UNIT II SEARCH METHODS AND VISUALIZATION 9

Search by simulated Annealing – Stochastic, Adaptive search by Evaluation – Evaluation Strategies –Genetic Algorithm – Genetic Programming – Visualization – Classification of Visual Data Analysis Techniques – Data Types – Visualization Techniques – Interaction techniques – Specific Visual data analysis Techniques

GENETIC ALGORITHM

Genetic Algorithms (GAs), originally introduced by John Holland have been popularized as universal optimization algorithms based on evolutionary methods. The most remarkable difference between Evolution Strategies and Genetic Algorithms is that GAs use a string-based, usually binary parameter encoding, resembling the discrete nucleotide coding scheme on cellular chromosomes. Therefore, GA genotypes can be defined as bit-vectors (bit-strings), on which point mutations are defined by switching bits with a certain probability. A recombinational operator of crossover, models the breaking of two chromosomes and subsequent crosswise restituation. Each individual is assigned a fitness value depending on the optimization task to be solved. Selection within the population is performed in a fitness-proportionate way: The more fit an individual, the more likely it is to be chosen for reproduction into the following generation.

Representation of Genotypes:

Nature encodes "growth programs", which describe the development of organisms, on the basis of a four-element alphabet, called the nucleotide bases A, C, G, and T on the DNA (deoxyribo nucleic acid) or A, C, G, and U on the RNA (ribonucleic acid) strands comprising the cellular genome. Binary alphabets are the preferred encoding used for Genetic Algorithm chromosomes.

Haploid GA Chromosomes

In its simplest form, a GA chromosome can be described as a haploid string, i.e., a single strand with alleles(one of two or more alternative forms of a gene that arise by mutation and are found at the same place on a chromosome.) over a ***k***-element alphabet $A = \{a_i, \dots, a_k\}$- Therefore, we define a single GA chromosome of length n as a vector g of the form

$$g = (g_1, \dots, g_n) \text{ with } g_i \in A \text{ for } 1 \leq i \leq n$$

Such a GA chromosome can be interpreted as a sequence of genes. Each gene is represented by its allele which exhibits a specific value from a discrete pool of possible settings A. Each gene locus i does not have its own alphabet A_i. Instead, all genes take their values from the same allele pool A.

Diploid and m-ploid GA Chromosomes

The notion of a single chromosome strand is easily extended to a polyploid chromosome with several strands. a diploid pair of chromosomes is represented as ***g_1 = (g_{11},... , g_{1n}) and g_2 = (g_{21},... , g_{2n})*** by the following structure.

$$\boldsymbol{g} = (\boldsymbol{g}_1, \boldsymbol{g}_2)^T = ((1, (g_{11}, g_{21})), \ldots, (n, (g_{1n}, g_{2n}))).$$

For each locus i the pair ***(i, (g_{1i} , g_{2i}))*** contains the locus index as its first component, the list of respective alleles of two chromosome strands gi and g2, each with n components as second. Polyploid or m-ploid chromosomes with m homologous strands, i.e., strands of equal length n, as follows:

$$g = ((1, (g_{11}, \ldots, g_{m1})), \ldots, (n, (g_{1n}, \ldots, g_{mn}))).$$

Mutations on GA Chromosomes:

Mutations in combination with recombinations are the driving forces of evolution. We first consider simple point-mutations on a GA chromosome ***g = (g_1 ,... , g_n)*** as taken from a discrete alphabet ***A = {a_1,..., a_k}***. Point mutations change the settings of genes at randomly selected gene locations. For each gene, its allele is replaced by a new value from A with mutation probability ***p_m***. The mutation operator **ω_{mut}** :=$G_A \rightarrow G_A$, with G_A denoting the set of all GA chromosomes over alphabet A, generates a new chromosome ***g′ = ω_{mut} (g) = ω_{mut} ((g_1 ,... , g_n))= (g_1' ,... , g_n')*** as follows:

$$g_i' = \begin{cases} x \in A - \{g_i\} & : \quad \chi \leq p_m \\ g_i & : \quad \text{otherwise} \end{cases}$$

Here χ is a uniformly distributed random variable from the interval [0,1]. The probability for a mutation per gene locus is denoted by **p_m**

Parent chromosome:	1 1 0 0 0 1 0 0 0 1
Mutant 1, p_m = 1.0:	0 0 1 1 1 0 1 1 1 0
Mutant 2, p_m = 0.5:	0 1 1 0 0 1 1 1 1 0
Mutant 3, p_m = 0.2:	1 1 1 0 1 1 0 0 0 1
Mutant 4, p_m = 0.1:	1 1 0 0 0 1 0 1 0 1

Mutations on a polyploid set of single-strand chromosomes is defined as,

$$\boldsymbol{g}^{(m)} = ((g_{11}, \ldots, g_{m1}), \ldots, (g_{1n}, \ldots, g_{mn}))$$

consisting of m homologous single chromosome strands of the form

g_i = (g_{i1},... , g_{in}) with $1 \leq i \leq n$

Recombinations among GA Chromosomes

GA chromosomes are mostly defined over a discrete allele alphabet, all the discrete variants of recombination operators for Evolution Strategies can be used here as well. For m homologous, haploid GA chromosomes $\boldsymbol{g_1 = (g_{11},\dots , g_{1n}) , \dots , g_m = (g_{m1},\dots , g_{mn})}$ a recombined GA chromosome ^rec can be composed with the help of a recombination mask **$\mu = (\mu_1, \mu_2 ,\dots , \mu_n)$**

$$g_{\mathbf{rec}} = (g_{\mu_1 1}, \dots, g_{\mu_n n}) \quad \text{with} \quad \mu_i \in \{1, \dots, m\}, 1 \le i \le n.$$

The i-th component of $\boldsymbol{g_{rec}}$ is therefore the **μ_i**-th element from the set of "genes" $\boldsymbol{\{g_{11},\dots , g_{1n}\}}$
The GA recombination operator - binary GA crossover plays an important role in the early for mutations and implementations of Genetic Algorithms.

One-Point crossover:

If the recombination mask **μ** has the special property which contains only two different elements, **i_1** and **i_2**, from the index set **$\{1 , \dots , m\}$** and there is exactly one index **$k \in \{1,\dots ,n - 1\}$** such that **$(\mu_1, \mu_2 ,\dots , \mu_k) = i_1$ and $(\mu_{k+1}, \mu_{k+2} ,\dots , \mu_n) = i_2$.**

Different types of cross over are:

Single-point and two-point crossover

Parametrized uniform crossover operators - similar to global ES recombination schemes.

Segmented and shuffle crossover work in a way similar to multi-point crossover. **Punctuated crossover** is one of the very few attempts to introduce self-adaptive strategy parameters into Genetic Algorithms

GA Operators

The two major GA operators, recombination as the primary and mutation as the secondary operator. Some more operators such as inversion, duplication, and deletion are also used.

We assume a haploid GA chromosome of the form

$$\boldsymbol{g} = (g_1, \dots, g_{i_1}, \dots, g_{i_2}, \dots, g_n) \quad \text{with} \quad 1 \le i_1 < i_2 \le n,$$

Inversion

On natural genomes, an inversion occurs when a chromosome is split twice, after locus **i_1 - 1** and before **i_2 + 1**, and the resulting middle section ***(g_{i1},... , g_{i2})*** is reinserted into the strand in reverse order:

$$g_{\text{inv}} = (g_1, \ldots, g_{i_2}, g_{i_2-1}, \ldots, g_{i_1+1}, g_{i_1}, \ldots, g_n)$$

Duplication

A side effect of crosswise recombination of two homologous chromosomes is the duplication and deletion of subsections on the genome. Formally, the insertion of a copy of a gene sequence ***(g_{i1},... , g_{i2})*** can be defined as follows:

$$g_{\text{dup}} = (g_1, \ldots, g_{i_1}, \ldots, g_{i_2}, g_{i_1}, \ldots, g_{i_2}, \ldots, g_n).$$

Deletion

The duplication operator is mainly applied in conjunction with its counterpart, the deletion operator. Here a chromosome loses a gene sequence ***(g_{i1},... , g_{i2}):***

$$g_{\text{del}} = (g_1, \ldots, g_{i_1-1}, g_{i_2+1}, \ldots, g_n).$$

Selection Functions

The selection functions define the "who shall live and who shall die" filters that all individuals pass through from one generation to the next. The Evolution Strategy's traditional selection procedure is done for survival of the best, but Genetic Algorithms apply a more "natural" selection scheme. In nature, an individual's probability of survival is influenced by an abundance of factors. However, it is not at all the case that only "the most fit" individuals survive. In fact, even less adapted, hence less fit, individuals have a chance to reproduce and transfer their genes to their progeny. Their genes survive into the next generations with certain probability.

Fitness Proportionate Selection:

For a population **$G = \{g_1, \ldots, g_\mu\}$** of size **$\mu$** with each individual **g_i** assigned a non-negative fitness value **$\eta(g_i)$** ∈ Kj, the probability **$Psel(\sigma_{prop}, g_i)$** for an individual to be selected fitness-proportionately is defined as

$$P_{\text{sel}}(\sigma_{\text{prop}}, g_i) = \frac{\eta(g_i)}{\eta_\Sigma(G)} \quad \text{with} \quad \eta_\Sigma(G) = \sum_{g \in G} \eta(g).$$

Here **$\eta_\Sigma(G)$** denotes the sum of all fitnesses of the population. The probability of an individual to be reproduced into the next generation is directly proportional to its fitness, hence it is fitness proportionate selection.

Rank-Based Selection

If the population size is small (much less than 100 individuals), with only these few super-individuals. Thus the gene pool loses its heterogeneity and is reduced to only a small set of search points. One remedy for this situation is to reduce the fitness differences among the individuals by assigning ranks to the individuals instead of using their actual fitness values. Just as in a tournament of **μ** competitors, the winner receives rank 1, the second best is assigned rank 2, etc., until the least fit individual ranks at **μ**.
The **μ** individuals are sorted in increasing fitness order such that $\eta(g_i) \leq \eta(g_j)$ for all $1 \leq i < j \leq \mu$. If $\rho(g)$ denotes the rank of individual $g \in G$ within this sorted sequence, its fitness $\eta_{rank}(g)$ is defined by

$$\eta_{\text{rank}}(\boldsymbol{g}) = \mu - \rho(\boldsymbol{g}) + 1$$

so that the rank-based selection probability is

$$P_{\text{sel}}(\sigma_{\text{rank}}, \boldsymbol{g}) = \frac{\eta_{\text{rank}}(\boldsymbol{g})}{\sum_{i=1}^{\mu} i}$$

Elitist Selection

The elitist selection scheme is the one used for selection with Evolution Strategies and can be used for Genetic Algorithms, too. For the **(μ, λ)** or **(μ + λ)** strategies, the best **μ** individuals from the set of mutants in the pool of **λ** or **μ + λ** individuals are selected as the parents of the next generation. The GA elitist selection scheme implements exactly this selection method.

$$P_{\text{sel}}(\sigma_{\text{elitist}}, \boldsymbol{g}) = \begin{cases} \frac{1}{\delta} & : \quad \boldsymbol{g} \in \text{Best}_\delta(G) \\ 0 & : \quad \text{otherwise} \end{cases}$$

The best **δ** individuals (usually $\delta \ll \mu$) are selected, each is assigned the same fitness of 1/**δ**, and fitness proportionate selection is performed, which results in a random selection among these **δ** individuals.

Random Selection

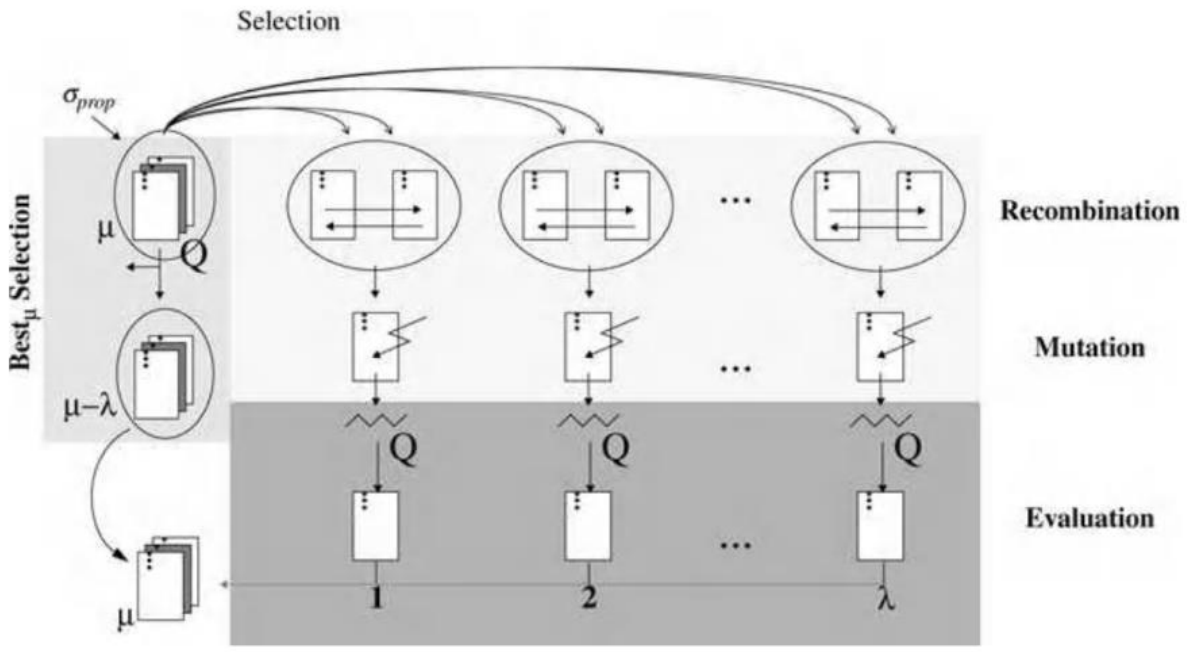
Sometimes, individuals $g \in G$ are selected by pure random choice,

$$P_{\text{sel}}(\sigma_{\text{random}}, \boldsymbol{g}) = \frac{1}{\mu},$$

Here **μ** denotes the number of individuals in generation G.

GA Evolution Scheme

GA evolution starts with a randomly generated initial population of **μ** genotypic structures. After interpretation and evaluation, the population enters a selection-variation cycle which is iterated for either a maximum number of generations or until some termination criterion (τ) is met.



The canonical Genetic Algorithm performs a (**μ**/2, **μ**) strategy. From the pool of **μ** parents **λ** = **μ** pairs of individuals are selected for recombination (crossover) and subsequent mutation. The resulting individuals represent the parents of the next generation. The individuals are not selected at random from the parent pool but according to one of the GA selection functions σ.

GENETIC PROGRAMMING

Genetic Programming (GP) is certainly one of the major steps towards automatic programming using evolutionary principles. The term genetic programming was coined by John Koza, who initially introduced his evolutionary programming approach as a "hierarchical genetic algorithm" and later on switched to a symbolic representation for evolving LISP programs. The main contribution of Koza's research group is documented by a three-volume treatise, demonstrating that Genetic Programming can successfully be applied to induce computer programs in a wide range of areas, such as symbolic regression or learning boolean parity functions, the evolution of emergent behavior, the evolution of robot control programs, the evolution of classifiers for prediction of transmembrane domains and omega loops in proteins, or the recent work on evolution of analog electrical circuits. The successful use of tree structures to represent data and program instructions has led to an immediate association of Genetic Programming with symbolic expressions, although many more encoding schemes can be used and actually are used for automatic programming by evolution. The following items characterize the field of Genetic Programming in a broader sense:

Program Induction: Genetic Programming deals with the induction of computer programs using evolutionary principles. The programs can be either directly executable (machine) code or expressions — data and/or instructions — of any programming language.

Learning Algorithms: Not all learning algorithms are explicitly represented as programs in a strict sense, such as neural networks and learning fuzzy systems. Algorithms for adapting these data structures (neuron weights, activation functions, fuzzy rule parameters, etc.) are also in the domain of Genetic Programming.

Representation: There are many different ways of representing programs, for example, by linear, string-based structures, by tree-like symbolic expressions, by growth grammars, or by graphs.

Operators: Selection operators are used to designate the survivors among the population of competing programs. Reproduction operators are used in conjunction with recombination operators, generating new variants by, primarily, mutation and crossover, or further operators, such as permutation, deletion, duplication, or encapsulation.

Representation of Computer Programs by Symbolic Expressions

Programs evolved by a tree-based GP system are typically composed from a finite set F of building blocks, the functions

$$F = \{f_1, \ldots, f_N\}, \quad \text{with} \quad \nu(F) = \{\nu(f_1), \ldots, \nu(f_N)\},$$

where $\upsilon(F)$ denotes the arities of the function symbols, and the terminals

$$\mathbf{T=\{t_1, t_2, \ldots, t_M\}}$$

Considering the terminals as function symbols of arity zero, both sets can be merged into a single set of elementary building blocks:

$$\mathbf{S = F \cup T}$$

The set $GPterm_s$ of program trees, representing the GP search space, can be defined as follows:

1. Each terminal $t \in S$ with $\upsilon(t) = 0$ is an element of $GPterm_s$

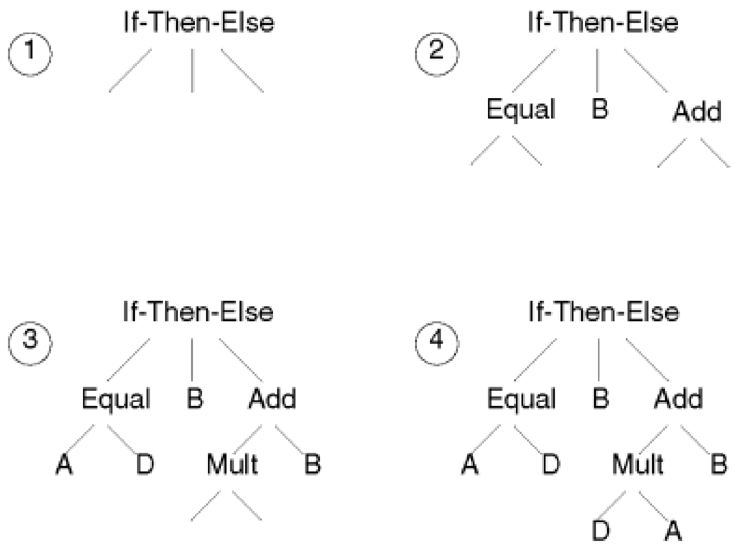
2. For each $f \in S$ with $\upsilon(f) = n$ and $g_1,\ldots,g_n \in \mathtt{GPterm}_s$, the term $f(g_1,\ldots,g_n)$ is also an element of $GPterm_s$

The GP encoding of the structures to be evolved has an important impact on whether the evolutionary approach will succeed or not. The choice of functions and terminals also influences the potential of the genetic operators to create innovative and optimized program structures.

Given the set of building blocks

$$S = \{Mult_2, Add_2, If\text{-}Then\text{-}Else_3, Equal_2, A_0, B_0, C_0, D_0\}$$

Where indices in S denote the arities of the symbols.



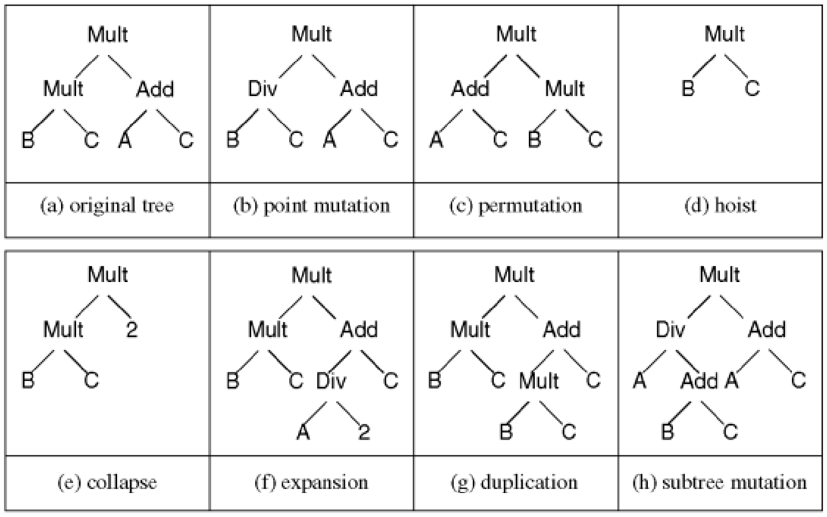
The above Figure illustrates the recursive, step-by-step construction of a random program term, an element of GPterm$_S$- A symbol from S is randomly selected (If-Then-Else) as the root of the expression tree. At each of the branches, further subterms have to be composed by further random selection from S. This procedure is repeated until all leaf nodes are labeled with terminals.

The depth of the generated expression tree is not constrained, although one usually defines a maximum tree depth and width in order to reduce memory requirements for storage and evaluation time of the program term. Furthermore, the symbol set S has to be closed with respect to composition. Each symbol must be combinable with any other symbol, so that the final expression is always syntactically correct and can be interpreted as a proper program or data structure.

$$S = \{Mult_{(int,int)\rightarrow int}, If\text{-}Then\text{-}Else_{(bool,int,int)\rightarrow int}, Equal_{(int,int)\rightarrow bool}, \ldots\} .$$

Mutations on Symbolic Expressions

The GP mutation operators come in a great number of varieties. Basically, subterms or leaves of a tree structure are substituted by either newly generated or duplicated terms or terminals.



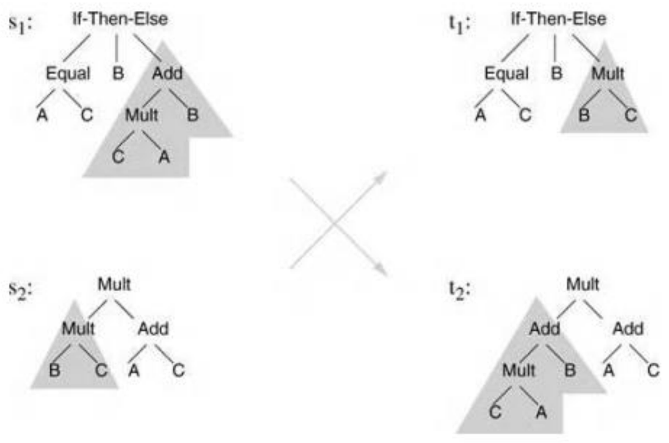
(a) original tree (b) point mutation (c) permutation (d) hoist (e) collapse (f) expansion (g) duplication (h) subtree mutation

The above figure gives a brief overview of mutation operators on tree structured expressions.

- **Point Mutation** A point mutation (b) exchanges a single node by a random node of the same class. In the simplest case this means that terminals are substituted by terminals, and function symbols are substituted by function symbols with the same arity (and types, if applicable).
- **Permutation** A permutation (c) merely permutes the sequence of arguments of a node. The hoist operator (d) substitutes the whole tree by a randomly selected proper subtree (terminals are not in the scope of selection).
- **Collapse Subtree Mutation** With the collapse subtree mutation (e) a subtree is replaced by a random terminal.
- **Expansion Mutation** The inverse operation is the expansion mutation (f) where a terminal is exchanged against a random, newly generated subtree.
- **Duplication** The duplication operator (g) replaces a terminal node with a copy of a proper subtree.
- **Subtree Mutation** The most general mutation operator is defined by subtree mutation (h), where a subtree is substituted with a newly generated subtree.

Crossover of Symbolic Expressions

Another important operator used in GP for generating new term structures is a variant of the one-point crossover known from Genetic Algorithms. By crossing two linear GA chromosomes, substrings are exchanged between the chromosomes. An analogous recombination operator for GP terms is defined by interchanging (sub)trees between two GP terms.



The above figure shows the simplest crossover version known as subtree exchange crossover, which is performed as follows: For each term a node is chosen at random. Inner nodes (including the root) as well as leaves are selectable. In Figure, the selected subtrees are marked by triangular shapes. The two recombined terms result from a mutual exchange of the selected sub-expressions.

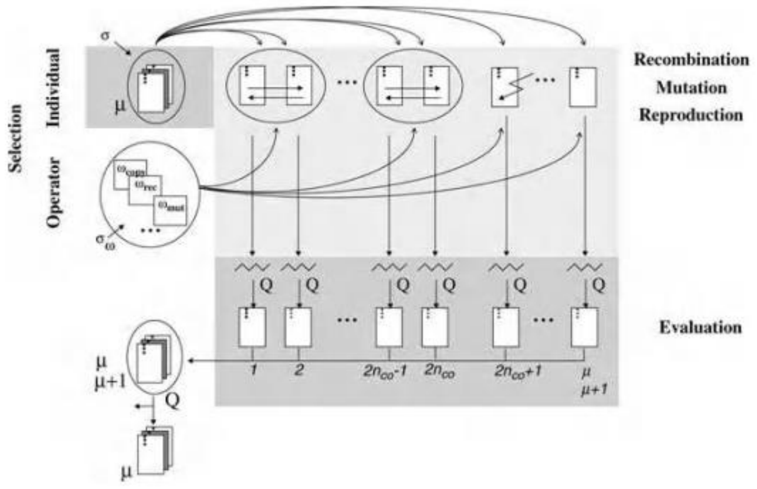
In comparison to the crossover or recombination operators of Genetic Algorithms or Evolution Strategies, an interesting aspect of GP crossover is the following. Even for a recombination among two identical trees, the GP crossover (self crossover) results in a pair of new structures whenever the two crossover nodes differ.

GA crossover for identical chromosomes is reduced to a simple reproduction operator without changing the structures. As the GA crossover operator enhances the similarity among the strings, mutation operators are essentially needed to introduce new allele settings into the genepool.

Several variants for GP crossover have been developed, such as **context preserving crossover (CPC)**, where subtrees are exchanged only if either their node coordinates match exactly (strong CPC) or match approximately (weak CPC). With **module crossover**, parametrized sub-trees are exchanged between two individual structures. The modules are comparable to parameterized macros in programming languages like C. One variant of module crossover is known as **encapsulation** and **decapsulation**. An extension of this approach denoted as **macro extraction.**

G P Evolution Scheme

For canonical Genetic Programming a slightly modified GA evolution scheme is used. The scheme depicted in the below figure is a slight modification of the basic GP algorithm and is better compared to the GA and ES schemes. The comma as well as the plus reproduction scheme can be used.



The above figure depicts the comma scheme where parents are not part of the selection pool. A notable extension of the GP evolution scheme is the operator pool, which contains a set of genetic operators, each attributed by a selection probability.

One reproduction cycle works as follows:

- First, a genetic operator is selected from the operator pool.
- Secondly, depending on the arity n of the operator, n individuals are chosen by a fitness-proportionate selection function (n = 1 for reproduction and mutation, n > 2 for crossover).
- After applying the operators, the new program structures are evaluated and constitute the selection pool for the parents of the following generation.