ISSN:0975-9646

Genetic Algorithms and Programming-An Evolutionary Methodology

T. Venkat Narayana Rao[1] , Srikanth Madiraju[2]

Computer Science and Engineering,
Hyderabad Institute of Technology and Management, Hyderabad, A P, India.

Abstract: **Genetic Programming (GP) is an automated method for creating a working computer program from a high-level problem statement for a given problem. Genetic programming starts from a high-level statement of "what needs to be done" and to automatically create a computer program to solve the problem. In artificial intelligence, genetic programming (GP) is an evolutionary algorithm-based methodology inspired by biological evolution to find computer programs that perform a user defined task. It is a specialization of genetic algorithms (GA) where in each individual is a computer program. It is a machine learning technique used to optimize population of computer programs according to a fitness span determined by a program's ability to perform a given computational task. This paper presents an idea pertaining to the principles of genetic programming which includes relative effectiveness of mutation, crossover, breeding computer programs and fitness test in genetic programming. The literature of traditional genetic algorithms contains related studies, but through GP, it saves time by freeing the human from having to design complex algorithms. GP not only help in designing the algorithms but it could assist in creating the optimal solutions than traditional counterparts in a noteworthy ways.**

Keywords: **Genetic Programming, subtree, chromosomes, mutation, Evolutionary.**

I. INTROUCTION

Way back in1954, the first work on Genetic Programming has initiated, highlighting the basic functionality of all the four basic aspects viz... Breeding, Mutation, Crossover and Fitness test. Genetic Programming (hence forth referred as GP) began with the evolutionary algorithms firstly developed by Fogel Owens and Walsh was applied to evolutionary simulations as given in table 1. During 1960s and early 1970s, evolutionary algorithms became widely recognized as optimization methods. Genetic programming addresses the problem of automatic programming, namely, the problem of how to enable a computer to do useful things without instructing it, step by step, on how to do it. The first statement of modern "tree-based" Genetic Programming i.e., procedural languages organized in tree-based structures and operated on by suitably defined GA-operators was given by Nichael L. Cramer (1985). Koza has argued that mutation is in fact useless in Genetic Programming because of the position-independence of GP subtrees, and because of the large number of chromosome positions in typical Genetic Programming populations [Koza 1992, pp. 105–107]. This paper shows the merits of application of the basic aspects of Genetic Programming that have proved more efficient is generating a good algorithms and pool of programs for better delivery of results[12]. The whole discussion lies in the advantage of utility of these major points in modern computations and algorithm writing. In the 1990s, GP was mainly used to solve relatively simple problems because it is very computationally intensive. Recently GP has produced many novel and outstanding results in areas such as quantum computing, electronic design, game playing, sorting and searching due to improvements in GP technology and the power[6][8]. These results include the replication or development of several post-year-2000 inventions. GP has also been applied to evolvable hardware as well as computer programs. The history of computer programming is a history of attempts to move away from the "craftsman" approach - structured programming, object-oriented programming, object libraries and rapid prototyping. But each of these advances leaves the code that does the real work firmly in the hands of the programmer. The ability to enable computers to learn to program themselves is utmost importance in freeing the computer industry and the computer user from code that is obsolete before it is released. Since the 1950s, computer scientists have tried, with varying degrees of success, to give computers the ability to learn. The umbrella term for this field of study is "machine learning" a phrase crafted in 1959 by the first person Samuel, who made a computer perform a serious learning task. Originally, Samuel used "machine learning" to mean computers programming themselves [Samuel, 1963]. That goal has, for many years, proven too difficult. So the machine learning community has pursued more modest goals. A good contemporary definition of machine learning is due to Mitchell's study of computer algorithms that improve automatically through experience [Mitchell, 1996]. Genetic programming, aspires to do precisely that - to induce a population of computer programs that improve automatically as they experience the data on which they are trained. Accordingly, GP has become the part of very large body of research called machine learning. Developing a theory for GP has been very difficult and so in 1990s GP

was considered a sort of outcast among search techniques. But after a series of breakthroughs in the early 2000s, the theory of GP had a formidable and rapid development, so much so that it has been possible to build exact probabilistic models of GP (schema theories and Markov chain models).
Genetic Programming is an extension of the Genetic Algorithm which was invented by John Holland (1975). Although the idea of evolving programs was first suggested by Forsyth (1981) and Cramer (1985) among others, it was proved, promoted and developed into a practical tool by John Koza. Genetic Programming is one technique amongst a whole range of possible evolutionary algorithms [3].

What Machine Learning

Although genetic programming is a relative newcomer to the world of machine learning, some of the earliest machine learning research bore a distinct resemblance to today's GP. In 1958 and 1959, Friedberg attempted to solve fairly simple problems by teaching a computer to write computer programs [Friedberg, 1958] [Friedberg et al., 1959]. Friedberg's programs were 64 instructions long and were able to manipulate bitwise, a 64-bit data vector. Each instruction had a virtual "opcode" and two operands, which could reference either the data vector or the instructions. An instruction could jump to any other instruction or it could manipulate any bit of the data vector. Friedberg's system learned by using a modern mutation operator - random initialization of the individual solutions and random changes in the instructions. The *process* of machine learning that is, defining of the environment and the techniques for letting the machine learning system experience the environment for both training and evaluation are surprisingly similar from system to system. In the next section of this chapter, we shall, therefore, focus on machine learning as *a high-level process.* In early 1980s, machine learning was recognized as a distinct scientific discipline. Since then, the field has grown tremendously. Systems now exist that in narrow domains, learn from experience and make useful predictions about the world. Today, machine learning is termed as an important part of real-world applications such as industrial process control, robotics control, time series prediction, prediction of credit worthiness and pattern recognition problems such as optical character recognition and voice recognition. At the highest level, any machine learning system faces a similar task - how to learn from its experience of the environment [1][4].

Table 1: Summary of evolutionary algorithms

Year	Inventor	Technique	Individual
1958	Friedberg	learning machine	virtual assembler
1959	Samuel	mathematics	polynomial
1965	Fogel, Owens and Walsh	evolutionary programming	automaton
1965	Rechenberg, Schwefel	evolutionary strategies	real-numbered vector
1975	Holland	genetic algorithms	fixed-size bit string
1978	Holland and Reitmann	genetic classifier systems	rules
1980	Smith	early genetic programming	var-size bit string
1985	Cramer	early genetic programming	tree
1986	Hicklin	early genetic programming	LISP
1987	Fujiki and Dickinson	early genetic programming	LISP
1987	Dickmanns, Schmidhuber and Winklhofer	early genetic programming	assembler
1992	Koza	genetic programming	tree

The whole field is now called *Evolutionary Computation*. In common with many search techniques, the Genetic Programming algorithm has three basic components.

- A population of candidate solutions (usually called genes or chromosomes).
- A set of operations (genetic operators) which act on members of this population to produce new solutions [9].
- A method for evaluating how good each solution is, which involves trying it out in an appropriate environment.

In Genetic Programming, each candidate solution is stored in the form of a tree structure. Two examples of these trees are shown in here i.e. Figure 1. The first of these might be interpreted as a function i.e. $p = 2.107p + 0.345$ and the second as the logical expression **(agent-4 saidYes) OR (agent-3 DidBetterThan me)**. Initially, the population of candidate solutions is generated randomly from a specification of the possible nodes and terminals which can be used to construct a legal tree.

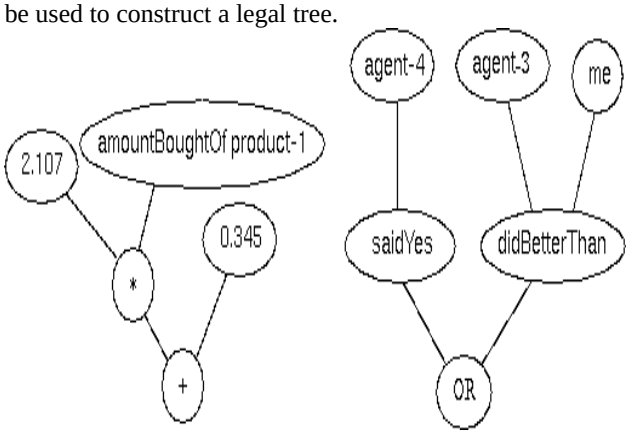


Figure 1. Example of Genetic Programming solution being stored in tree pattern.

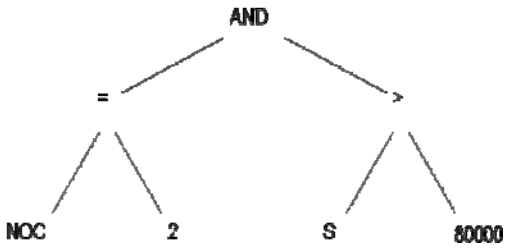
Different problems in artificial intelligence, symbolic processing, and machine learning can be viewed as wanted discovery of a computer program that results in some desired output for particular inputs being fed. In this new genetic programming, pool of computer programs are genetically bred using "the Darwinian principle of survival" of the fittest and using a genetic crossover (recombination) operator appropriate for genetically mating computer programs.

A. Breeding features:

- **Attributed features:**
  -competes with neural nets and alike
  -needs huge populations (thousands).
- **Special features :**
  -non-linear chromosomes: trees, graphs, Computer Programs as Trees.
  - Mutation possible but not necessary (disputed!) .

Start off with a large "pool" of random computer programs. Need a way of coming up with the best solution to the problem using the programs in the "pool". Based on the definition of the problem and criteria specified in the fitness test, mutations and crossovers are used to come up with new programs which will solve the problem further.

For example: IF (NOC = 2) AND (S > 80000) THEN good ELSE bad
can be represented by the following tree:
IF formula THEN good ELSE bad . Only unknown is the right formula, hence our search space (phenotypes) is the set of formulas i.e. Natural fitness of a formula**:** percentage of well classified cases of the model it stands for
Natural representation of formulas (genotypes) is: parse trees



The Trees are a universal form. We can represent an equation in the form of a tree for a given example below :

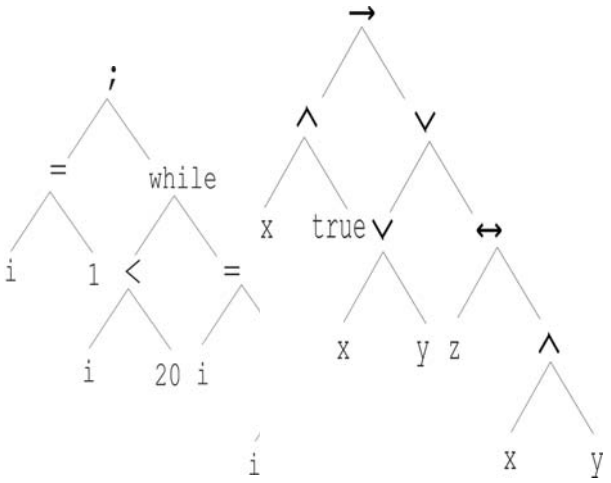
$$2 \cdot \pi + \left((x+3) - \frac{y}{5+1} \right)$$

$$(x \wedge \text{true}) \rightarrow ((x \vee y) \vee (z \leftrightarrow (x \wedge y)))$$

```
i =1;
while (i < 20)
{
        i = i +1
}
```

Tree based representation:

$(x \wedge \text{true}) \rightarrow ((x \vee y) \vee (z \leftrightarrow (x \wedge y)))$



In GA, ES, EP chromosomes are linear structures (bit strings, integer string, real-valued vectors, and permutations). Tree shaped chromosomes are non-linear structures. In GA, ES, EP the size of the chromosomes is fixed. Trees in Genetic Programming may vary in depth and width.

B. The Fitness Test Function

Identifying the way of evaluating how good a given computer program is at solving the problem at hand [11] [12]. How good can a program cope with its environment? Can be measured in many ways, i.e. error, distance, time, complexity etc.

- Fitness Test Criteria;
- Time complexity a good criteria i.e. n^2 vs. nlogn.
- Accuracy - Values of variables.
- Combinations of criteria may also be tested.

Fitness is the measure used by GP during simulated evolution of how well a program has learned to predict the output(s) from the input(s) i.e. the features of the learning domain. The goal of having a fitness evaluation is to give feedback to the learning algorithm regarding which individuals should have a higher probability of being allowed to multiply and reproduce, which individuals

should have a higher probability of being removed from the population. The fitness function is calculated on what we have earlier referred to as the training sets. Continuous Fitness Function, the fitness function should be designed to give graded and continuous feedback about how well a program performs on the training set. There are also other methods for calculating fitness.
In co-evolution methods for fitness evaluation [Angeline and Pollack, 1993][Hillis, 1992], individuals compete against each other without an explicit fitness value. In a game-playing application, the winner in a game may be given a higher probability of reproduction than the loser. In some cases, two different populations may be evolved simultaneously with conflicting goals. For example, one population might try to evolve programs that sort lists of numbers while the other population tries to evolve lists of numbers that are hard to sort. This method is inspired by arms races in nature where, for example, predators and prey evolve together with conflicting goals. In some cases, it might be advantageous to combine very different concepts in the fitness criteria. We could add terms for the length of the evolved programs or their execution speed, etc. Such fitness function is referred to as a *multiobjective fitness function.*

Each individual in a population is allotted with a fitness value as a result of its communication with the environment. Fitness is the driving force of Darwinian natural selection and, similarly to genetic algorithms. The environment is a set of cases which provides a basis for evaluating the fitness of the S expressions in the population. For example, for the exclusive-or function, the obvious choice for the environment is the set of four combinations of possible values for the two variable atoms D0 and D1 along with the associated value of the exclusive-or function for the four such combinations. For most of the problems described herein, the raw fitness of any LISP S-expression is the sum of the distances between the point in the range space returned by the S-expression for a given set of arguments and the correct point in the range space. The S-expression may Boolean-valued, integer-valued, real-valued, complex-valued, vector valued, multiple-valued, or symbolic-valued. If the S-expression is integer-valued or real-valued, the sum of distances is the sum of absolute values of the differences between the numbers involved. In particular, the raw fitness r(i,t) of an individual LISP S-expression i in the population of size M at any generational time step t is :

$$r(i,t) = \sum_{j=1}^{Ne} S(i,j)\ [\text{Ex-Or}]\ C(j)$$

Where S (i,j) is the value returned by S-expression i for environmental case j (of Ne environmental cases) and where C(j) is the correct value for environmental case j. If the S-expression is Boolean-valued or symbolic-valued, the sum of distances is equivalent to the number of mismatches. If the S-expression is complex-valued, or vector-valued, or multiple valued, the sum of the distances is the sum of the distances separately obtained from each component of the vector or list. The closer this sum of distances is to zero, the better is the S-expression.
One can use the sum of the distances or the square root of the sum of the squares of the distances in this computation. It is important that the fitness function return a range of various values that distinguish the performance of single entities in the pool. As an example, a fitness function test that returns only two values (say, a true for a solution and a false otherwise) provides not enough information for helping guide to an adaptive process. Any outcome that is discovered with such a fitness function test is, then, essentially can be an accident (a false return). A wrong choice of the function set in relation to the number of environment cases for a given case can raise the same situatution. For example, if the Boolean function OR is in the function set for the exclusive-or problem, this function alone satisfies three of the four environment cases. Since the initial random population of individuals will almost certainly be numerous S-expressions equivalent to the OR function, we are effectively left with only two distinguishing levels of the fitness (i.e. 4 for a solution and 3 otherwise).

The process of solving some typical problems can be reframed as a search for a most fit individual computer program in the range of possible computer programs. In particular, the search Space is the hyperspace of LISP symbolic expressions (called S-expressions) encapsulating functions and terminals appropriate to the problem domain. As noticed, the LISP S-expression which solves each of the problems described above will surface from a simulated evolutionary process using a new genetic programming paradigm using a "hierarchical genetic algorithm".
The functions may be standard arithmetic operations, standard programming operations, standard mathematical functions and various domain-specific functions[10]. A fitness function evaluates how well each individual LISP S-expression in the population performs in the particular problem environment. In many problems, the fitness is measured by the sum of the distances i.e. taken for all the environmental cases between the point in the range space (whether Boolean-valued, integer-valued, real-valued, complex-valued, vector-valued, symbolic valued, or multiple-valued) created by the S-expression for a given set of arguments and the correct point in the range space. An algorithm based on the Darwinian model of reproduction and survival of the fittest and genetic recombination is used to create a new population of individuals from the current population of individuals.
The two participating parental S-expressions are selected in proportion to fitness. The resulting offspring S-expressions are composed of sub expressions "building blocks" from their parents. Then, the new population of offspring i.e. the new generation replaces the old population of parents, the old generation. Then, each individual in the new population is measured with the fitness function and the process is repeated. At every level of this highly parallel, locally

governed, and defragmented process, the state of the process will include only of the current population of individuals. Moreover, the only input to the algorithmic process will be the observed fitness of the individuals in the current population in correlation with the problem environment. This algorithm will produce populations which, over a period of generations, intend to show increasing average fitness in dealing with their environment, and which, in addition, will tend to robustly i.e. rapidly and effectively adapt and work accordingly to the changes in their environment. The solution produced by this algorithm at any given time can be viewed as the entire population of distinctive alternative solutions (typically with improved overall average fitness as compared to the beginning of the algorithm), or, more commonly, as the single best individual in the population at that time. The hierarchical character of the computer programs that are produced by the genetic programming paradigm is an important characteristic of the genetic programming. The results of this genetic programming methodology process are inherently hierarchical.

The dynamic variability of the computer programs that are developed along the way to a solution is also an important feature of the genetic programming paradigm. In each case, it would be difficult and unnatural to try to specify or limit the size and shape of the eventual solution in advance. Moreover, the advance specification or restriction of the size and shape of the solution to a problem narrows the window by which the system views the world and might well prohibit finding the solution to the problem.

Another important feature of the genetic programming paradigm is absence of preprocessing of inputs and the fact that the solution is expressed directly in terms of the functions and arguments from the problem domain. This makes the results immediately comprehensible and intelligible in the terms of the problem domain. Most importantly, the "genetic programming" paradigm is general and provides a single, unified approach to a variety of seemingly different problems in a variety of areas.

II. THE CROSSOVER (RECOMBINATION) OPERATION

Crossing over, process in genetics by which the two chromosomes of a homologous pair swap equal segments with each other. Crossing over occurs in the first division of meiosis. At that stage each chromosome has replicated into two strands called sister chromatids[5]. The two homologous chromosomes of a pair synapse, or come together. While the chromosomes are synapsed, breaks occur at corresponding points in two of the non-sister chromatids, i.e., in one chromatid of each chromosome[1]. Since the chromosomes are homologous, breaks at corresponding points mean that the segments that are broken off contain corresponding genes, i.e., alleles. The broken sections are then exchanged between the chromosomes to form complete new units, and each new recombined chromosome of the pair can go to a different daughter sex cell as shown in Figure 2. Crossing over results in recombination of genes found on the same chromosome, called linked genes that would otherwise always be transmitted together. Because the frequency of crossing over between any two linked genes is proportional to the chromosomal distance between them, crossing over frequencies are used to build genetic, or linkage, maps of genes on chromosomes.

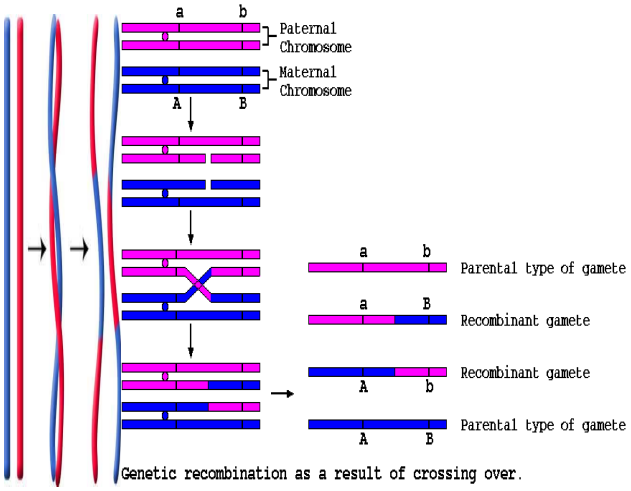


Figure2.Chromosomes Pairs and Genetic Recombinations

There are three principal constraints on biological crossover:

- Biological crossover takes place only between members of the *similar* species. In fact, living creatures put much energy into identifying other members of their species - often putting their own existence at risk to do so. Bird songs, for example, attract mates of the same species and predators.
- Biological crossover occurs with remarkable attention to preservation of "semantics". Thus, crossover usually results in the same gene from the father being matched with the same gene from the mother. In other words, the hair color gene does not get swapped for the tallness gene[2].
- Biological crossover is *homologous*. The two DNA strands are able to line up identical or very similar base pair sequences so that their crossover is perfect almost down to the molecular level. But this does not eliminate crossover at duplicate gene sites or other variations, as long as very similar sequences are available.

In nature, most crossover events are successful i.e. they result in viable offspring. This is a sharp contrast to GP crossover, where over 75% of the crossover events are what would be termed in biology "lethal". What causes this difference? In a sense, GP takes on an enormous task. It must evolve genes (building blocks) so that crossover makes sense and it must evolve a solution to the problem all in a few hundred generations. It took nature billions of years to come up with the preconditions so that crossover itself could evolve. GP crossover is very different from biological crossover. Crossover in standard GP is unconstrained and

uncontrolled. Crossover points are selected randomly in both parents. There are no predefined building blocks (genes). Crossover is expected to find the good building blocks and not to disorder them even while the building blocks grow.

- In the basic GP system, any subtree may be crossed over with any other subtree. There is no requirement that the two subtrees fulfill similar functions. In biology, because of homology, the different alleles of the swapped genes make only minor changes in the same basic function.
- There is no requirement that a subtree, after being swapped, is in a context in the new individual that has any relation to the context in the old individual. In biology, the genes swapped are swapped with the corresponding gene in the other parent.
- Were GP to develop a good subtree building block, it would be very likely to be disrupted by crossover rather than preserved and spread. In biology, crossover happens mostly between similar genetic materials. It takes place so as to conserve gene function with only minor changes.

There is no reason to suppose that randomly initialized individuals in a GP population are members of the same species-they are created randomly.

Crossovers in Programs:

The crossover (recombination) operation for the genetic programming paradigm creates variation in the population by producing offspring's that combine traits from two parents. The crossover operation starts with two parental S-expressions and produces at least one offspring S-expression. In general, at least one parent is chosen from the population with a probability equal to their respective normalized fitness values. In this paper, both parents are so chosen. The operation begins by randomly and independently selecting one point in each parent using a Probability distribution. Note that the number of points in the two parents typically is not equal.

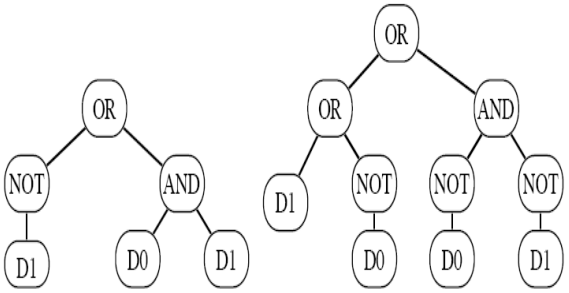
- Two parental programs are selected from the population based on fitness.
- A crossover point is randomly chosen in the first and second parent.
- The sub tree rooted at the crossover point of the first, or receiving, parent is deleted and replaced by the sub tree from the second, or contributing, parent.
- Crossover is the predominant operation in genetic programming (i.e. genetic algorithm) work and is performed with a high probability that is about 85% to 90%.

The "crossover fragment" for a particular parent is the rooted sub-tree whose root is the crossover point for that parent and where the sub-tree consists of the entire sub-tree lying below the crossover point (i.e. more distant from the root of the original tree). Viewed in terms of lists in LISP programming language , the crossover fragment is the sub-list starting at the crossover point [4].

The first offspring is produced by deleting the crossover fragment of the first parent from the first parent and then impregnating the crossover fragment of the second parent at the crossover point of the first parent. In producing this first offspring the first parent acts as the base parent (the female parent) and the second parent acts as the impregnating parent (the male parent). The second offspring is produced in a symmetric manner as shown in example1.

EXAMPLE1:

For example, consider the two parental LISP S-expressions below:

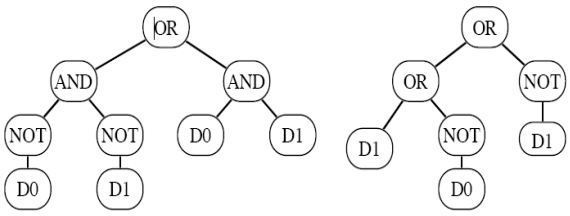


In terms of LISP S-expressions, the two parents are (OR **(NOT D1)** (AND D0 D1)) and

(OR (OR D1 (NOT D0)) **(AND (NOT D0) (NOT D1)**)

Assume that the points of both trees above are numbered in a depth-first way starting at the left. Suppose that the second point (out of the 6 points of the first parent) is selected as the crossover point for the first parent and that the sixth point (out of the 10 points of the second parent) is selected as the crossover point of the second parent. The crossover points are therefore the NOT function in the first parent and the AND function in the second parent. Thus, the bold, underlined portions of each parent above are the crossover fragments. The two crossover fragments are shown below:

The two offspring resulting from crossover are shown below.



Note that the first offspring above is a perfect solution for the exclusive-or function, namely
(OR (AND (NOT D0) (NOT D1)) (AND D0 D1)).

Note that because entire sub-trees are swapped, this genetic crossover (recombination) operation produces valid LISP S-expressions as offspring regardless of which point is selected in either parent. If the root of one tree happens to be selected as the crossover point, the crossover operation will insert that entire parent into the second tree at the crossover point of second parent. In addition, the sub-tree from the second parent will, in this case, then become the second offspring. If the roots of two parents happen to be chosen as crossover points, the crossover operation simply degenerates to an instance of fitness proportionate reproduction on those two parents.

If a terminal is located at the crossover point in precisely one parent, then the sub-tree from the second parent is inserted at the location of the terminal in the first parent and the terminal from the first parent is inserted at the location of the sub-tree in the second parent. In this case, the crossover operation often has the effect of increasing the depth of one tree and decreasing the depth of the second tree. If terminals are located at both crossover points selected, the crossover operation merely swaps these terminals from tree to tree [7] [12].

III. MUTATION IN NATURE

Mutations can involve large sections of DNA becoming duplicated, usually through genetic recombination. These duplications are a major source of raw material for evolving new genes, with tens to hundreds of genes duplicated in animal genomes for million years and have following characteristics.

- Ultimate source of genetic variation.
- Radiation, chemicals change genetic information.
- Causes new genes.
- One chromosome.
- Asexual.
- Very rare combinations possible.

A gene mutation is a permanent change in the DNA sequence that makes up a gene. Mutations range in size from a single DNA building block (DNA base) to a large segment of a chromosome. Gene mutations occur in two ways: they can be inherited from a parent or acquired during a person's lifetime. Mutations that are passed from parent to child are called hereditary mutations or germ line mutations (because they are present in the egg and sperm cells, which are also called germ cells). This type of mutation is present throughout a person's life in virtually every cell in the body. Figure 3. showcases a example involving crossover and mutation process for a inducting assembler .

Somatic also called as acquired mutations occur in the DNA of individual cells at some time during a person's life. These changes can be caused by environmental factors such as ultraviolet radiation from the sun, or can occur if a mistake is made as DNA copies itself during cell division. Acquired mutations in somatic cells (cells other than sperm and egg cells) cannot be passed on to the next generation.

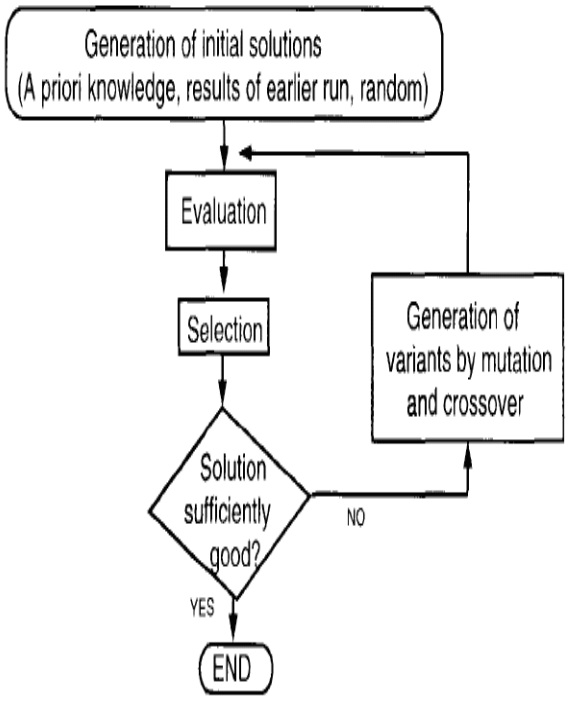


Figure 3. example showing a mechanism in crossover and mutuation to induce an assembler.

Some genetic changes are very rare, others are common in the population. Genetic changes that occur in more than 1 % of the population are called polymorphisms [6] [1] [9]. They are common enough to be considered a normal variation in the DNA. Polymorphisms are responsible for many of the normal differences between people such as eye color, hair color, and blood type. Although many polymorphisms have no negative effects on a person's health, some of these variations may influence the risk of developing certain disorders. Mutations can involve large sections of DNA becoming duplicated, usually through genetic recombination.

Entropy driven variation, such as mutation, is the principal source of variability in evolution. There are many types of mutation, including [Watson and Wonklhofer et al., 1987] as given below:

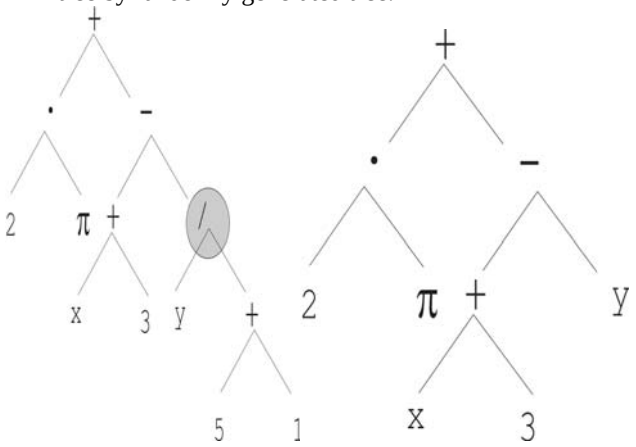
- Changes from one base pair to another are a possibility. These often produce neutral or useful variations. Although a base pair switch occurs about once every ten million replications or less, there are hot spots where base pair switching is up to twenty-five times the usual rate.
- Additions or deletions of one or more base pairs. This is called a frame shift mutation and often has drastic consequences on the functioning of the gene.
- Large DNA sequence rearrangements. These may occur for any number of reasons and are almost always lethal to the organism.

Mutation operates on only one individual. Normally, after crossover has occurred, each child produced by the crossover undergoes mutation with a low probability. The probability of mutation is a parameter of the run. A separate application of crossover and mutation, however, is also possible and provides another reasonable procedure. When an individual has been selected for mutation, one type of mutation operator in tree GP selects a point in the tree arbitrarily and replaces the existing subtree at that point with a new randomly generated subtree. The new randomly generated subtree is created in the same way, and subject to the same limitations (on depth or size) as programs in the initial random population. The altered individual is then located back into the population. In linear GP, mutation is a bit different. When an individual is chosen for mutation, the mutation operator first selects one instruction from that individual for mutation. It then makes one or more changes in that instruction. The type of change is chosen randomly from the following list:

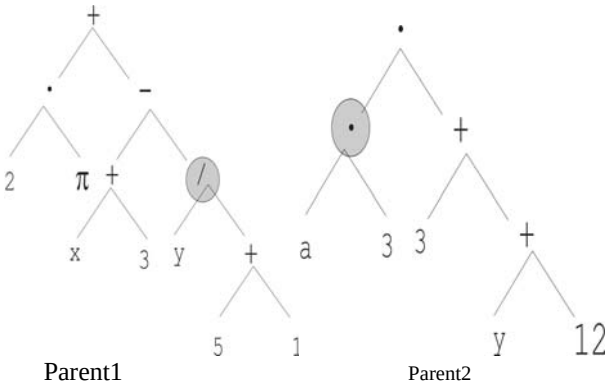
- Any of the register designations may be changed to another randomly chosen register designation that is in the register set.
- The operator in the instruction may be changed to another operator that is in the function set.
- A constant may be changed to another randomly chosen constant in the designated constant range.

Mutations in Programs:

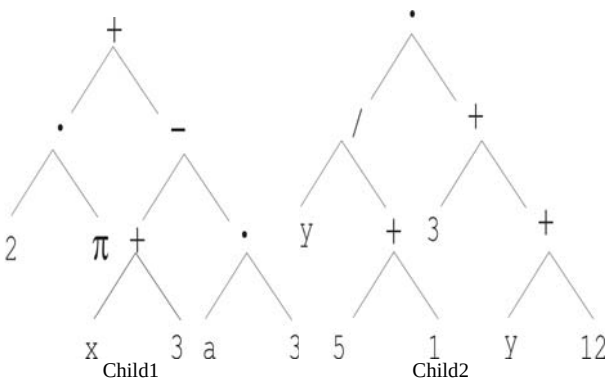
- Single parental program is probabilistically selected from the population based on fitness.
- Mutation point randomly chosen. the sub tree rooted at that point is deleted, and a new subtree is grown there using the same random growth process that was used to generate the initial population.
- Asexual operations are typically performed sparingly (with a *low probability* of, probabilistically selected from the population based on fitness).
- Most common mutation: replace randomly chosen sub tree by randomly generated tree.



- Mutation has two parameters: Probability p_m to choose mutation vs. recombination and the Probability to choose an internal point as the root of the sub tree to be replaced. Remarkably p_m is advised to be 0 (Koza'92) or very small, like 0.05 (Banzhaf et al. '98). The size of the child can exceed the size of the parent. The below subtrees shows the parent and child patterns in detail.



Parent1 Parent2



Child1 Child2

The mutation operation potentially can be beneficial in reintroducing diversity in a population that may be tending to prematurely converge.

IV. APPLICATIONS OF GENETIC PROGRAMMING

The detailed summary of applications of Genetic Programming domain is listed in the table 2. and Tables 3.

A. Designing Electronic Circuits

John Koza, a professor at Stanford and CEO of Genetic Programming Inc. is perhaps the person most responsible for making GP more acceptable in the eyes of the AI community. He and his team have successfully applied genetic programming techniques to a variety of applications ranging from bioinformatics to distributed systems. One of their most successful endeavors has been to the generation of electronic circuit designs. Here, the programs are actually all about the flow of information around the circuits, so the function set contains functions which mimic the actions of transistors, resistors, etc., on the flow of electricity.

According to the web site at Genetic Programming Inc: "there are now 36 instances where genetic programming has automatically produced a result that is competitive with human performance, including 15 instances where genetic programming has created an entity that either infringes or duplicates the functionality of a previously patented 20th-century invention, 6 instances where genetic programming has done the same with respect to a 21st-century invention, and 2 instances where genetic programming has created a patentable new invention".

B. Evolutionary Art

One of the most exciting and creative areas in which genetic programming is being is applied is evolutionary art. In contrast to most GP applications, in evolutionary art, the user often acts directly as the fitness function. That is, the GP engine generates a set of programs which can produce images (i.e. JPEG's etc.), either by transforming a given image, or generating pixel data from scratch. These images are then shown to the user, who performs the selection by choosing those which they most like. The GP engine then generates a population from the chosen images and selects from it images which fairly closely resemble the ones chosen by the user, or which have some properties similar to the chosen ones, e.g., color distribution. The user then selects those with most appeal again, and the process continues until the user is so happy with the image that they put it on their homepage. The evolutionary art community includes many artists and computing professionals, and the artworks their programs produce generate much interest (similar to how everyone was amazed by fractal images when they first came out). Such an approach was recently used to generate images for an ad-campaign by Absolute Vodka, for example.

Table 2. : Applications of Genetic Programming

Application domain	Year	Application	Source
algorithms	1996	acyclic graph evaluation	[Ehrenburg, 1996]
	1997	caching algorithms	[Paterson and Livesey, 1997]
	1996	chaos exploration	[Oakley, 1996]
	1994	crossing over between subpopulations	[Ryan, 1994]
	1994	randomizers (R)	[Jannink, 1994]
	1996	recursion	[Wong and Leung, 1996]
	1993	sorting algorithms	[Kinnear, Jr., 1993a, Kinnear, Jr., 1994]
	1992	sorting networks	[Hillis, 1992]
art	1994	artworks	[Spector and Alpern, 1994]
	1993	images	[Sims, 1993a]
	1995	jazz melodies	[Spector and Alpern, 1995]
	1995	musical structure	[Spector and Alpern, 1995]
	1994	virtual reality	[Das et al., 1994]
biotechnology	1996	biochemistry	[Raymer et al., 1996]
	1995	control of biotechnological processes	[Bettenhausen et al., 1995a]
	1995	DNA sequence classification	[Handley, 1995]
	1993	detector discovering and use	[Koza, 1993b]
	1994	protein core detection (R2)	[Handley, 1994a]
	1994	protein segment classification	[Koza and Andre, 1996a]
	1994	protein sequence recognition	[Koza, 1994b]
	1993	sequencing	[Handley, 1993a]
	1996	solvent exposure prediction (R2)	[Handley, 1996b]
computer graphics	1994	3D object evolution	[Nguyen and Huang, 1994]
	1994	3D modeling (R)	[Nguyen et al., 1993]
	1991	artificial evolution	[Sims, 1991a]
	1991	artificial evolution	[Sims, 1991b]
	1993	computer animation	[Ngo and Marks, 1993]
	1995	computer animation	[Gritz and Hahn, 1995]
	1997	computer animation	[Gritz and Hahn, 1997]
computing	1995	computer security	[Crosbie and Spafford, 1995]
	1994	damage-immune programs	[Dickinson, 1994]
	1996	data compression	[Nordin and Banzhaf, 1996]
	1992	data encoding	[Koza, 1992d]
	1996	data processing structure identification	[Gray et al., 1996a]
	1991	decision trees	[Koza, 1991]
	1996	decision trees	[Masand and Piatesky-Shapiro, 1996]
	1996	inferential estimation	[McKay et al., 1996]
	1994	machine language	[Nordin, 1994]
	1994	monitoring	[Atkin and Cohen, 1993]
	1995	machine language	[Crepeau, 1995]
	1996	object orientation	[Bruce, 1996]
	1996	parallelization	[Walsh and Ryan, 1996]
	1997	parallelization	[Ryan and Walsh, 1997]
	1996	specification refinement	[Haynes et al., 1996]
	1995	software fault number prediction	[Robinson and McIlroy, 1995a]
	1994	virtual reality	[Das et al., 1994]

Table 3.:Science Oriented Applications of Genetic Programming

Application domain	Year	Application	Source
modelling	1995	biotechnological fed-batch fermentation	[Bettenhausen et al., 1995b]
	1995	macro-mechanical model	[Schoenauer et al., 1995]
	1997	metallurgic process model	[Greeff and Aldrich, 1997]
	1995	model identification	[Schoenauer et al., 1996]
	1995	model induction	[Babovic, 1995]
	1994	spatial interaction models	[Openshaw and Turton, 1994]
	1995	system identification	[Iba et al., 1995b]
natural languages	1994	confidence of text classification (R)	[Masand, 1994]
	1994	language decision trees	[Siegel, 1994]
	1996	language processing	[Dunning and Davis, 1996]
	1997	sense clustering	[Park and Song, 1997]
optimization	1994	database query optimization	[Kraft et al., 1994]
	1996	database query optimization	[Stillger and Spiliopoulou, 1996]
	1994	job shop problem	[Atlan et al., 1994]
	1996	maintenance scheduling	[Langdon, 1996a]
	1996	network (LAN)	[Choi, 1996]
	1995	railroad track maintenance	[Grimes, 1995]
	1994	training subset selection	[Gathercole and Ross, 1994]
pattern recognition	1994	classification	[Tackett and Carmi, 1994]
	1996	classification	[Abramson and Hunter, 1996]
	1997	classification	[Gray and Maxwell, 1997]
	1995	dynamics extraction	[Dzeroski et al., 1995]
	1994	feature extraction	[Andre, 1994a]
	1994	filtering	[Oakley, 1994b]
	1994	combustion engine misfire detection	[Hampo et al., 1994]
	1996	magnetic resonance data classification	[Gray et al., 1996b]
	1996	myoelectric signal recognition	[Fernandez et al., 1996]
	1993	noise filtering	[Oakley, 1993]
	1994	optical character recognition	[Andre, 1994b]
	1996	object classification	[Ryu and Eick, 1996]
signal processing	1992	control vehicle systems	[Hampo and Marko, 1992]
	1996	digital	[Esparcia-Alcazar and Sharman, 1996]
	1993	signal filtering	[Oakley, 1993]
	1993	signal modeling	[Sharman and Esparcia-Alcazar, 1993]
	1996	waveform recognition	[Fernandez et al., 1996]

CONCLUSION

The data presented in this paper is formulated on various studies and research work carried out, presenting various examples that explain in detail about the aspects of genetic programming. The final analysis of the data indicates that crossover is more successful than mutation overall, though mutation is often better for small populations, depending on the domain. However, the difference between the two is usually small, and often statistically insignificant. Apart from its straightforward instrumental uses, the study of GP opens up a new and wide range of possibilities for social simulators that of models based on a learning technique where the structure of what is learnt. It is a methodology that can be used to generate some aspects of the creative learning caliber of humans. Of course, the GP algorithm is not a perfect mirror of human cognition. To be used effectively as a descriptive element in a social simulation, it needs to be adapted to ensure that it is as realistic as possible. GP is not yet a completely mature technique. As such, its impact in the field of social simulation has just begun. No doubt its impact will be at least as great as those of previous paradigms such as neural networks or genetic algorithms. It introduces a new computational analogy but because it is unparalleled as a creative computational technique thus we anticipate that in the days to come GP would be applied and we may be genuinely surprised at the results.

REFERENCES

[1] Andre, D. and Teller, A. 1996. A Study in Program Response and the Negative Effects of Introns. In Genetic Programming. In Proceedings of the First Annual Conference on Genetic Programming (GP96), edited by John Koza et al.The MIT Press. pp. 12–20.

[2] Angeline, P.J. 1996. Two Self-Adaptive Crossover Operators for Genetic Programming. In Advances in Genetic Programming 2, edited by P.J. Angeline and K.E. Kinnear, Jr. The MIT Press. pp. 89– 109.

[3] Koza, J.R. 1992. Genetic Programming: On the Programming of Computers by Means of Natural Selection.. The MIT Press.Koza, J .R. 1994.

[4] Genetic Programming II: Automatic Discovery of Reusable Programs. The MIT Press.Mitchell, M. 1996. An Introduction to Genetic Algorithms.TheMIT Press.

[5] Shaffer, J.D., and L.J. Eshelman. 1991. On Crossover as an Evolutionarily Viable Strategy. In Proceedings of the Fourth International Conference on Genetic Algorithms, edited by R.K. Belew and L.B. Booker. Morgan Kaufmann.

[6] Genetic programming from tiera.ruW Banzhaf, JR Koza, C Ryan, L Spector, C Jacob - 1998 - bib.tiera.ru

[7] Genetic programming and redundancy T Blickle, L Thiele - 1994 – Citeseer

[8] Discovery of subroutines in genetic programming JP Rosca, DH Ballard - 1996 – Citeseer.

[9] Genetic programming: A paradigm for genetically breeding populations of computer programs to solve problems. JR Koza – Soucek and the IRIS Group – Citeseer

[10] Subtree crossover: Building block engine or macromutation PJ Angeline - Genetic Programming, 1997.

[11] Fitnress distance correlation in structural mutation genetic programming L Vanneschi, M Tomassini, P Collard, M Clergue - Genetic Programming, 2003 – Springer

[12] Survey of genetic algorithms and genetic programming JR Koza – Technology Producing Quality Products Mobile 2002

Prof.T.Venkat Narayana Rao, received B.E in Computer Technology and Engineering from Nagpur University, Nagpur, India, M.B.A (Systems) and M.Tech in computer Science from Jawaharlal Nehru Technological University, Hyderabad, A.P., India and a Research Scholar in JNTU. He has 20 years of vast experience in Computer Science and Engineering areas pertaining to academics and industry related I.T issues. He is presently Professor and Head, Department of Computer Science and Engineering, Hyderabad Institute of Technology and management, Gowdavally, R.R.Dist., A.P,INDIA. He is nominated as an Editor and Reviewer for 15 International journals relating to Computer Science and Information Technology. He is currently working on research areas which include Digital Image Processing, Digital Watermarking, Data Mining and Network Security and other Emerging areas of Information Technologies. He can be reached at tvnrbobby@yahoo.com

Srikanth Madiraju, pursuing B.Tech final year from Hyderabad Institute of Technology and Management, gowdawalli, RR dist**., Jawaharlal** Nehru Technological University, Hyderabad, A.P, and India. He has organized national level tech-cultural festivals (iconix and esparto) during the year 2008 and 2009. He is life member and college representative for STED (society for triggering engineer's development, a jntu student chapter). In addition to this he also won many paper presentation events at inter-collegiate level and an active member of WWF CLUB (world wildlife federation**).** He can be reached at **mdrjsrikanth@yahoo.com**