into **n** pies, where n is the number of individuals in the population. Each individual gets a portion of the circle which is proportional to its fitness value.

Two implementations of fitness proportionate selection are possible −

I. Roulette Wheel Selection

In a roulette wheel selection, the circular wheel is divided as described before. A fixed point is chosen on the wheel circumference as shown and the wheel is rotated. The region of the wheel which comes in front of the fixed point is chosen as the parent. For the second parent, the same process is repeated.



Chromosome	Fitness Value
A	8.2
B	3.2
C	1.4
D	1.2
E	4.2
F	0.3

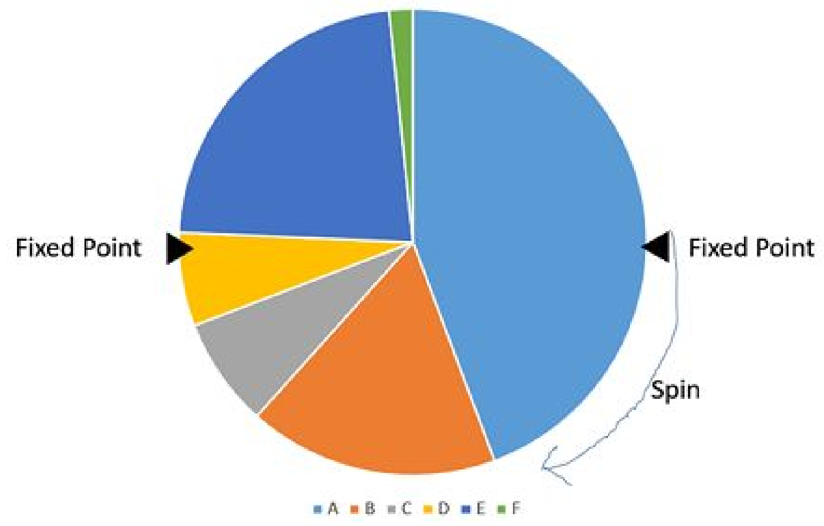
It is clear that a fitter individual has a greater pie on the wheel and therefore a greater chance of landing in front of the fixed point when the wheel is rotated.

Therefore, the probability of choosing an individual depends directly on its fitness Implementation wise, we use the following steps −

- Calculate S = the sum of a finesses.
- Generate a random number between 0 and S.
- Starting from the top of the population, keep adding the finesses to the partial sum P, till P<S.
- The individual for which P exceeds S is the chosen individual.

II. Stochastic Universal Sampling(SUS)

Stochastic Universal Sampling is quite similar to Roulette wheel selection, however instead of having just one fixed point, we have multiple fixed points as shown in the following image. Therefore, all the parents are chosen in just one spin of the wheel. Also, such a setup encourages the highly fit individuals to be chosen at least once.

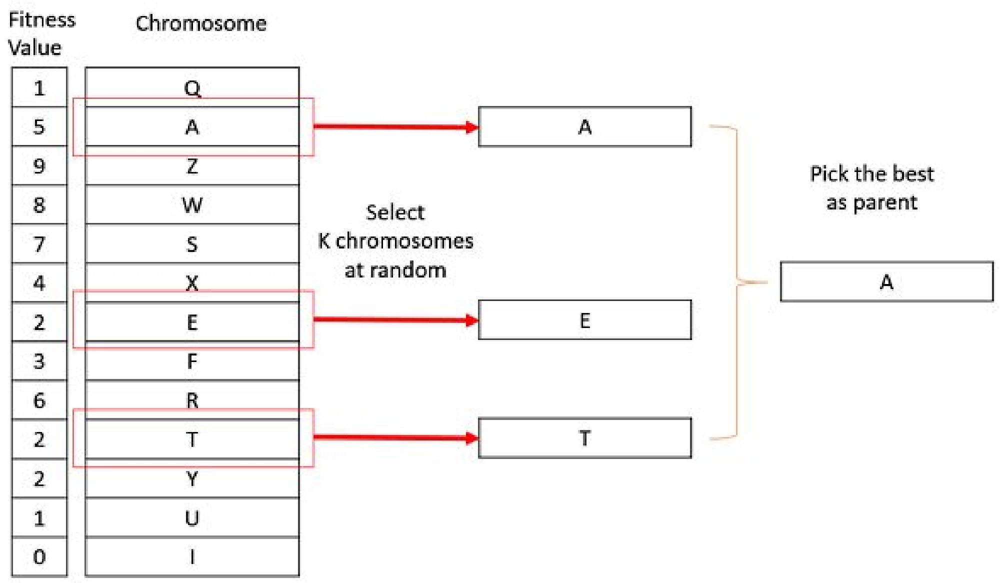


Chromosome	Fitness Value
A	8.2
B	3.2
C	1.4
D	1.2
E	4.2
F	0.3

It is to be noted that fitness proportionate selection methods don’t work for cases where the fitness can take a negative value.

2. <u>Tournament Selection</u>

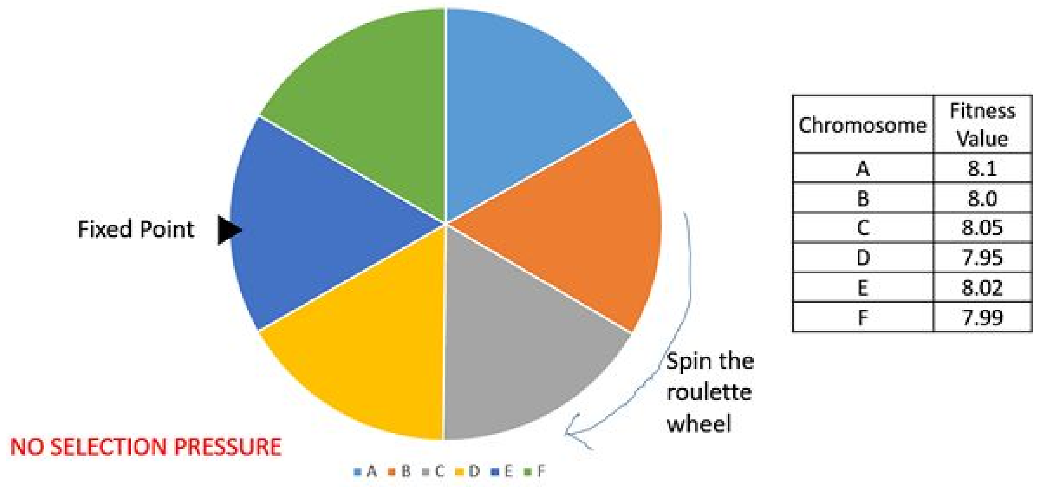
In K-Way tournament selection, we select K individuals from the population at random and select the best out of these to become a parent. The same process is repeated for selecting the next parent. Tournament Selection is also extremely popular in literature as it can even work with negative fitness values.



Fitness Value	Chromosome
1	Q
5	A
9	Z
8	W
7	S
4	X
2	E
3	F
6	R
2	T
2	Y
1	U
0	I

3. Rank Selection

Rank Selection also works with negative fitness values and is mostly used when the individuals in the population have very close fitness values (this happens usually at the end of the run). This leads to each individual having an almost equal share of the pie (like in case of fitness proportionate selection) as shown in the following image and hence each individual no matter how fit relative to each other has an approximately same probability of getting selected as a parent.



Chromosome	Fitness Value
A	8.1
B	8.0
C	8.05
D	7.95
E	8.02
F	7.99

In this, we remove the concept of a fitness value while selecting a parent. However, every individual in the population is ranked according to their fitness. The selection of the parents depends on the rank of each individual and not the fitness. The higher ranked individuals are preferred more than the lower ranked ones.

Chromosome	Fitness Value	Rank
A	8.1	1
B	8.0	4
C	8.05	2
D	7.95	6
E	8.02	3

F	7.99	5

4. Random Selection

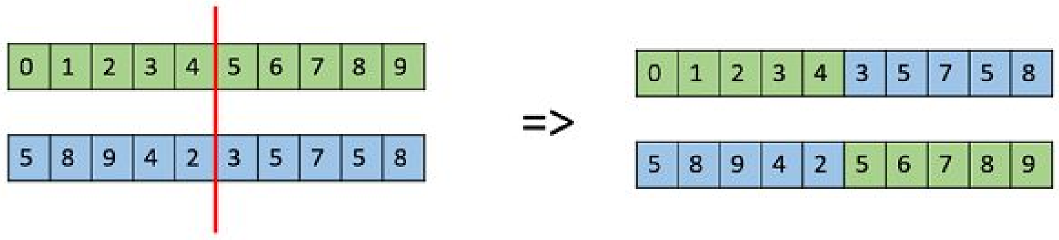
In this strategy we randomly select parents from the existing population. There is no selection pressure towards fitter individuals and therefore this strategy is usually avoided.

CrossOver Methods

The crossover operator is analogous to reproduction and biological crossover. In this more than one parent is selected and one or more off-springs are produced using the genetic material of the parents. Crossover is usually applied in a GA with a high probability. It is to be noted that crossover operators are very generic and the GA Designer might choose to implement a problem-specific crossover operator as well.

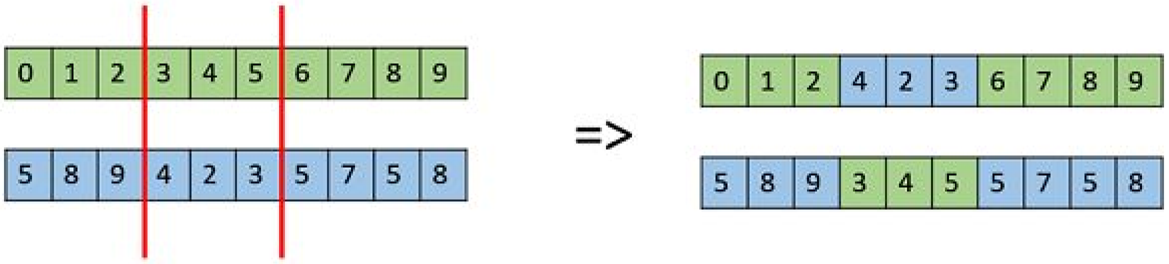
1. One Point Crossover

In this one-point crossover, a random crossover point is selected and the tails of its two parents are swapped to get new off-springs. Strings are characterized by Positional Bias.



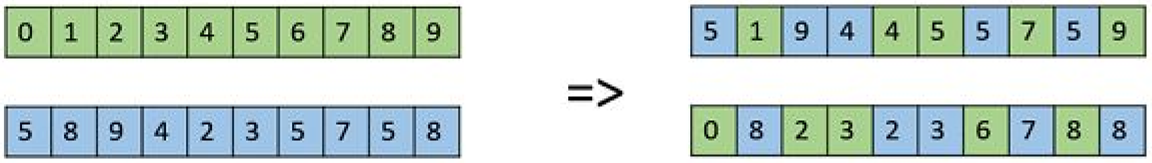
2. Multi Point Crossover

Multi point crossover is a generalization of the one-point crossover wherein alternating segments are swapped to get new off-springs. If we look at 2 point crossover, it is a specific case of a N-point Crossover technique. Two random points are chosen on the individual chromosomes (strings) and the genetic material is exchanged at these points.



3. Uniform Crossover

In a uniform crossover, we don't divide the chromosome into segments, rather we treat each gene separately. In this, we essentially flip a coin for each chromosome to decide whether or not it'll be included in the off-spring. We can also bias the coin to one parent, to have more genetic material in the child from that parent.



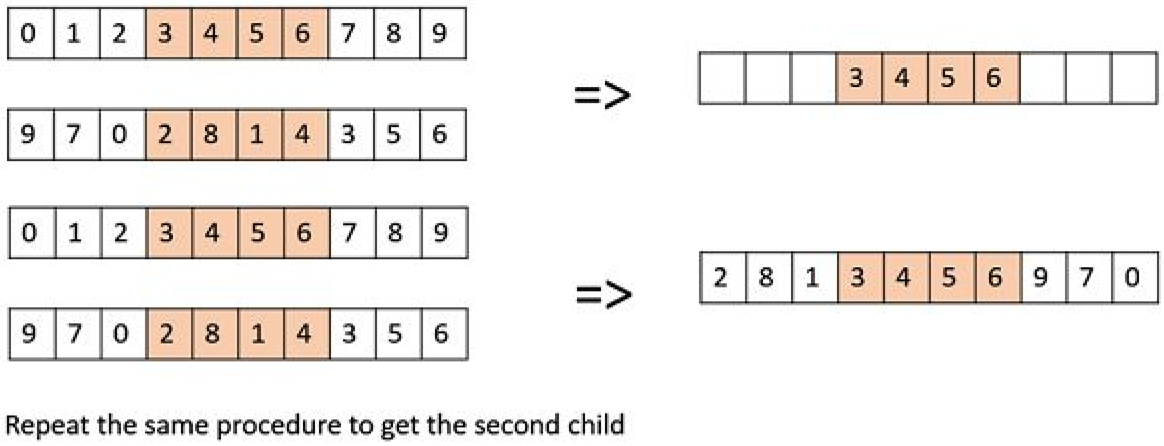
The crossover between two good solutions may not always yield a better or as good a solution. Since parents are good, the probability of the child being good is high. If offspring is not good (poor solution), it will be removed in the next iteration during "Selection".

4. Davis' Order Crossover(OX1)

OX1 is used for permutation based crossovers with the intention of transmitting information about relative ordering to the off-springs. It works as follows −

- Create two random crossover points in the parent and copy the segment between them from the first parent to the first offspring.
- Now, starting from the second crossover point in the second parent, copy the remaining unused numbers from the second parent to the first child, wrapping around the list.

- Repeat for the second child with the parent's role reversed.



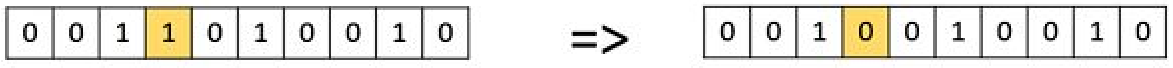
Mutation Methods

In simple terms, mutation may be defined as a small random tweak in the chromosome, to get a new solution. It is used to maintain and introduce diversity in the genetic population and is usually applied with a low probability. If the probability is very high, the GA gets reduced to a random search.

Mutation is the part of the GA which is related to the "exploration" of the search space. It has been observed that mutation is essential to the convergence of the GA while crossover is not.

1. Bit Flip Mutation

In this bit flip mutation, we select one or more random bits and flip them. This is used for binary encoded GAs.

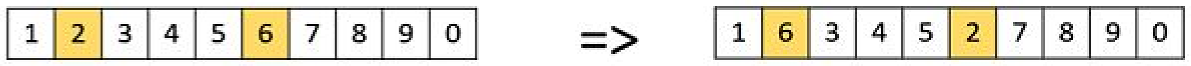


2. Random Resetting

Random Resetting is an extension of the bit flip for the integer representation. In this, a random value from the set of permissible values is assigned to a randomly chosen gene.

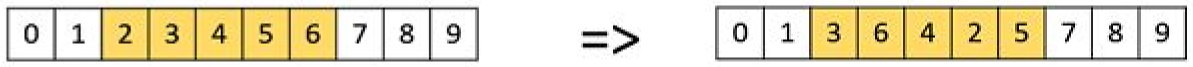
3. Swap Mutation

In swap mutation, we select two positions on the chromosome at random, and interchange the values. This is common in permutation based encodings.



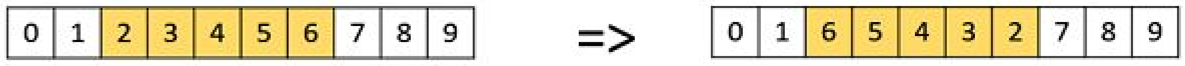
4. Scramble Mutation

Scramble mutation is also popular with permutation representations. In this, from the entire chromosome, a subset of genes is chosen and their values are scrambled or shuffled randomly.



5. Inversion Mutation

In inversion mutation, we select a subset of genes like in scramble mutation, but instead of shuffling the subset, we merely invert the entire string in the subset.



<u>Sequence Pattern Mining</u>

The task of **sequential pattern mining** is a data mining task specialized for analyzing **sequential data**, to discover **sequential patterns**. More precisely, it consists of discovering interesting subsequences in **a set of sequences**, where the interestingness of a subsequence can be measured in terms of various criteria such as its occurrence frequency, length, and profit. Sequential pattern mining has numerous real-life applications due to the fact that data is naturally encoded as **sequences of symbols** in many fields such as bioinformatics, e-learning, market basket analysis, texts, and webpage click-stream analysis.To do **sequential pattern mining**, a user must provide a sequence database and specify a parameter called the **minimum support threshold**. This parameter indicates a minimum number of sequences in which a pattern must appear to be considered frequent, and be shown to the user. For example, if a user sets the minimum support threshold to 2 sequences, the task of **sequential pattern mining** consists of finding all subsequences appearing in at least 2 sequences of the input database.

<u>GSP (Generalized Sequential Pattern)</u>

GSP is an apriori based(level-wise) algorithm. GSP algorithm makes multiple database passes. In the first pass, all single items (1-sequences) are counted. From the frequent items, a set of candidate 2-sequences are formed, and another pass is made to identify their frequency. The frequent 2-sequences are used to generate the candidate 3-sequences, and this process is repeated until no more frequent sequences are found. There are two main steps in the algorithm :

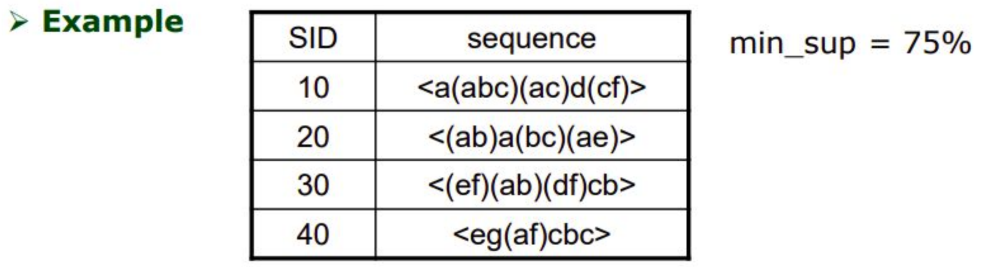
- Candidate Generation. Given the set of frequent (k-1)-frequent sequences Fk-1, the candidates for the next pass are generated by joining F(k-1) with itself. A pruning phase eliminates any sequence, at least one of whose subsequences is not frequent.
- Support Counting. Normally, a hash tree–based search is employed for efficient support counting. Finally non-maximal frequent sequences are removed.

Algorithm

```
F1 = the set of frequent 1-sequence
k=2,
do while F(k-1)!= Null;
    Generate candidate sets Ck (set of candidate k-sequences);
        For all input sequences s in the database D
            do
        Increment count of all a in Ck if s supports a
        Fk = {a ∊ Ck such that its frequency exceeds the threshold}
            k = k+1;
        Result = Set of all frequent sequences is the union of all Fks
            End do
End do
```

Trace

Sample Dataset

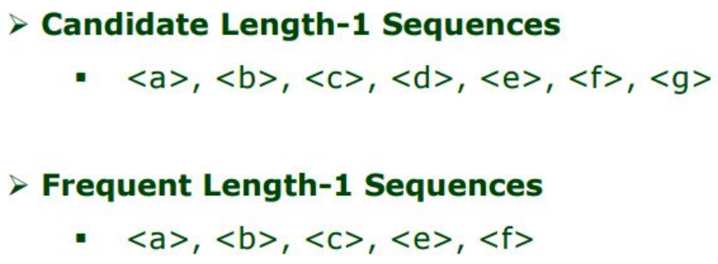


➢ **Example**

SID	sequence
10	<a(abc)(ac)d(cf)>
20	<(ab)a(bc)(ae)>
30	<(ef)(ab)(df)cb>
40	<eg(af)cbc>

min_sup = 75%

Length 1 Sequences



➢ **Candidate Length-1 Sequences**

- <a>, <b>, <c>, <d>, <e>, <f>, <g>

➢ **Frequent Length-1 Sequences**

- <a>, <b>, <c>, <e>, <f>

Length 2 Sequences

- **Candidate & Frequent Length-2 Sequences**

	<a>	<b>	<c>	<e>	<f>
<a>	<aa>	<ab> <ba>	<ac> <ca>	<ae> <ea>	<af> <fa>
<b>		<bb>	<bc> <cb>	<be> <eb>	<bf> <fb>
<c>			<cc>	<ce> <ec>	<cf> <fc>
<e>				<ee>	<ef> <fe>
<f>					<ff>

	<a>	<b>	<c>	<e>	<f>
<a>		<(ab)>	<(ac)>	<(ae)>	<(af)>
<b>			<(bc)>	<(be)>	<(bf)>
<c>				<(ce)>	<(cf)>
<e>					<(ef)>
<f>					

SID	sequence
10	<a(abc)(ac)d(cf)>
20	<(ab)a(bc)(ae)>
30	<(ef)(ab)(df)cb>
40	<eg(af)cbc>

Length 3 Sequences

- **Candidate & Frequent Length-3 Sequences**

	<ab>	<ac>	<bc>	<(ab)>
<ab>	<aab> <abb>	<abc> <acb>	~~<abc>~~	<(ab)b> <a(ab)>
<ac>		<aac> <acc>	~~<abc>~~ <bac>	<(ab)c>
<bc>			<bbc> <bcc>	~~<(ab)c>~~
<(ab)>				<(ab)(ab)>

SID	sequence
10	<a(abc)(ac)d(cf)>
20	<(ab)a(bc)(ae)>
30	<(ef)(ab)(df)cb>
40	<eg(af)cbc>

PrefixSpan(Prefix-projected sequential pattern mining)

PrefixSpan discovers all frequent sequential patterns occurring in a sequence database. But only prefix-based projection: less projections and quickly shrinking sequences.

Algorithm

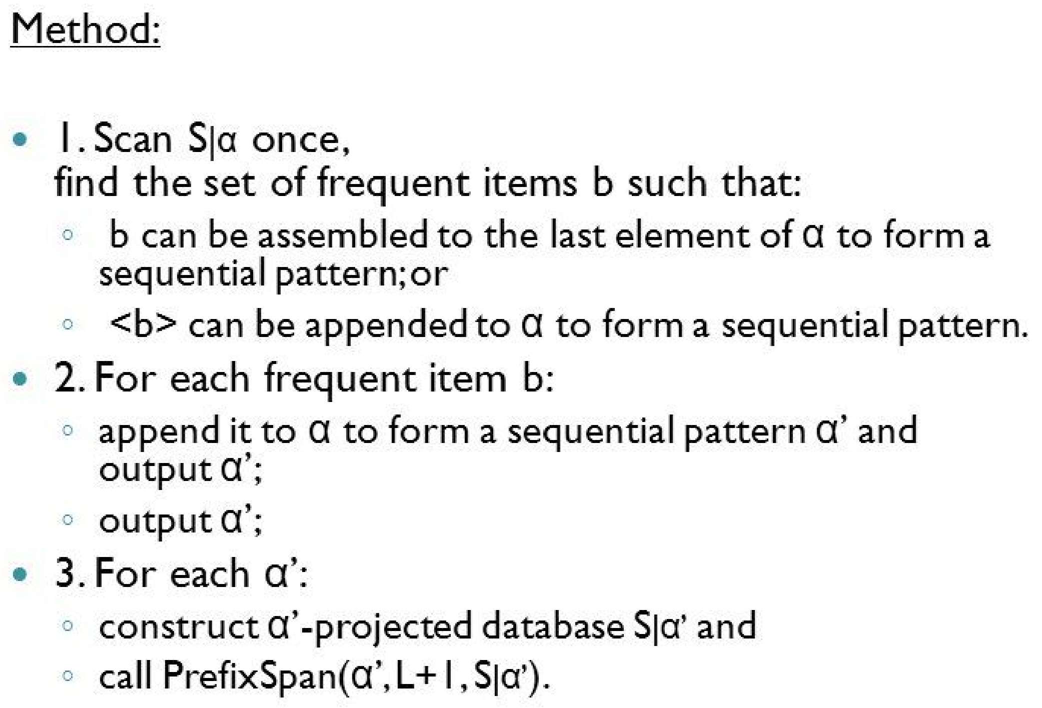
Input : A sequence of Database S, the minimum support threshold min_sup

Output : The complete set of sequential patterns

Method : PrefixSpan(<>,o,S)

Parameters *:* α : sequential pattern; l: the length of α;

S|α: the α-projected database, if α ≠<>; otherwise; the sequence database S



Method:

- 1. Scan S|α once,
  find the set of frequent items b such that:
  - b can be assembled to the last element of α to form a sequential pattern;or
  - <b> can be appended to α to form a sequential pattern.
- 2. For each frequent item b:
  - append it to α to form a sequential pattern α' and output α';
  - output α';
- 3. For each α':
  - construct α'-projected database S|α' and
  - call PrefixSpan(α',L+1, S|α').

Projected Database : A set of maximal-length suffixes with a given prefix

Trace

Sample Dataset

➢ **Example**

SID	sequence
10	<a(abc)(ac)d(cf)>
20	<(ab)a(bc)(ae)>
30	<(ef)(ab)(df)cb>
40	<eg(af)cbc>

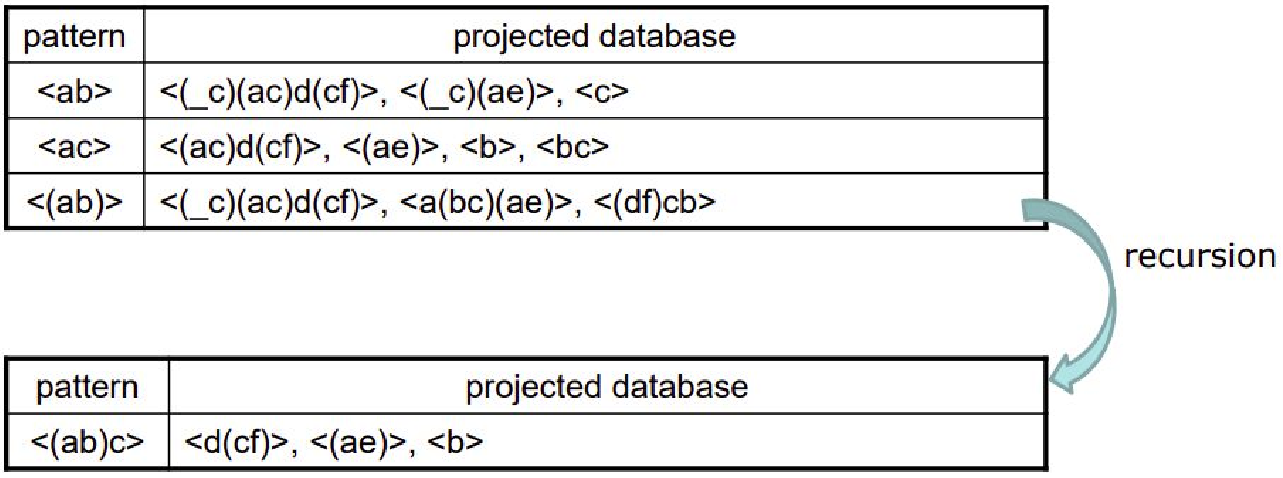
min_sup = 75%

pattern	projected database
<a>	<(abc)(ac)d(cf)>, <(_b)a(bc)(ae)>, <(_b)(df)cb>, <(_f)cbc>
<b>	<(_c)(ac)d(cf)>, <a(bc)(ae)>, <(df)cb>, <c>
<c>	<(ac)d(cf)>, <(ae)>, <b>, <bc>
<e>	<(_f)(ab)(df)cb>, <g(af)cbc>
<f>	<(ab)(df)cb>, <cbc>

pattern	projected database
<a>	<(abc)(ac)d(cf)>, <(_b)a(bc)(ae)>, <(_b)(df)cb>, <(_f)cbc>
<b>	<(_c)(ac)d(cf)>, <a(bc)(ae)>, <(df)cb>, <c>
<c>	<(ac)d(cf)>, <(ae)>, <b>, <bc>
<e>	<(_f)(ab)(df)cb>, <g(af)cbc>
<f>	<(ab)(df)cb>, <cbc>

recursion

pattern	projected database
<ab>	<(_c)(ac)d(cf)>, <(_c)(ae)>, <c>
<ac>	<(ac)d(cf)>, <(ae)>, <b>, <bc>
<(ab)>	<(_c)(ac)d(cf)>, <a(bc)(ae)>, <(df)cb>



pattern	projected database
<ab>	<(_c)(ac)d(cf)>, <(_c)(ae)>, <c>
<ac>	<(ac)d(cf)>, <(ae)>, <b>, <bc>
<(ab)>	<(_c)(ac)d(cf)>, <a(bc)(ae)>, <(df)cb>

pattern	projected database
<(ab)c>	<d(cf)>, <(ae)>, <b>

Strength of PrefixSpan

1. Efficient (major cost is to construct projected databases)
2. Project databases keep shrinking rapidly

Limitations of PrefixSpan

1. Searching frequency redundantly

Schema Theorem of Genetic Algorithm

The Schema Theorem says that short, low-order schemata with above-average fitness increase exponentially in frequency in successive generations. The theorem was proposed by John Holland in the 1970s. It was initially widely taken to be the foundation for explanations of the power of genetic algorithms. However, this interpretation of its implications has been criticized in several publications reviewed in , where the Schema Theorem is shown to be a special case of the Price equation with the schema indicator function as the macroscopic measurement. A schema is a template that identifies a subset of strings with similarities at certain string positions. Schema theorem serves as the analysis tool for the GA process.
Applicable to a canonical GA

- binary representation
- fixed length individuals
- fitness proportional selection
- single point crossover
- gene-wise mutation

Schema

A schema is a set of binary strings that match the template for schema H
A template is made up of 1s, 0s and *s where * is the don't care symbol that matches either 0 or 1.

Order o(H)

The order of a schema is the number of its fixed bits (length without number of *s in the template). E.g if H = 11**0 then o(H) = 3

Length δ(H)

The defining length is the distance between its first and the last fixed bits.
Example : if H = *1**1 then δ(H) = 5-2=3

Count

Suppose x is an individual that belongs to the schema H, then we say that x is an instance of H. m(H,k) denotes the number of instances of H in the k_{th} generation.

Fitness

f(x) denotes the fitness value of x. f(H,k) denotes average fitness of H in the k_{th} generation.

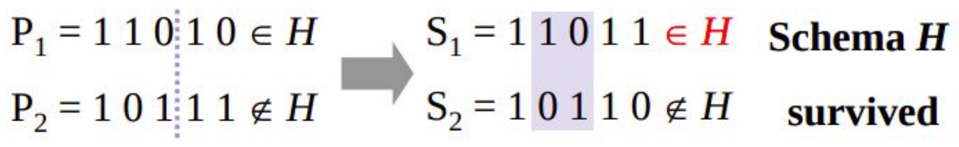
$$f(H,k) = \frac{\sum_{x \in H} f(x)}{m(H,k)}$$

Effect of GA on Schema

1. Effect of *Selection* on Schema
   Assumption : fitness proportional selection (like Roulette Selection)
   Selection probability for the individual x
   $$p_s(x) = \frac{f(x)}{\sum_{i=1}^{N} f(x_i)}$$
   where N is the total number of individuals.
   The expected number of instances of H in the mating pool M(H,k) is
   $$M(H,k) = \frac{\sum_{x \in H} f(x)}{\bar{f}} = m(H,k)\frac{f(H,k)}{\bar{f}}$$
   Schemas with fitness greater than the population average are likely to appear more in the next generation.
2. Effect of *Crossover* on Schema
   Assumption : Single-point crossover
   Schema H survives crossover operation if
   - One of the parents is an instance of the schema H and
   - One of the offspring is an instance of the schema H

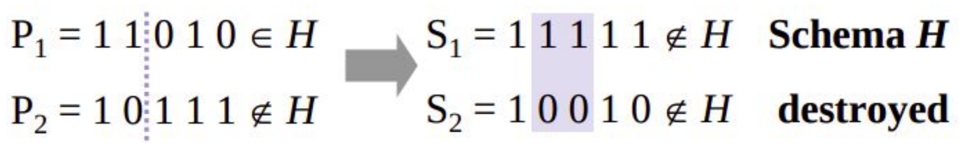
Crossover Survival examples

Consider H = *10**



$P_1 = 1\,1\,0 \,\vdots\, 1\,0 \in H \qquad S_1 = 1\,1\,0\,1\,1 \in H$

$P_2 = 1\,0\,1 \,\vdots\, 1\,1 \notin H \qquad S_2 = 1\,0\,1\,1\,0 \notin H$



$P_1 = 1\,1 \,\vdots\, 0\,1\,0 \in H \qquad S_1 = 1\,1\,1\,1\,1 \notin H$

$P_2 = 1\,0 \,\vdots\, 1\,1\,1 \notin H \qquad S_2 = 1\,0\,0\,1\,0 \notin H$

Suppose a parent is an instance of a schema H. When the crossover occurs within the bits of the defining length, it is destroyed unless the other parent repairs the destroyed portion.

Given a string with length l and a schema H with the defining length δ(H), the probability that the crossover occurs within the bits of the defining length is δ(H)/(l - 1). The upper bound of the probability that the schema H being destroyed is

D_c(H) ≤ p_c* δ(H)/(l - 1). Here p_c is the crossover probability.

The lower bound on the probability S_c(H) that H survives is

$$S_c(H) = 1 - D_c(H) \geq 1 - p_c \frac{\delta(H)}{l-1}$$

Schemas with low order are more likely to survive.

3. Effect of ***Mutation*** on Schema

   Assumption : mutation is applied gene by gene

   For a schema H to survive, all fixed bits must remain unchanged.

   Probability of a gene not being changed is (1-p_m) where p_m is the mutation probability of a gene.

The probability a schema H survives under mutation is given by

$$S_m(H) = (1 - p_m)^{o(H)}$$

Schemas with low order are more likely to survive.

Thus the Schema theorem can be expressed as

$$E[m(H,k+1)] \geq m(H,k)\frac{f(H,k)}{\bar{f}}\left(1-p_c\frac{\delta(H)}{l-1}\right)(1-p_m)^{o(H)}$$

The schema theorem states that the schema with above average fitness, short defining length and lower order is more likely to survive.

<u>Implementation of Schema theorem for optimization of ($x^3 - 2x^2 + x$)</u>

String processing table before mating

String No.	Initial Population	x	f(x)	Probability of Selection	Expected Count	Actual Count
1	01100	12	1452	0.0338	0.1552	0
2	11000	24	12696	0.2954	1.1816	1
3	10101	21	8400	0.1954	0.7816	1
4	11100	28	20433	0.4754	1.9016	2

Sum (f(x)) = 42981
Average(f(x)) = 10745.25
Consider 4 Schemas, H1 = 1****, H2 = *10**, H3 = 1***1 and H4 = *11** and applying it to the above table.

Schema processing table before reproduction

Schema Number	Schema	String Representatives	Schema Avg. Fitness	m
H1	1****	2,3,4	13843	51488
H2	*10**	2	12696	47262
H3	1***1	3	8400	31269
H4	*11**	1,4	10942.5	40734

String processing after mating

String No.	Mating pool after reproduction	Mate	Crossover site	New Population	x	f(x)
1	11100	3	3	11101	29	22736
2	11100	4	3	11000	24	12696
3	10101	1	3	10100	17	4351
4	11000	2	3	11100	28	20433

Sum (f(x)) = 60216

Average(f(x)) = 15054

Max(f(x)) = 22736

After Reproduction, for the string 11101 if Mutation is performed then 11101 changes to 11111 and the fitness value generated is 27900.

Thus the result generated by the Schema theorem on the function (x^3- $2x^2$+ x) is the same as the result obtained from applying Genetic Algorithm over the function.